CONTENTS

1 Cg Language Specification ........................................ 3
  1.1 Cg Language Specification ........................................ 3

2 Cg Commands .......................................................... 27
  2.1 cgc ................................................................. 27
  2.2 cgfxcat ............................................................ 39
  2.3 cginfo ............................................................... 40

3 Cg Profiles ............................................................. 41
  3.1 arbfp1 ............................................................... 41
  3.2 arbvp1 ............................................................... 43
  3.3 ds_5_0 ............................................................... 44
  3.4 fp20 ................................................................. 46
  3.5 fp30 ................................................................. 52
  3.6 fp40 ................................................................. 54
  3.7 glsIf ................................................................. 56
  3.8 glslg ................................................................. 58
  3.9 glsl ................................................................. 59
  3.10 glslv ............................................................... 60
  3.11 gp4fp ............................................................... 62
  3.12 gp4gp ............................................................... 69
  3.13 gp4 ................................................................. 77
  3.14 gp4vp ............................................................... 78
  3.15 gp5fp ............................................................... 84
  3.16 gp5gp ............................................................... 84
  3.17 gp5 ................................................................. 85
  3.18 gp5tcp .............................................................. 86
  3.19 gp5tep .............................................................. 87
  3.20 gp5vp .............................................................. 89
  3.21 gs_4_0 .............................................................. 90
  3.22 gs_5_0 .............................................................. 92
  3.23 hsl110 ............................................................. 93
  3.24 hsl111 ............................................................ 94
  3.25 hslIf .............................................................. 94
  3.26 hslsv ............................................................. 97
  3.27 hs_5_0 ............................................................. 98
  3.28 ps_1_1 ............................................................. 100
  3.29 ps_1_2 ............................................................. 101
  3.30 ps_1_3 ............................................................. 102
<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgGetFirstLeafEffectParameter</td>
<td>223</td>
</tr>
<tr>
<td>cgGetFirstError</td>
<td>223</td>
</tr>
<tr>
<td>cgGetFirstEffectParameter</td>
<td>222</td>
</tr>
<tr>
<td>cgGetFirstEffectAnnotation</td>
<td>221</td>
</tr>
<tr>
<td>cgGetFirstEffect</td>
<td>220</td>
</tr>
<tr>
<td>cgGetFirstDependentParameter</td>
<td>219</td>
</tr>
<tr>
<td>cgGetErrorString</td>
<td>219</td>
</tr>
<tr>
<td>cgGetErrorHandler</td>
<td>218</td>
</tr>
<tr>
<td>cgGetErrorCallback</td>
<td>217</td>
</tr>
<tr>
<td>cgGetError</td>
<td>216</td>
</tr>
<tr>
<td>cgGetEnum</td>
<td>215</td>
</tr>
<tr>
<td>cgGetEffectParameterBySemantic</td>
<td>214</td>
</tr>
<tr>
<td>cgGetEffectParameterBuffer</td>
<td>213</td>
</tr>
<tr>
<td>cgGetEffectName</td>
<td>212</td>
</tr>
<tr>
<td>cgGetEffect</td>
<td>211</td>
</tr>
<tr>
<td>cgGetAutoCompile</td>
<td>210</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>209</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>208</td>
</tr>
<tr>
<td>cgGetArrayTotalSize</td>
<td>207</td>
</tr>
<tr>
<td>cgGetArrayType</td>
<td>206</td>
</tr>
<tr>
<td>cgGetArrayParameterBySemantic</td>
<td>205</td>
</tr>
<tr>
<td>cgGetArrayIndex</td>
<td>204</td>
</tr>
<tr>
<td>cgGetAnnotationType</td>
<td>203</td>
</tr>
<tr>
<td>cgGetAnnotationName</td>
<td>202</td>
</tr>
<tr>
<td>cgGetAnnotationValues</td>
<td>201</td>
</tr>
<tr>
<td>cgGetBufferSize</td>
<td>200</td>
</tr>
<tr>
<td>cgGetCompilerIncludeCallback</td>
<td>201</td>
</tr>
<tr>
<td>cgGetConnectedParameter</td>
<td>200</td>
</tr>
<tr>
<td>cgGetConnectedStateAssignmentParameter</td>
<td>199</td>
</tr>
<tr>
<td>cgGetConnectedStateAssignmentToParameter</td>
<td>198</td>
</tr>
<tr>
<td>cgGetContextBehavior</td>
<td>197</td>
</tr>
<tr>
<td>cgGetDefaultAnnotationParameter</td>
<td>196</td>
</tr>
<tr>
<td>cgGetDefaultAnnotationParameters</td>
<td>195</td>
</tr>
<tr>
<td>cgGetWidth</td>
<td>194</td>
</tr>
<tr>
<td>cgGetArrayType</td>
<td>193</td>
</tr>
<tr>
<td>cgGetArrayTotalSize</td>
<td>192</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>191</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>190</td>
</tr>
<tr>
<td>cgGetArraySize</td>
<td>189</td>
</tr>
<tr>
<td>cgGetArrayType</td>
<td>188</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>187</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>186</td>
</tr>
<tr>
<td>cgGetArrayTotalSize</td>
<td>185</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>184</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>183</td>
</tr>
<tr>
<td>cgGetArraySize</td>
<td>182</td>
</tr>
<tr>
<td>cgGetArrayType</td>
<td>181</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>180</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>179</td>
</tr>
<tr>
<td>cgGetArrayTotalSize</td>
<td>178</td>
</tr>
<tr>
<td>cgGetArrayDimension</td>
<td>177</td>
</tr>
<tr>
<td>cgGetArrayParameter</td>
<td>176</td>
</tr>
</tbody>
</table>

---
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.91</td>
<td>cgGetFirstLeafParameter</td>
</tr>
<tr>
<td>4.92</td>
<td>cgGetFirstParameter</td>
</tr>
<tr>
<td>4.93</td>
<td>cgGetFirstParameterAnnotation</td>
</tr>
<tr>
<td>4.94</td>
<td>cgGetFirstPass</td>
</tr>
<tr>
<td>4.95</td>
<td>cgGetFirstPassAnnotation</td>
</tr>
<tr>
<td>4.96</td>
<td>cgGetFirstProgram</td>
</tr>
<tr>
<td>4.97</td>
<td>cgGetFirstProgramAnnotation</td>
</tr>
<tr>
<td>4.98</td>
<td>cgGetFirstSamplerState</td>
</tr>
<tr>
<td>4.99</td>
<td>cgGetFirstSamplerStateAssignment</td>
</tr>
<tr>
<td>4.100</td>
<td>cgGetFirstState</td>
</tr>
<tr>
<td>4.101</td>
<td>cgGetFirstStateAssignment</td>
</tr>
<tr>
<td>4.102</td>
<td>cgGetFirstStructParameter</td>
</tr>
<tr>
<td>4.103</td>
<td>cgGetFirstTechnique</td>
</tr>
<tr>
<td>4.104</td>
<td>cgGetFirstTechniqueAnnotation</td>
</tr>
<tr>
<td>4.105</td>
<td>cgGetFirstUniformBufferParameter</td>
</tr>
<tr>
<td>4.106</td>
<td>cgGetFloatAnnotationValues</td>
</tr>
<tr>
<td>4.107</td>
<td>cgGetFloatStateAssignmentValues</td>
</tr>
<tr>
<td>4.108</td>
<td>cgGetIntAnnotationValues</td>
</tr>
<tr>
<td>4.109</td>
<td>cgGetIntStateAssignmentValues</td>
</tr>
<tr>
<td>4.110</td>
<td>cgGetLastErrorString</td>
</tr>
<tr>
<td>4.111</td>
<td>cgGetLastListing</td>
</tr>
<tr>
<td>4.112</td>
<td>cgGetLockingPolicy</td>
</tr>
<tr>
<td>4.113</td>
<td>cgGetMatrixParameter</td>
</tr>
<tr>
<td>4.114</td>
<td>cgGetMatrixParameterdc</td>
</tr>
<tr>
<td>4.115</td>
<td>cgGetMatrixParameterdr</td>
</tr>
<tr>
<td>4.116</td>
<td>cgGetMatrixParameterfc</td>
</tr>
<tr>
<td>4.117</td>
<td>cgGetMatrixParameterfr</td>
</tr>
<tr>
<td>4.118</td>
<td>cgGetMatrixParameteric</td>
</tr>
<tr>
<td>4.119</td>
<td>cgGetMatrixParameterir</td>
</tr>
<tr>
<td>4.120</td>
<td>cgGetMatrixParameterOrder</td>
</tr>
<tr>
<td>4.121</td>
<td>cgGetMatrixSize</td>
</tr>
<tr>
<td>4.122</td>
<td>cgGetNamedEffect</td>
</tr>
<tr>
<td>4.123</td>
<td>cgGetNamedEffectAnnotation</td>
</tr>
<tr>
<td>4.124</td>
<td>cgGetNamedEffectParameter</td>
</tr>
<tr>
<td>4.125</td>
<td>cgGetNamedEffectUniformBuffer</td>
</tr>
<tr>
<td>4.126</td>
<td>cgGetNamedParameter</td>
</tr>
<tr>
<td>4.127</td>
<td>cgGetNamedParameterAnnotation</td>
</tr>
<tr>
<td>4.128</td>
<td>cgGetNamedPass</td>
</tr>
<tr>
<td>4.129</td>
<td>cgGetNamedPassAnnotation</td>
</tr>
<tr>
<td>4.130</td>
<td>cgGetNamedProgramAnnotation</td>
</tr>
<tr>
<td>4.131</td>
<td>cgGetNamedProgramParameter</td>
</tr>
<tr>
<td>4.132</td>
<td>cgGetNamedProgramUniformBuffer</td>
</tr>
<tr>
<td>4.133</td>
<td>cgGetNamedSamplerState</td>
</tr>
<tr>
<td>4.134</td>
<td>cgGetNamedSamplerStateAssignment</td>
</tr>
<tr>
<td>4.135</td>
<td>cgGetNamedState</td>
</tr>
<tr>
<td>4.136</td>
<td>cgGetNamedStateAssignment</td>
</tr>
<tr>
<td>4.137</td>
<td>cgGetNamedStructParameter</td>
</tr>
<tr>
<td>4.138</td>
<td>cgGetNamedSubParameter</td>
</tr>
<tr>
<td>4.139</td>
<td>cgGetNamedTechnique</td>
</tr>
<tr>
<td>4.140</td>
<td>cgGetNamedTechniqueAnnotation</td>
</tr>
<tr>
<td>4.141</td>
<td>cgGetNamedUniformBufferParameter</td>
</tr>
<tr>
<td>4.142</td>
<td>cgGetNamedUserType</td>
</tr>
<tr>
<td>4.143</td>
<td>cgGetNextAnnotation</td>
</tr>
<tr>
<td>4.144</td>
<td>cgGetNextEffect</td>
</tr>
</tbody>
</table>
4.145 cgGetNextLeafParameter ........................................ 273
4.146 cgGetNextParameter ........................................... 274
4.147 cgGetNextPass ................................................ 275
4.148 cgGetNextProgram ............................................. 276
4.149 cgGetNextState ................................................ 277
4.150 cgGetNextStateAssignment ................................. 278
4.151 cgGetNextTechnique ........................................ 279
4.152 cgGetNumConnectedToParameters ......................... 280
4.153 cgGetNumDependentAnnotationParameters .......... 281
4.154 cgGetNumDependentProgramArrayStateAssignmentParameters 281
4.155 cgGetNumDependentStateAssignmentParameters .... 283
4.156 cgGetNumParentTypes ..................................... 284
4.157 cgGetNumProgramDomains ................................ 285
4.158 cgGetNumStateEnumerants ................................ 286
4.159 cgGetNumSupportedProfiles ................................ 287
4.160 cgGetNumUserTypes ......................................... 288
4.161 cgGetParameterBaseResource ............................ 289
4.162 cgGetParameterBaseType ................................ 290
4.163 cgGetParameterBufferIndex ............................... 290
4.164 cgGetParameterBufferOffset ............................. 291
4.165 cgGetParameterClass ....................................... 292
4.166 cgGetParameterClassEnum ................................ 293
4.167 cgGetParameterClassString ................................ 294
4.168 cgGetParameterColumns ................................... 295
4.169 cgGetParameterContext .................................... 296
4.170 cgGetParameterDefaultValue .............................. 297
4.171 cgGetParameterDefaultValuedc ......................... 298
4.172 cgGetParameterDefaultValuedr ......................... 299
4.173 cgGetParameterDefaultValuefc ......................... 301
4.174 cgGetParameterDefaultValuefr ......................... 302
4.175 cgGetParameterDefaultValueic ......................... 303
4.176 cgGetParameterDefaultValueiir ........................ 304
4.177 cgGetParameterDirection ................................ 306
4.178 cgGetParameterEffect ...................................... 307
4.179 cgGetParameterIndex ...................................... 307
4.180 cgGetParameterName ....................................... 308
4.181 cgGetParameterNamedType ................................. 309
4.182 cgGetParameterOrdinalNumber ......................... 310
4.183 cgGetParameterProgram .................................. 311
4.184 cgGetParameterResource .................................. 312
4.185 cgGetParameterResourceIndex ......................... 312
4.186 cgGetParameterResourceName ......................... 313
4.187 cgGetParameterResourceSize ............................ 314
4.188 cgGetParameterResourceType ............................ 315
4.189 cgGetParameterRows ....................................... 316
4.190 cgGetParameterSemantic ................................ 317
4.191 cgGetParameterSettingMode .............................. 318
4.192 cgGetParameterType ........................................ 319
4.193 cgGetParameterValue ...................................... 320
4.194 cgGetParameterValuedc ................................. 321
4.195 cgGetParameterValuedr ................................. 322
4.196 cgGetParameterValuefc ................................. 323
4.197 cgGetParameterValuefr ................................. 324
4.198 cgGetParameterValueic ................................. 325
4.199 cgGetParameterValuei
4.200 cgGetParameterValues
4.201 cgGetParameterVariability
4.202 cgGetParentType
4.203 cgGetPassName
4.204 cgGetPassProgram
4.205 cgGetPassTechnique
4.206 cgGetProfile
4.207 cgGetProfileDomain
4.208 cgGetProfileProperty
4.209 cgGetProfileSibling
4.210 cgGetProfileString
4.211 cgGetProgramBuffer
4.212 cgGetProgramBufferMaxIndex
4.213 cgGetProgramBufferMaxSize
4.214 cgGetProgramContext
4.215 cgGetProgramDomain
4.216 cgGetProgramDomainProfile
4.217 cgGetProgramDomainProgram
4.218 cgGetProgramInput
4.219 cgGetProgramOptions
4.220 cgGetProgramOutput
4.221 cgGetProgramOutputVertices
4.222 cgGetProgramProfile
4.223 cgGetProgramStateAssignmentValue
4.224 cgGetProgramString
4.225 cgGetResource
4.226 cgGetResourceString
4.227 cgGetSamplerStateAssignmentParameter
4.228 cgGetSamplerStateAssignmentState
4.229 cgGetSamplerStateAssignmentValue
4.230 cgGetSemanticCasePolicy
4.231 cgGetStateAssignmentIndex
4.232 cgGetStateAssignmentPass
4.233 cgGetStateAssignmentState
4.234 cgGetStateContext
4.235 cgGetStateEnumerant
4.236 cgGetStateEnumerantName
4.237 cgGetStateEnumerantValue
4.238 cgGetStateLatestProfile
4.239 cgGetStateName
4.240 cgGetStateResetCallback
4.241 cgGetStateSetCallback
4.242 cgGetStateType
4.243 cgGetStateValidateCallback
4.244 cgGetString
4.245 cgGetStringAnnotationValue
4.246 cgGetStringAnnotationValues
4.247 cgGetStringParameterValue
4.248 cgGetStringStateAssignmentValue
4.249 cgGetSupportedProfile
4.250 cgGetTechniqueEffect
4.251 cgGetTechniqueName
4.252 cgGetTextureStateAssignmentValue
4.253 cgGetUniformBufferBlockName ........................................... 375
4.254 cgGetUniformBufferParameter ............................................ 376
4.255 cgGetType ................................................................. 377
4.256 cgGetTypeBase ............................................................ 378
4.257 cgGetTypeClass ............................................................ 378
4.258 cgGetTypeSizes ............................................................ 379
4.259 cgGetTypeString .......................................................... 380
4.260 cgGetTypeUserType ......................................................... 381
4.261 cgIsAnnotation ............................................................. 382
4.262 cgIsBuffer ................................................................. 383
4.263 cgIsContext ............................................................... 384
4.264 cgIsEffect ................................................................. 384
4.265 cgIsInterfaceType ......................................................... 385
4.266 cgIsParameter ............................................................. 386
4.267 cgIsParameterGlobal ....................................................... 387
4.268 cgIsParameterReferenced .................................................. 388
4.269 cgIsParameterUsed ........................................................ 389
4.270 cgIsParentType ............................................................. 390
4.271 cgIsPass ................................................................. 391
4.272 cgIsProfileSupported ....................................................... 391
4.273 cgIsProgram .............................................................. 392
4.274 cgIsProgramCompiled ...................................................... 393
4.275 cgIsState ................................................................. 394
4.276 cgIsStateAssignment ....................................................... 395
4.277 cgIsTechnique ............................................................ 396
4.278 cgIsTechniqueValidated .................................................... 396
4.279 cgMapBuffer ............................................................. 397
4.280 cgResetPassState .......................................................... 399
4.281 cgSetArraySize ............................................................ 399
4.282 cgSetAutoCompile .......................................................... 401
4.283 cgSetBoolAnnotation ....................................................... 402
4.284 cgSetBoolArrayStateAssignment ....................................... 403
4.285 cgSetBoolStateAssignment ............................................... 404
4.286 cgSetBufferData ........................................................... 405
4.287 cgSetBufferSubData ....................................................... 406
4.288 cgSetCompilerIncludeCallback ....................................... 407
4.289 cgSetCompilerIncludeFile ............................................... 408
4.290 cgSetCompilerIncludeString ............................................ 408
4.291 cgSetContextBehavior ..................................................... 409
4.292 cgSetEffectName .......................................................... 411
4.293 cgSetEffectParameterBuffer ............................................ 412
4.294 cgSetErrorCallback ....................................................... 414
4.295 cgSetErrorHandler ......................................................... 415
4.296 cgSetFloatAnnotation ...................................................... 416
4.297 cgSetFloatArrayStateAssignment ..................................... 417
4.298 cgSetFloatStateAssignment ............................................. 418
4.299 cgSetIntAnnotation ........................................................ 419
4.300 cgSetIntArrayStateAssignment ......................................... 420
4.301 cgSetIntStateAssignment .................................................. 420
4.302 cgSetLastListing .......................................................... 421
4.303 cgSetLockingPolicy ......................................................... 422
4.304 cgSetMatrixParameter ...................................................... 423
4.305 cgSetMatrixParameterdc .................................................. 424
4.306 cgSetMatrixParameterdr .................................................. 425
5 CgGL API

5.1 cgBindProgram
5.2 cgCreateBufferFromObject
5.3 cgCreateBuffer
5.4 cgDetectGLSLVersion
5.5 cgDisableClientState
5.6 cgDisableProfile
5.7 cgDisableProgramProfiles
5.8 cgDisableTextureParameter
5.9 cgEnableClientState
5.10 cgEnableProfile
5.11 cgEnableProgramProfiles
5.12 cgEnableTextureParameter
5.13 cgGetBufferObject
5.14 cgGetContextGLSLVersion
5.15 cgGetContextOptimalOptions
5.16 cgGetGLSLVersion
5.17 cgGetGLSLVersionString
5.18 cgGetLatestProfile
5.19 cgGetManageTextureParameters
5.20 cgGetMatrixParameterArraydc
5.21 cgGetMatrixParameterArraydr
5.22 cgGetMatrixParameterArrayfc
5.23 cgGetMatrixParameterArrayfr
5.24 cgGetMatrixParameterArray
5.25 cgGetMatrixParameterd
5.26 cgGetMatrixParameterdr
5.27 cgGetMatrixParameterfc
5.28 cgGetMatrixParameterfr
5.29 cgGetMatrixParameter
5.30 cgGetOptimalOptions
5.31 cgGetParameter1d
5.32 cgGetParameter1f
5.33 cgGetParameter2d
5.34 cgGetParameter2f
5.35 cgGetParameter3d
5.36 cgGetParameter3f
5.37 cgGetParameter4d
5.38 cgGetParameter4f
5.39 cgGetParameterArray1d
5.40 cgGetParameterArray1f
5.41 cgGetParameterArray2d
5.42 cgGetParameterArray2f
5.43 cgGetParameterArray3d
5.44 cgGetParameterArray3f
5.45 cgGetParameterArray4d
5.46 cgGetParameterArray4f
5.101 cgGLUnbindProgram .................................................. 589
5.102 cgGLUnloadProgram .................................................. 590

6 CgD3D9 API ................................................................. 593
   6.1 cgD3D9BindProgram .................................................. 593
   6.2 cgD3D9EnableDebugTracing ......................................... 594
   6.3 cgD3D9EnableParameterShadowing ................................. 595
   6.4 cgD3D9GetDevice ..................................................... 596
   6.5 cgD3D9GetLastError ................................................ 597
   6.6 cgD3D9GetLatestPixelProfile ..................................... 597
   6.7 cgD3D9GetLatestVertexProfile ................................... 598
   6.8 cgD3D9GetManageTextureParameters .............................. 599
   6.9 cgD3D9GetOptimalOptions ......................................... 600
   6.10 cgD3D9GetTextureParameter ...................................... 601
   6.11 cgD3D9GetVertexDeclaration .................................... 601
   6.12 cgD3D9IsParameterShadowingEnabled ............................ 603
   6.13 cgD3D9IsProfileSupported ....................................... 603
   6.14 cgD3D9IsProgramLoaded .......................................... 604
   6.15 cgD3D9LoadProgram .............................................. 605
   6.16 cgD3D9RegisterStates ............................................ 606
   6.17 cgD3D9ResourceToDevice ........................................ 607
   6.18 cgD3D9SetDevice .................................................. 608
   6.19 cgD3D9SetManageTextureParameters ............................. 609
   6.20 cgD3D9SetSamplerState ......................................... 610
   6.21 cgD3D9SetTextureParameter ..................................... 612
   6.22 cgD3D9SetTexture ................................................ 612
   6.23 cgD3D9SetTextureWrapMode ..................................... 614
   6.24 cgD3D9SetUniformArray ......................................... 615
   6.25 cgD3D9SetUniformMatrixArray ................................ 616
   6.26 cgD3D9SetUniformMatrix ....................................... 618
   6.27 cgD3D9SetUniform ................................................ 619
   6.28 cgD3D9TranslateCGerror ......................................... 621
   6.29 cgD3D9TranslateHRESULT ......................................... 621
   6.30 cgD3D9TypeToSize ................................................. 622
   6.31 cgD3D9UnbindProgram ............................................ 623
   6.32 cgD3D9UnloadAllPrograms ....................................... 624
   6.33 cgD3D9UnloadProgram ............................................ 625
   6.34 cgD3D9ValidateVertexDeclaration .............................. 626

7 CgD3D10 API ................................................................. 629
   7.1 cgD3D10BindProgram ................................................ 629
   7.2 cgD3D10CreateBuffer .............................................. 630
   7.3 cgD3D10CreateBufferFromObject .................................. 631
   7.4 cgD3D10GetBufferByIndex ........................................ 632
   7.5 cgD3D10GetBufferObject ......................................... 633
   7.6 cgD3D10GetCompiledProgram ..................................... 633
   7.7 cgD3D10GetDevice .................................................. 634
   7.8 cgD3D10GetIASignatureByPass ................................... 635
   7.9 cgD3D10GetLastError ............................................. 636
   7.10 cgD3D10GetLatestGeometryProfile .............................. 637
   7.11 cgD3D10GetLatestPixelProfile .................................. 638
   7.12 cgD3D10GetLatestVertexProfile ................................ 638
   7.13 cgD3D10GetManageTextureParameters ............................ 639
   7.14 cgD3D10GetOptimalOptions ..................................... 640
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.15</td>
<td>cgD3D10GetProgramErrors</td>
<td>641</td>
</tr>
<tr>
<td>7.16</td>
<td>cgD3D10IsProfileSupported</td>
<td>642</td>
</tr>
<tr>
<td>7.17</td>
<td>cgD3D10IsProgramLoaded</td>
<td>643</td>
</tr>
<tr>
<td>7.18</td>
<td>cgD3D10LoadProgram</td>
<td>644</td>
</tr>
<tr>
<td>7.19</td>
<td>cgD3D10RegisterStates</td>
<td>645</td>
</tr>
<tr>
<td>7.20</td>
<td>cgD3D10SetDevice</td>
<td>646</td>
</tr>
<tr>
<td>7.21</td>
<td>cgD3D10SetManageTextureParameters</td>
<td>647</td>
</tr>
<tr>
<td>7.22</td>
<td>cgD3D10SetSamplerStateParameter</td>
<td>648</td>
</tr>
<tr>
<td>7.23</td>
<td>cgD3D10SetTextureParameter</td>
<td>649</td>
</tr>
<tr>
<td>7.24</td>
<td>cgD3D10SetTextureSamplerStateParameter</td>
<td>649</td>
</tr>
<tr>
<td>7.25</td>
<td>cgD3D10TranslateCGerror</td>
<td>650</td>
</tr>
<tr>
<td>7.26</td>
<td>cgD3D10TranslateHRESULT</td>
<td>651</td>
</tr>
<tr>
<td>7.27</td>
<td>cgD3D10TypeToSize</td>
<td>652</td>
</tr>
<tr>
<td>7.28</td>
<td>cgD3D10UnbindProgram</td>
<td>653</td>
</tr>
<tr>
<td>7.29</td>
<td>cgD3D10UnloadProgram</td>
<td>654</td>
</tr>
<tr>
<td>8</td>
<td>CgD3D11 API</td>
<td>657</td>
</tr>
<tr>
<td>8.1</td>
<td>cgD3D11BindProgram</td>
<td>657</td>
</tr>
<tr>
<td>8.2</td>
<td>cgD3D11CreateBuffer</td>
<td>658</td>
</tr>
<tr>
<td>8.3</td>
<td>cgD3D11CreateBufferFromObject</td>
<td>659</td>
</tr>
<tr>
<td>8.4</td>
<td>cgD3D11GetBufferByIndex</td>
<td>660</td>
</tr>
<tr>
<td>8.5</td>
<td>cgD3D11GetBufferObject</td>
<td>661</td>
</tr>
<tr>
<td>8.6</td>
<td>cgD3D11GetCompiledProgram</td>
<td>661</td>
</tr>
<tr>
<td>8.7</td>
<td>cgD3D11GetDevice</td>
<td>662</td>
</tr>
<tr>
<td>8.8</td>
<td>cgD3D11GetIASignatureByPass</td>
<td>663</td>
</tr>
<tr>
<td>8.9</td>
<td>cgD3D11GetLastError</td>
<td>664</td>
</tr>
<tr>
<td>8.10</td>
<td>cgD3D11GetLatestDomainProfile</td>
<td>665</td>
</tr>
<tr>
<td>8.11</td>
<td>cgD3D11GetLatestGeometryProfile</td>
<td>666</td>
</tr>
<tr>
<td>8.12</td>
<td>cgD3D11GetLatestHullProfile</td>
<td>666</td>
</tr>
<tr>
<td>8.13</td>
<td>cgD3D11GetLatestPixelProfile</td>
<td>667</td>
</tr>
<tr>
<td>8.14</td>
<td>cgD3D11GetLatestVertexProfile</td>
<td>668</td>
</tr>
<tr>
<td>8.15</td>
<td>cgD3D11GetManageTextureParameters</td>
<td>669</td>
</tr>
<tr>
<td>8.16</td>
<td>cgD3D11GetOptimalOptions</td>
<td>670</td>
</tr>
<tr>
<td>8.17</td>
<td>cgD3D11GetProgramErrors</td>
<td>671</td>
</tr>
<tr>
<td>8.18</td>
<td>cgD3D11IsProfileSupported</td>
<td>672</td>
</tr>
<tr>
<td>8.19</td>
<td>cgD3D11IsProgramLoaded</td>
<td>672</td>
</tr>
<tr>
<td>8.20</td>
<td>cgD3D11LoadProgram</td>
<td>673</td>
</tr>
<tr>
<td>8.21</td>
<td>cgD3D11RegisterStates</td>
<td>674</td>
</tr>
<tr>
<td>8.22</td>
<td>cgD3D11SetDevice</td>
<td>675</td>
</tr>
<tr>
<td>8.23</td>
<td>cgD3D11SetManageTextureParameters</td>
<td>676</td>
</tr>
<tr>
<td>8.24</td>
<td>cgD3D11SetSamplerStateParameter</td>
<td>677</td>
</tr>
<tr>
<td>8.25</td>
<td>cgD3D11SetTextureParameter</td>
<td>678</td>
</tr>
<tr>
<td>8.26</td>
<td>cgD3D11SetTextureSamplerStateParameter</td>
<td>679</td>
</tr>
<tr>
<td>8.27</td>
<td>cgD3D11TranslateCGerror</td>
<td>680</td>
</tr>
<tr>
<td>8.28</td>
<td>cgD3D11TranslateHRESULT</td>
<td>681</td>
</tr>
<tr>
<td>8.29</td>
<td>cgD3D11TypeToSize</td>
<td>682</td>
</tr>
<tr>
<td>8.30</td>
<td>cgD3D11UnbindProgram</td>
<td>683</td>
</tr>
<tr>
<td>8.31</td>
<td>cgD3D11UnloadProgram</td>
<td>684</td>
</tr>
<tr>
<td>9</td>
<td>Cg Standard Library</td>
<td>685</td>
</tr>
<tr>
<td>9.1</td>
<td>abs</td>
<td>685</td>
</tr>
<tr>
<td>9.2</td>
<td>acos</td>
<td>686</td>
</tr>
<tr>
<td>9.3</td>
<td>all</td>
<td>687</td>
</tr>
<tr>
<td>9.4</td>
<td>any</td>
<td>688</td>
</tr>
<tr>
<td>9.5</td>
<td>asin</td>
<td>689</td>
</tr>
<tr>
<td>9.6</td>
<td>atan2</td>
<td>690</td>
</tr>
<tr>
<td>9.7</td>
<td>atan</td>
<td>691</td>
</tr>
<tr>
<td>9.8</td>
<td>bitCount</td>
<td>692</td>
</tr>
<tr>
<td>9.9</td>
<td>bitfieldExtract</td>
<td>693</td>
</tr>
<tr>
<td>9.10</td>
<td>bitfieldInsert</td>
<td>694</td>
</tr>
<tr>
<td>9.11</td>
<td>bitfieldReverse</td>
<td>695</td>
</tr>
<tr>
<td>9.12</td>
<td>ceil</td>
<td>696</td>
</tr>
<tr>
<td>9.13</td>
<td>clamp</td>
<td>697</td>
</tr>
<tr>
<td>9.14</td>
<td>clip</td>
<td>698</td>
</tr>
<tr>
<td>9.15</td>
<td>cos</td>
<td>699</td>
</tr>
<tr>
<td>9.16</td>
<td>cos</td>
<td>700</td>
</tr>
<tr>
<td>9.17</td>
<td>cross</td>
<td>702</td>
</tr>
<tr>
<td>9.18</td>
<td>ddx</td>
<td>702</td>
</tr>
<tr>
<td>9.19</td>
<td>ddy</td>
<td>703</td>
</tr>
<tr>
<td>9.20</td>
<td>degrees</td>
<td>705</td>
</tr>
<tr>
<td>9.21</td>
<td>determinant</td>
<td>706</td>
</tr>
<tr>
<td>9.22</td>
<td>distance</td>
<td>707</td>
</tr>
<tr>
<td>9.23</td>
<td>dot</td>
<td>708</td>
</tr>
<tr>
<td>9.24</td>
<td>exp</td>
<td>709</td>
</tr>
<tr>
<td>9.25</td>
<td>exp</td>
<td>710</td>
</tr>
<tr>
<td>9.26</td>
<td>faceforward</td>
<td>711</td>
</tr>
<tr>
<td>9.27</td>
<td>findLSB</td>
<td>712</td>
</tr>
<tr>
<td>9.28</td>
<td>findMSB</td>
<td>713</td>
</tr>
<tr>
<td>9.29</td>
<td>floatToIntBits</td>
<td>714</td>
</tr>
<tr>
<td>9.30</td>
<td>floatToRawIntBits</td>
<td>715</td>
</tr>
<tr>
<td>9.31</td>
<td>floor</td>
<td>716</td>
</tr>
<tr>
<td>9.32</td>
<td>fmod</td>
<td>717</td>
</tr>
<tr>
<td>9.33</td>
<td>frac</td>
<td>718</td>
</tr>
<tr>
<td>9.34</td>
<td>frexp</td>
<td>719</td>
</tr>
<tr>
<td>9.35</td>
<td>fwidth</td>
<td>720</td>
</tr>
<tr>
<td>9.36</td>
<td>intBitsToFloat</td>
<td>722</td>
</tr>
<tr>
<td>9.37</td>
<td>inverse</td>
<td>723</td>
</tr>
<tr>
<td>9.38</td>
<td>isfinite</td>
<td>723</td>
</tr>
<tr>
<td>9.39</td>
<td>isinf</td>
<td>724</td>
</tr>
<tr>
<td>9.40</td>
<td>isnan</td>
<td>725</td>
</tr>
<tr>
<td>9.41</td>
<td>ldexp</td>
<td>726</td>
</tr>
<tr>
<td>9.42</td>
<td>length</td>
<td>727</td>
</tr>
<tr>
<td>9.43</td>
<td>lerp</td>
<td>728</td>
</tr>
<tr>
<td>9.44</td>
<td>lit</td>
<td>730</td>
</tr>
<tr>
<td>9.45</td>
<td>log10</td>
<td>731</td>
</tr>
<tr>
<td>9.46</td>
<td>log2</td>
<td>732</td>
</tr>
<tr>
<td>9.47</td>
<td>log</td>
<td>733</td>
</tr>
<tr>
<td>9.48</td>
<td>max</td>
<td>734</td>
</tr>
<tr>
<td>9.49</td>
<td>min</td>
<td>735</td>
</tr>
<tr>
<td>9.50</td>
<td>modf</td>
<td>736</td>
</tr>
<tr>
<td>9.51</td>
<td>mul</td>
<td>737</td>
</tr>
<tr>
<td>9.52</td>
<td>normalize</td>
<td>741</td>
</tr>
<tr>
<td>9.53</td>
<td>pack</td>
<td>742</td>
</tr>
<tr>
<td>9.54</td>
<td>pow</td>
<td>744</td>
</tr>
<tr>
<td>9.55</td>
<td>radians</td>
<td>745</td>
</tr>
<tr>
<td>9.56</td>
<td>reflect</td>
<td>746</td>
</tr>
<tr>
<td>9.57</td>
<td>refract</td>
<td>746</td>
</tr>
<tr>
<td>9.58</td>
<td>round</td>
<td>747</td>
</tr>
<tr>
<td>Section</td>
<td>Function</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td></td>
</tr>
<tr>
<td>9.59</td>
<td>sqrt</td>
<td></td>
</tr>
<tr>
<td>9.60</td>
<td>saturate</td>
<td></td>
</tr>
<tr>
<td>9.61</td>
<td>sign</td>
<td></td>
</tr>
<tr>
<td>9.62</td>
<td>sincos</td>
<td></td>
</tr>
<tr>
<td>9.63</td>
<td>sinh</td>
<td></td>
</tr>
<tr>
<td>9.64</td>
<td>sin</td>
<td></td>
</tr>
<tr>
<td>9.65</td>
<td>smoothstep</td>
<td></td>
</tr>
<tr>
<td>9.66</td>
<td>sqrt</td>
<td></td>
</tr>
<tr>
<td>9.67</td>
<td>step</td>
<td></td>
</tr>
<tr>
<td>9.68</td>
<td>tanh</td>
<td></td>
</tr>
<tr>
<td>9.69</td>
<td>tan</td>
<td></td>
</tr>
<tr>
<td>9.70</td>
<td>tex1D</td>
<td></td>
</tr>
<tr>
<td>9.71</td>
<td>tex1Dbias</td>
<td></td>
</tr>
<tr>
<td>9.72</td>
<td>tex1Dcmpbias</td>
<td></td>
</tr>
<tr>
<td>9.73</td>
<td>tex1Dcmplod</td>
<td></td>
</tr>
<tr>
<td>9.74</td>
<td>tex1Dfetch</td>
<td></td>
</tr>
<tr>
<td>9.75</td>
<td>tex1Dlod</td>
<td></td>
</tr>
<tr>
<td>9.76</td>
<td>tex1Dproj</td>
<td></td>
</tr>
<tr>
<td>9.77</td>
<td>tex1Dsize</td>
<td></td>
</tr>
<tr>
<td>9.78</td>
<td>tex1DARRAYbias</td>
<td></td>
</tr>
<tr>
<td>9.79</td>
<td>tex1DARRAYcmpeqbias</td>
<td></td>
</tr>
<tr>
<td>9.80</td>
<td>tex1DARRAYcmpeqlod</td>
<td></td>
</tr>
<tr>
<td>9.81</td>
<td>tex1DARRAYfetch</td>
<td></td>
</tr>
<tr>
<td>9.82</td>
<td>tex1DARRAYlod</td>
<td></td>
</tr>
<tr>
<td>9.83</td>
<td>tex1DARRAY</td>
<td></td>
</tr>
<tr>
<td>9.84</td>
<td>tex1DARRAYproj</td>
<td></td>
</tr>
<tr>
<td>9.85</td>
<td>tex1DARRAYsize</td>
<td></td>
</tr>
<tr>
<td>9.86</td>
<td>tex2Dbias</td>
<td></td>
</tr>
<tr>
<td>9.87</td>
<td>tex2D</td>
<td></td>
</tr>
<tr>
<td>9.88</td>
<td>tex2Dcmpbias</td>
<td></td>
</tr>
<tr>
<td>9.89</td>
<td>tex2Dcmpeqbias</td>
<td></td>
</tr>
<tr>
<td>9.90</td>
<td>tex2Dcmpeqlod</td>
<td></td>
</tr>
<tr>
<td>9.91</td>
<td>tex2Dfetch</td>
<td></td>
</tr>
<tr>
<td>9.92</td>
<td>tex2Dlod</td>
<td></td>
</tr>
<tr>
<td>9.93</td>
<td>tex2Dproj</td>
<td></td>
</tr>
<tr>
<td>9.94</td>
<td>tex2DARRAYbias</td>
<td></td>
</tr>
<tr>
<td>9.95</td>
<td>tex2DARRAYfetch</td>
<td></td>
</tr>
<tr>
<td>9.96</td>
<td>tex2DARRAYlod</td>
<td></td>
</tr>
<tr>
<td>9.97</td>
<td>tex2DARRAY</td>
<td></td>
</tr>
<tr>
<td>9.98</td>
<td>tex2DARRAYproj</td>
<td></td>
</tr>
<tr>
<td>9.99</td>
<td>tex2DARRAYsize</td>
<td></td>
</tr>
<tr>
<td>9.100</td>
<td>tex2DMSfetch</td>
<td></td>
</tr>
<tr>
<td>9.101</td>
<td>tex2DMSsize</td>
<td></td>
</tr>
<tr>
<td>9.102</td>
<td>tex2DMSARRAYfetch</td>
<td></td>
</tr>
<tr>
<td>9.103</td>
<td>tex2DMSARRAYsize</td>
<td></td>
</tr>
<tr>
<td>9.104</td>
<td>tex3Dbias</td>
<td></td>
</tr>
<tr>
<td>9.105</td>
<td>tex3Dfetch</td>
<td></td>
</tr>
<tr>
<td>9.106</td>
<td>tex3Dlod</td>
<td></td>
</tr>
<tr>
<td>9.107</td>
<td>tex3D</td>
<td></td>
</tr>
<tr>
<td>9.108</td>
<td>tex3Dproj</td>
<td></td>
</tr>
<tr>
<td>9.109</td>
<td>tex3Dsize</td>
<td></td>
</tr>
<tr>
<td>9.110</td>
<td>texBUF</td>
<td></td>
</tr>
<tr>
<td>9.111</td>
<td>texBUFSIZE</td>
<td></td>
</tr>
<tr>
<td>9.112</td>
<td>texCUBEARRAYbias</td>
<td></td>
</tr>
</tbody>
</table>
10 CgFX States

10.1 AddressUi ................................................. 813
10.2 AddressVi ................................................. 813
10.3 AddressWi ................................................ 814
10.4 AlphaArg0 ................................................. 815
10.5 AlphaArg1 ................................................. 815
10.6 AlphaArg2 ................................................. 816
10.7 AlphaBlendEnable ......................................... 816
10.8 AlphaFunc ................................................. 817
10.9 AlphaOp .................................................. 818
10.10 AlphaRef ................................................. 818
10.11 AlphaTestEnable ......................................... 819
10.12 AmbientMaterialSource ................................. 819
10.13 Ambient .................................................. 820
10.14 AutoNormalEnable ....................................... 821
10.15 BlendColor ............................................... 821
10.16 BlendEnable ............................................. 822
10.17 BlendEquation ........................................... 822
10.18 BlendEquationSeparate ................................ 823
10.19 BlendFunc ............................................... 824
10.20 BlendFuncSeparate ...................................... 824
10.21 BlendOpAlpha ............................................. 825
10.22 BlendOp .................................................. 826
10.23 BorderColori ............................................. 826
10.24 BumpEnvLOffset ......................................... 827
10.25 BumpEnvLScale .......................................... 827
10.26 BumpEnvMat00 ........................................... 828
10.27 BumpEnvMat01 ........................................... 828
10.28 BumpEnvMat10 ........................................... 829
10.29 BumpEnvMat11 ........................................... 830
10.30 ClearColor ............................................... 830
10.31 ClearDepth ............................................... 831
10.32 ClearStencil ............................................. 831
10.33 Clipping ................................................ 832
10.250vertexShaderConstant4 ........................................... 961
10.251VertexShaderConstantB .......................................... 961
10.252VertexShaderConstantF .......................................... 962
10.253VertexShaderConstantI .......................................... 962
10.254VertexShaderConstant ........................................... 963
10.255VertexShader ...................................................... 964
10.256ViewTransform .................................................... 964
10.257WorldTransform ................................................... 965
10.258Wrap0 .............................................................. 965
10.259Wrap10 .............................................................. 966
10.260Wrap11 .............................................................. 967
10.261Wrap12 .............................................................. 967
10.262Wrap13 .............................................................. 968
10.263Wrap14 .............................................................. 969
10.264Wrap15 .............................................................. 969
10.265Wrap1 .............................................................. 970
10.266Wrap2 .............................................................. 970
10.267Wrap3 .............................................................. 971
10.268Wrap4 .............................................................. 972
10.269Wrap5 .............................................................. 972
10.270Wrap6 .............................................................. 973
10.271Wrap7 .............................................................. 974
10.272Wrap8 .............................................................. 974
10.273Wrap9 .............................................................. 975
10.274ZEnable ............................................................ 976
10.275ZFunc .............................................................. 976
10.276ZWriteEnable ........................................................ 977

11 CgFX Sampler States ......................................................... 979
11.1 AddressU .................................................................. 979
11.2 AddressV .................................................................. 980
11.3 AddressW .................................................................. 980
11.4 BorderColor .............................................................. 981
11.5 CompareFunc ............................................................ 982
11.6 CompareMode ............................................................ 982
11.7 DepthMode .................................................................. 983
11.8 GenerateMipmap .......................................................... 983
11.9 LODBias .................................................................... 984
11.10 MagFilter ............................................................... 984
11.11 MaxAnisotropy .......................................................... 985
11.12 MinMipLevel ............................................................. 986
11.13 MinFilter .................................................................. 986
11.14 MinMipLevel ............................................................. 987
11.15 MipFilter .................................................................. 988
11.16 MipMapLodBias ........................................................... 988
11.17 SRGBTexture ............................................................ 989
11.18 Texture .................................................................... 989
11.19 WrapR ................................................................. 990
11.20 WrapS ..................................................................... 990
11.21 WrapT .................................................................... 990

12 Cg Topics ...................................................................... 991
12.1 Cg 1.2 Runtime API Additions ........................................ 991
12.2 Cg ................................................................. 996
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>12.3</td>
<td>glut</td>
<td>997</td>
</tr>
<tr>
<td>12.4</td>
<td>Mac OS X</td>
<td>997</td>
</tr>
<tr>
<td>12.5</td>
<td>Trace</td>
<td>998</td>
</tr>
<tr>
<td>12.6</td>
<td>win64</td>
<td>1000</td>
</tr>
</tbody>
</table>

13 CG Toolkit Releases

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>13.1</td>
<td>cg_2_1_0009</td>
<td>1003</td>
</tr>
<tr>
<td>13.2</td>
<td>cg_2_1_0012</td>
<td>1003</td>
</tr>
<tr>
<td>13.3</td>
<td>cg_2_1_0016</td>
<td>1004</td>
</tr>
<tr>
<td>13.4</td>
<td>cg_2_1_0017</td>
<td>1005</td>
</tr>
<tr>
<td>13.5</td>
<td>cg_2_2_0004</td>
<td>1005</td>
</tr>
<tr>
<td>13.6</td>
<td>cg_2_2_0006</td>
<td>1006</td>
</tr>
<tr>
<td>13.7</td>
<td>cg_2_2_0010</td>
<td>1006</td>
</tr>
<tr>
<td>13.8</td>
<td>cg_2_2_0017</td>
<td>1007</td>
</tr>
<tr>
<td>13.9</td>
<td>cg_3_0_0007</td>
<td>1008</td>
</tr>
<tr>
<td>13.10</td>
<td>cg_3_0_0015</td>
<td>1009</td>
</tr>
<tr>
<td>13.11</td>
<td>cg_3_0_0016</td>
<td>1009</td>
</tr>
<tr>
<td>13.12</td>
<td>cg_3_1_0010</td>
<td>1010</td>
</tr>
<tr>
<td>13.13</td>
<td>cg_3_1_0013</td>
<td>1012</td>
</tr>
</tbody>
</table>

Index

<table>
<thead>
<tr>
<th>Subsection</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1017</td>
</tr>
</tbody>
</table>
Contents:
1.1 Cg Language Specification

Copyright (c) 2001-2012 NVIDIA Corp.

This is version 3.1 of the Cg Language specification. This language specification describes version 3.1 of the Cg language.

Language Overview

The Cg language is primarily modeled on ANSI C, but adopts some ideas from modern languages such as C++ and Java, and from earlier shading languages such as RenderMan and the Stanford shading language. The language also introduces a few new ideas. In particular, it includes features designed to represent data flow in stream-processing architectures such as GPUs. Profiles, which are specified at compile time, may subset certain features of the language, including the ability to implement loops and the precision at which certain computations are performed.

Like C, Cg is designed primarily as a low-level programming language. Features are provided that map as directly as possible to hardware capabilities. Higher level abstractions are designed primarily to not get in the way of writing code that maps directly to the hardware in the most efficient way possible. The changes in the language from C primarily reflect differences in the way GPU hardware works compared to conventional CPUs. GPUs are designed to run large numbers of small threads of processing in parallel, each running a copy of the same program on a different data set.

Differences from ANSI C

Cg was developed based on the ANSI-C language with the following major additions, deletions, and changes. (This is a summary—more detail is provided later in this document):

Silent Incompatibilities

Most of the changes from ANSI C are either omissions or additions, but there are a few potentially silent incompatibilities. These are changes within Cg that could cause a program that compiles without errors to behave in a manner different from C:

- The type promotion rules for constants are different when the constant is not explicitly typed using a type cast or type suffix. In general, a binary operation between a constant that is not explicitly typed and a variable is performed at the variable’s precision, rather than at the constant’s default precision.
- Declarations of struct perform an automatic typedef (as in C++) and thus could override a previously declared type.
- Arrays are first-class types that are distinct from pointers. As a result, array assignments semantically perform a copy operation for the entire array.

Similar Operations That Must be Expressed Differently

There are several changes that force the same operation to be expressed differently in Cg than in C:
• A Boolean type, `bool`, is introduced, with corresponding implications for operators and control constructs.
• Arrays are first-class types because Cg does not support pointers.
• Functions pass values by value/result, and thus use an `out` or `inout` modifier in the formal parameter list to return a parameter. By default, formal parameters are `in`, but it is acceptable to specify this explicitly. Parameters can also be specified as `in out`, which is semantically the same as `inout`.

C features not present in Cg

• Language profiles (described in the Profiles section) may subset language capabilities in a variety of ways. In particular, language profiles may restrict the use of for and while loops. For example, some profiles may only support loops that can be fully unrolled at compile time.
• Reserved keywords `goto`, `switch`, `case`, and `default` are not supported, nor are labels.
• Pointers and pointer-related capabilities, such as the `&` and `->` operators, are not supported.
• Arrays are supported, but with some limitations on size and dimensionality. Restrictions on the use of computed subscripts are also permitted. Arrays may be designated as packed. The operations allowed on packed arrays may be different from those allowed on unpacked arrays. Predefined packed types are provided for vectors and matrices. It is strongly recommended that these predefined types be used.
• There is no `enum` or `union`.
• There are no bit-field declarations in structures.
• All integral types are implicitly signed, there is no `signed` keyword.

Cg features not present in C

• A binding semantic may be associated with a structure tag, a variable, or a structure element to denote that object’s mapping to a specific hardware or API resource. Binding semantics are described in the Binding Semantics section.
• There is a built-in swizzle operator: `.xyzw` or `.rgba` for vectors. This operator allows the components of a vector to be rearranged and also replicated. It also allows the creation of a vector from a scalar.
• For an lvalue, the swizzle operator allows components of a vector or matrix to be selectively written.
• There is a similar built-in swizzle operator for matrices: `_m<row><col>[_m<row><col>][...]. This operator allows access to individual matrix components and allows the creation of a vector from elements of a matrix. For compatibility with DirectX 8 notation, there is a second form of matrix swizzle, which is described later.
• Numeric data types are different. Cg’s primary numeric data types are `float`, `half`, and `fixed`. Fragment profiles are required to support all three data types, but may choose to implement `half` and/or `fixed` at `float` precision. Vertex profiles are required to support `half` and `float`, but may choose to implement `half` at `float` precision. Vertex profiles may omit support for `fixed` operations, but must still support definition of `fixed` variables. Cg allows profiles to omit run-time support for `int` and other integer types. Cg allows profiles to treat `double` as `float`.
• Many operators support per-element vector operations.
• The `?:`, `||`, `&&`, `!`, and comparison operators can be used with `bool` vectors to perform multiple conditional operations simultaneously. The side effects of all operands to vector `?:`, `||`, and `&&` operators are always executed.
• Non-static global variables, and parameters to top-level functions (such as `main()`) may be designated as uniform. A uniform variable may be read and written within a program, just like any other variable. However, the uniform modifier indicates that the initial value of the variable/parameter is expected to be constant across a large number of invocations of the program.
• A new set of `sampler*` types represents handles to texture sampler units.
• Functions may have default values for their parameters, as in C++. These defaults are expressed using assignment syntax.
• Function and operator overloading is supported.
• Variables may be defined anywhere before they are used, rather than just at the beginning of a scope as in C. (That is, we adopt the C++ rules that govern where variable declarations are allowed.) Variables may not be redeclared within the same scope.
• Vector constructors, such as the form `float4(1,2,3,4)`, and matrix constructors may be used anywhere in...
an expression.
• A struct definition automatically performs a corresponding typedef, as in C++.
• C++-style // comments are allowed in addition to C-style /* ... */ comments.
• A limited form of inheritance is supported; interface types may be defined which contain only member functions (no data members) and struct types may inherit from a single interface and provide specific implementations for all the member functions. Interface objects may not be created; a variable of interface type may have any implementing struct type assigned to it.

Detailed Language Specification

Definitions

The following definitions are based on the ANSI C standard:

Object

An object is a region of data storage in the execution environment, the contents of which can represent values. When referenced, an object may be interpreted as having a particular type.

Declaration

A declaration specifies the interpretation and attributes of a set of identifiers.

Definition

A declaration that also causes storage to be reserved for an object or code that will be generated for a function named by an identifier is a definition.

Profiles

Compilation of a Cg program, a top-level function, always occurs in the context of a compilation profile. The profile specifies whether certain optional language features are supported. These optional language features include certain control constructs and standard library functions. The compilation profile also defines the precision of the float, half, and fixed data types, and specifies whether the fixed and sampler* data types are fully or only partially supported. The profile also specifies the environment in which the program will be run. The choice of a compilation profile is made externally to the language, by using a compiler command-line switch, for example.

The profile restrictions are only applied to the top-level function that is being compiled and to any variables or functions that it references, either directly or indirectly. If a function is present in the source code, but not called directly or indirectly by the top-level function, it is free to use capabilities that are not supported by the current profile.

The intent of these rules is to allow a single Cg source file to contain many different top-level functions that are targeted at different profiles. The core Cg language specification is sufficiently complete to allow all of these functions to be parsed. The restrictions provided by a compilation profile are only needed for code generation, and are therefore only applied to those functions for which code is being generated. This specification uses the word “program” to refer to the top-level function, any functions the top-level function calls, and any global variables or typedef definitions it references.

Each profile must have a separate specification that describes its characteristics and limitations.

This core Cg specification requires certain minimum capabilities for all profiles. In some cases, the core specification distinguishes between vertex-program and fragment-program profiles, with different minimum capabilities for each.

Declarations and declaration specifiers.

A Cg program consists of a series of declarations, each of which declares one or more variables or functions, or declares and defines a single function. Each declaration consists of zero or more declaration specifiers, a type, and one or more declarators. Some of the declaration specifiers are the same as those in ANSI C; others are new to Cg

const
Marks a variable as a constant that cannot be assigned to within the program. Unless this is combined with `uniform` or `varying`, the declarator must include an initializer to give the variable a value.

**extern**

Marks this declaration as solely a declaration and not a definition. There must be a non-`extern` declaration elsewhere in the program.

**in**

Only usable on parameter and `varying` declarations. Marks the parameter or varying as an input to the function or program. Function parameters with no `in`, `out`, or `inout` specifier are implicitly `in`.

**inline**

Only usable on a function definition. Tells the compiler that it should always inline calls to the function if at all possible.

**inout**

Only usable on parameter and `varying` declarations. Marks the parameter or varying as both an input to and an output from the function or program.

**static**

Only usable on global variables. Marks the variable as ‘private’ to the program, and not visible externally. Cannot be combined with `uniform` or `varying`.

**out**

Only usable on parameter and `varying` declarations. Marks the parameter or varying as an output from the function or program.

**uniform**

Usable on global variables and parameters to the top-level main function of a program and to define constant buffers (see the `Constant Buffers` section). If specified on a non-top-level function parameter it is ignored. The intent of this rule is to allow a function to serve as either a top-level function or as one that is not.

Note that `uniform` variables may be read and written just like non-`uniform` variables. The `uniform` qualifier simply provides information about how the initial value of the variable is to be specified and stored, through a mechanism external to the language.

**varying**

Only usable on global variables and parameters to the top-level main function of a program. If specified on a non-top-level function parameter it is ignored.

**profile name**

The name of any profile (or profile wildcard – see `Profiles`) may be used as a specifier on any function declaration. It defines a function that is only visible in the corresponding profiles.

The specifiers `uniform` and `varying` specify how data is transferred between the rest of the world and a Cg program. Typically, the initial value of a `uniform` variable or parameter is stored in a different class of hardware register for a `varying`. Furthermore, the external mechanism for specifying the initial value of `uniform` variables or parameters may be different than that used for specifying the initial value of `varying` variables or parameters. Parameters qualified as `uniform` are normally treated as persistent state, while `varying` parameters are treated as streaming data, with a new value specified for each stream record (such as within a vertex array).

Non-static global variables are treated as `uniform` by default, while parameters to the top-level function are treated as `varying` by default.
Each declaration is visible (“in scope”) from the point of its declarator until the end of the enclosing block or the end of the compilation unit if outside any block. Declarations in named scopes (such as structs and interfaces) may be visible outside of their scope using explicit scope qualifiers, as in C++.

**Semantics**

Each declarator in a declaration may optionally have a semantic specified with it. A semantic specifies how the variable is connected to the environment in which the program runs. All semantics are profile specific (so they have different meanings in different profiles), though there is some attempt to be consistent across profiles. Each profile specification must specify the set of semantics which the profile understands, as well as what behavior occurs for any other unspecified semantics.

**Function Declarations**

Functions are declared essentially as in C. A function that does not return a value must be declared with a `void` return type. A function that takes no parameters may be declared in one of two ways:

As in C, using the `void` keyword:

```c
functionName(void)
```

With no parameters at all:

```c
functionName()
```

Functions may be declared as `static`. If so, they may not be compiled as a program and are not visible externally.

**Function overloading and optional arguments**

Cg supports function overloading; that is you may define multiple functions with the same name. The function actually called at any given call site is based on the types of the arguments at that call site; the definition that best matches is called. See the `function overloading` section for the precise rules. Trailing arguments with initializers are optional arguments; defining a function with optional arguments is equivalent to defining multiple overloaded functions that differ by having and not having the optional argument. The value of the initializer is used only for the version that does not have the argument and is ignored if the argument is present.

**Overloading of Functions by Profile**

Cg supports overloading of functions by compilation profile. This capability allows a function to be implemented differently for different profiles. It is also useful because different profiles may support different subsets of the language capabilities, and because the most efficient implementation of a function may be different for different profiles.

The profile name must precede the return type name in the function declaration. For example, to define two different versions of the function `myfunc` for the `profileA` and `profileB` profiles:

```c
profileA float myfunc(float x) {...};
profileB float myfunc(float x) {...};
```

If a type is defined (using a `typedef`) that has the same name as a profile, the identifier is treated as a type name, and is not available for profile overloading at any subsequent point in the file.

If a function definition does not include a profile, the function is referred to as an “open-profile” function. Open-profile functions apply to all profiles.

Several wildcard profile names are defined. The name `vs` matches any vertex profile, while the name `ps` matches any fragment or pixel profile. The names `ps_1` and `ps_2` match any DX8 pixel shader 1.x profile, or DX9 pixel shader 2.x profile, respectively. Similarly, the names `vs_1` and `vs_2` match any DX vertex shader 1.x or 2.x, respectively. Additional valid wildcard profile names may be defined by individual profiles.

In general, the most specific version of a function is used. More details are provided in the section on function overloading, but roughly speaking, the search order is the following:
1. version of the function with the exact profile overload
2. version of the function with the most specific wildcard profile overload (e.g. vs, “ps_1”)
3. version of function with no profile overload

This search process allows generic versions of a function to be defined that can be overridden as needed for particular hardware.

Syntax for Parameters in Function Definitions

Functions are declared in a manner similar to C, but the parameters in function definitions may include a binding semantic (discussed later) and a default value.

Each parameter in a function definition takes the following form:

```
<declspecs> <type> identifier [: <binding_semantic>] [= <default>]
```

`<default>` is an expression that resolves to a constant at compile time.

Default values are only permitted for uniform parameters, and for in parameters to non top-level functions.

Function Calls

A function call returns an rvalue. Therefore, if a function returns an array, the array may be read but not written. For example, the following is allowed:

```
y = myfunc(x)[2];
```

But, this is not:

```
myfunc(x)[2] = y;
```

For multiple function calls within an expression, the calls can occur in any order—it is undefined.

Types

Cg’s types are as follows:

- The `int` type is preferably 32-bit two’s complement. Profiles may optionally treat `int` as `float`.
- The `unsigned` type is preferably a 32-bit ordinal value. `unsigned` may also be used with other integer types to make different sized unsigned values
- The `char`, `short`, and `long` types are two’s complement integers of various sizes. The only requirement is that `char` is no larger than `short`, `short` is no larger than `int` and `long` is at least as large as `int`
- The `float` type is as close as possible to the IEEE single precision (32-bit) floating point format. Profiles must support the `float` data type.
- The `half` type is lower-precision IEEE-like floating point. Profiles must support the `half` type, but may choose to implement it with the same precision as the `float` type.
- The `fixed` type is a signed type with a range of at least [-2,2) and with at least 10 bits of fractional precision. Overflow operations on the data type clamp rather than wrap. Fragment profiles must support the `fixed` type, but may implement it with the same precision as the `half` or `float` types. Vertex profiles are required to provide partial support (as defined below) for the `fixed` type. Vertex profiles have the option to provide full support for the `fixed` type or to implement the `fixed` type with the same precision as the `half` or `float` types.
- The `bool` type represents Boolean values. Objects of `bool` type are either true or false.
- The `cint` type is 32-bit two’s complement. This type is meaningful only at compile time; it is not possible to declare objects of type `cint`.
- The `cfloat` type is IEEE single-precision (32-bit) floating point. This type is meaningful only at compile time; it is not possible to declare objects of type `cfloat`.
- The `void` type may not be used in any expression. It may only be used as the return type of functions that do not return a value.
• The sampler* types are handles to texture objects. Formal parameters of a program or function may be of type sampler*. No other definition of sampler* variables is permitted. A sampler* variable may only be used by passing it to another function as an in parameter. Assignment to sampler* variables is not permitted, and sampler* expressions are not permitted.

The following sampler types are always defined: sampler, sampler1D, sampler2D, sampler3D, samplerCUBE, samplerRECT.

The base sampler type may be used in any context in which a more specific sampler type is valid. However, a sampler variable must be used in a consistent way throughout the program. For example, it cannot be used in place of both a sampler1D and a sampler2D in the same program. The sampler type is deprecated and only provided for backwards compatibility with Cg 1.0.

Fragment profiles are required to fully support the sampler, sampler1D, sampler2D, sampler3D, and samplerCUBE data types. Fragment profiles are required to provide partial support (as defined below) for the samplerRECT data type and may optionally provide full support for this data type.

Vertex profiles are required to provide partial support for the six sampler data types and may optionally provide full support for these data types.

• An array type is a collection of one or more elements of the same type. An array variable has a single index.

• Some array types may be optionally designated as packed, using the packed type modifier. The storage format of a packed type may be different from the storage format of the corresponding unpacked type. The storage format of packed types is implementation dependent, but must be consistent for any particular combination of compiler and profile. The operations supported on a packed type in a particular profile may be different than the operations supported on the corresponding unpacked type in that same profile. Profiles may define a maximum allowable size for packed arrays, but must support at least size 4 for packed vector (1D array) types, and 4x4 for packed matrix (2D array) types.

• When declaring an array of arrays in a single declaration, the packed modifier refers to all of the arrays. However, it is possible to declare an unpacked array of packed arrays by declaring the first level of array in a typedef using the packed keyword and then declaring an array of this type in a second statement. It is not possible to declare a packed array of unpacked arrays.

• For any supported numeric data type TYPE, implementations must support the following packed array types, which are called vector types. Type identifiers must be predefined for these types in the global scope:

```c
typedef packed TYPE TYPE1[1];
typedef packed TYPE TYPE2[2];
typedef packed TYPE TYPE3[3];
typedef packed TYPE TYPE4[4];
```

For example, implementations must redefine the type identifiers float1, float2, float3, float4, and so on for any other supported numeric type.

• For any supported numeric data type TYPE, implementations must support the following packed array types, which are called matrix types. Implementations must also redefine type identifiers (in the global scope) to represent these types:

```c
packed TYPE1 TYPE1x1[1][1];
packed TYPE2 TYPE1x2[1][1];
packed TYPE3 TYPE1x3[1][1];
packed TYPE4 TYPE1x4[1][1];
packed TYPE1 TYPE2x1[2][1];
packed TYPE2 TYPE2x2[2][2];
packed TYPE3 TYPE2x3[2][2];
packed TYPE4 TYPE2x4[2][2];
packed TYPE1 TYPE3x1[3][1];
packed TYPE2 TYPE3x2[3][2];
packed TYPE3 TYPE3x3[3][3];
packed TYPE4 TYPE3x4[3][3];
packed TYPE1 TYPE4x1[4][1];
packed TYPE2 TYPE4x2[4][2];
packed TYPE3 TYPE4x3[4][3];
packed TYPE4 TYPE4x4[4][4];
```
For example, implementations must predefine the type identifiers float2x1, float3x3, float4x4, and so on. A typedef follows the usual matrix-naming convention of TYPErows_X_columns. If we declare float4x4 a, then

a[3] is equivalent to a._m30_m31_m32_m33

Both expressions extract the third row of the matrix.

- Implementations are required to support indexing of vectors and matrices with constant indices.
- A struct type is a collection of one or more members of possibly different types. It may include both function members (methods) and data members (fields).

Struct and Interface types

Interface types are defined with an interface keyword in place of the normal struct keyword. Interface types may only declare member functions, not data members. Interface member functions may only be declared, not defined (no default implementations in C++ parlance).

Struct types may inherit from a single interface type, and must define an implementation member function for every member function declared in the interface type.

Partial Support of Types

This specification mandates “partial support” for some types. Partial support for a type requires the following:

- Definitions and declarations using the type are supported.
- Assignment and copy of objects of that type are supported (including implicit copies when passing function parameters).
- Top-level function parameters may be defined using that type.

If a type is partially supported, variables may be defined using that type but no useful operations can be performed on them. Partial support for types makes it easier to share data structures in code that is targeted at different profiles.

Type Categories

- The signed integral type category includes types cint, char, short, int, and long.
- The unsigned integral type category includes types unsigned char, unsigned short, unsigned int, and unsigned long. unsigned is the same as unsigned int
- The integral category includes both signed integral and unsigned integral types
- The floating type category includes types cfloat, float, half, and fixed (Note that floating really means floating or fixed/fractional.)
- The numeric type category includes integral and floating types.
- The compile-time type category includes types cfloat and cint. These types are used by the compiler for constant type conversions.
- The dynamic type category includes all interface and unsized array types
- The concrete type category includes all types that are not included in the compile-time and dynamic type category.
- The scalar type category includes all types in the numeric category, the bool type, and all types in the compile-time category. In this specification, a reference to a category type (such as a reference to a numeric type) means one of the types included in the category (such as float, half, or fixed).

Constants

Constant literals are defined as in C, including an optional 0 or 0x prefix for octal or hexadecimal constants, and e exponent suffix for floating point constants. A constant may be explicitly typed or implicitly typed. Explicit typing of a constant is performed, as in C, by suffixing the constant with a one or two characters indicating the type of the constant:

- d for double
- f for float
- h for half
Cg 3.1 Reference Manual, Release 3.1

• i for int
• l for long
• s for short
• t for char
• u for unsigned, which may also be followed by s, t, i, or l
• x for fixed

Any constant that is not explicitly typed is implicitly typed. If the constant includes a decimal point or an ‘e’ exponent suffix, it is implicitly typed as cfloat. If it does not include a decimal point, it is implicitly typed as cint.

By default, constants are base 10. For compatibility with C, integer hexadecimal constants may be specified by prefixing the constant with 0x, and integer octal constants may be specified by prefixing the constant with 0.

Compile-time constant folding is preferably performed at the same precision that would be used if the operation were performed at run time. Some compilation profiles may allow some precision flexibility for the hardware; in such cases the compiler should ideally perform the constant folding at the highest hardware precision allowed for that data type in that profile.

If constant folding cannot be performed at run-time precision, it may optionally be performed using the precision indicated below for each of the numeric datatypes:

float
s23e8 (“fp32”) IEEE single precision floating point
half
s10e5 (“fp16”) floating point w/ IEEE semantics
fixed
S1.10 fixed point, clamping to [-2, 2)
double
s52e11 (“fp64”) IEEE double precision floating point
int
signed 32 bit twos-complement integer
char
signed 8 bit twos-complement integer
short
signed 16 bit twos-complement integer
long
signed 64 bit twos-complement integer

Type Conversions

Some type conversions are allowed implicitly, while others require an cast. Some implicit conversions may cause a warning, which can be suppressed by using an explicit cast. Explicit casts are indicated using C-style syntax (e.g., casting variable to the float4 type may be achieved via “(float4)variablename”).

Scalar conversions

Implicit conversion of any scalar numeric type to any other scalar numeric type is allowed. A warning may be issued if the conversion is implicit and it is possible that precision is lost. implicit conversion of any scalar object type to any compatible scalar object type is also allowed. Conversions between incompatible scalar object types or object and numeric types are not allowed, even with an explicit cast. “sampler” is compatible with “sampler1D”, “sampler2D”, “sampler3D”, “samplerCube”, and “samplerRECT”. No
other object types are compatible (“sampler1D” is not compatible with “sampler2D”, even though both are compatible with “sampler”).

Scalar types may be implicitly converted to vectors and matrixes of compatible type. The scalar will be replicated to all elements of the vector or matrix. Scalar types may also be explicitly cast to structure types if the scalar type can be legally cast to every member of the structure.

Vector conversions

Vectors may be converted to scalar types (selects the first element of the vector). A warning is issued if this is done implicitly. A vector may also be implicitly converted to another vector of the same size and compatible element type.

A vector may be converted to a smaller compatible vector, or a matrix of the same total size, but a warning if issued if an explicit cast is not used.

Matrix conversions

Matrixes may be converted to a scalar type (selects to 0,0 element). As with vectors, this causes a warning if its done implicitly. A matrix may also be converted implicitly to a matrix of the same size and shape and compatible element type.

A Matrix may be converted to a smaller matrix type (selects the upper-left submatrix), or to a vector of the same total size, but a warning is issued if an explicit cast is not used.

Structure conversions

A structure may be explicitly cast to the type of its first member, or to another structure type with the same number of members, if each member of the struct can be converted to the corresponding member of the new struct. No implicit conversions of struct types are allowed.

Array conversions

An array may be explicitly converted to another array type with the same number of elements and a compatible element type. A compatible element type is any type to which the element type of the initial array may be implicitly converted to. No implicit conversions of array types are allowed.

<table>
<thead>
<tr>
<th>Source type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
</tr>
<tr>
<td>T -------+--------+--------+--------+--------+--------</td>
</tr>
<tr>
<td>a Scalar</td>
</tr>
<tr>
<td>r -------+--------+--------+--------+--------+--------</td>
</tr>
<tr>
<td>g Vector</td>
</tr>
<tr>
<td>e -------+--------+--------+--------+--------+--------</td>
</tr>
<tr>
<td>t Matrix</td>
</tr>
<tr>
<td>t -------+--------+--------+--------+--------+--------</td>
</tr>
<tr>
<td>y -------+--------+--------+--------+--------+--------</td>
</tr>
<tr>
<td>p Array</td>
</tr>
<tr>
<td>e -------+--------+--------+--------+--------+--------</td>
</tr>
</tbody>
</table>

A = allowed implicitly or explicitly
W = allowed, but warning issued if implicit
E = only allowed with explicit cast
- = not allowed

Notes:
(1) not allowed if target is larger than source. Warning if target is smaller than source
(2) only allowed if source and target are the same total size
(3) only if the first member of the source can be converted to the target
(4) only if the target struct contains a single field of the
source type
(5) only if both source and target have the same number of
members and each member of the source can be converted
to the corresponding member of the target.
(6) Source and target sizes must be the same and element types
must be compatible
(7) Array type must be an array of vectors that matches the
matrix type.

Explicit casts are:
• compile-time type when applied to expressions of compile-time type.
• numeric type when applied to expressions of numeric or compile-time types.
• numeric vector type when applied to another vector type of the same number of elements.
• numeric matrix type when applied to another matrix type of the same
number of rows and columns.

Type Equivalency
Type T1 is equivalent to type T2 if any of the following are true:
• T2 is equivalent to T1.
• T1 and T2 are the same scalar, vector, or structure type. A packed array type is not equivalent to the same size
unpacked array.
• T1 is a typedef name of T2.
• T1 and T2 are arrays of equivalent types with the same number of elements.
• The unqualified types of T1 and T2 are equivalent, and both types have the same qualifications.
• T1 and T2 are functions with equivalent return types, the same number of parameters, and all corresponding
parameters are pair-wise equivalent.

Type-Promotion Rules
The cf float and cint types behave like float and int types, except for the usual arithmetic conversion behavior
(defined below) and function-overloading rules (defined later).

The usual arithmetic conversions for binary operators are defined as follows:
1. If one operand is cint it is converted to the other type
2. If one operand is cf float and the other is floating, the cf float is converted to the other type
3. If both operands are floating then the smaller type is converted to the larger type
4. If one operand is floating and the other is integral, the integral argument is converted to the floating type.
5. If both operands are integral the smaller type is converted to the larger type
6. If one operand is signed integral while the other is unsigned integral and they are the same size, the signed type
is converted to unsigned.

Note that conversions happen prior to performing the operation.

Assignment
Assignment of an expression to a concrete typed object converts the expression to the type of the object. The resulting
value is then assigned to the object or value.

The value of the assignment expressions (=, +=, and so on) is defined as in C:
An assignment expression has the value of the left operand after the assignment but is not an lvalue. The type of an
assignment expression is the type of the left operand unless the left operand has a qualified type, in which case it is
the unqualified version of the type of the left operand. The side effect of updating the stored value of the left operand
occurs between the previous and the next sequence point.
An assignment of an expression to a dynamic typed object is only possible if the type of the expression is compatible
with the dynamic object type. The object will then take on the type of the expression assigned to it until the next
assignment to it.
“Smearing” of Scalars to Vectors

If a binary operator is applied to a vector and a scalar, the scalar is automatically type-promoted to a same-sized vector by replicating the scalar into each component. The ternary ?: operator also supports smearing. The binary rule is applied to the second and third operands first, and then the binary rule is applied to this result and the first operand.

Namespaces

Just as in C, there are two namespaces. Each has multiple scopes, as in C.

- Tag namespace, which consists of struct tags
- Regular namespace:
  - typedef names (including an automatic typedef from a struct declaration)
  - variables
  - function names

Constant Buffers

Constant buffers are used to define uniform blocks in Cg. A constant buffer is defined using the uniform keyword:

```
uniform <blockname> {...} <object> : BUFFER [ [N] ];
```

The N refers to a unit number and is optional. Each object defined within the block will have a unique slot in the constant buffer.

Arrays and Subscripting

Arrays are declared as in C, except that they may optionally be declared to be packed, as described earlier. Arrays in Cg are first-class types, so array parameters to functions and programs must be declared using array syntax, rather than pointer syntax. Likewise, assignment of an array-typed object implies an array copy rather than a pointer copy.

Arrays with size [1] may be declared but are considered a different type from the corresponding non-array type.

Because the language does not currently support pointers, the storage order of arrays is only visible when an application passes parameters to a vertex or fragment program. Therefore, the compiler is currently free to allocate temporary variables as it sees fit.

The declaration and use of arrays of arrays is in the same style as in C. That is, if the 2D array A is declared as

```
float A[4][4];
```

then, the following statements are true:

- The array is indexed as A[row][column];
- The array can be built with a constructor using

```
float4x4 A = { { A[0][0], A[0][1], A[0][2], A[0][3] },
  { A[1][0], A[1][1], A[1][2], A[1][3] },
```

- A[0] is equivalent to float4(A[0][0], A[0][1], A[0][2], A[0][3])

Support must be provided for structs containing arrays.

Unsized Arrays

Objects may be declared as unsized arrays by using a declaration with an empty size [] and no initializer. If a declarator uses unsized array syntax with an initializer, it is declared with a concrete (sized) array type based on the declarator. Unsized arrays are dynamic typed objects that take on the size of any array assigned to them.
Minimum Array Requirements

Profiles are required to provide partial support for certain kinds of arrays. This partial support is designed to support vectors and matrices in all profiles. For vertex profiles, it is additionally designed to support arrays of light state (indexed by light number) passed as uniform parameters, and arrays of skinning matrices passed as uniform parameters.

Profiles must support subscripting, copying, size querying and swizzling of vectors and matrices. However, subscripting with run-time computed indices is not required to be supported.

Vertex profiles must support the following operations for any non-packed array that is a uniform parameter to the program, or is an element of a structure that is a uniform parameter to the program. This requirement also applies when the array is indirectly a uniform program parameter (that is, it and or the structure containing it has been passed via a chain of in function parameters). The three operations that must be supported are

- rvalue subscripting by a run-time computed value or a compile-time value.
- passing the entire array as a parameter to a function, where the corresponding formal function parameter is declared as in.
- querying the size of the array with a .length suffix.

The following operations are explicitly not required to be supported:

- lvalue-subscripting
- copying
- other operators, including multiply, add, compare, and so on

Note that when a uniform array is rvalue subscripted, the result is an expression, and this expression is no longer considered to be a uniform program parameter. Therefore, if this expression is an array, its subsequent use must conform to the standard rules for array usage.

These rules are not limited to arrays of numeric types, and thus imply support for arrays of struct, arrays of matrices, and arrays of vectors when the array is a uniform program parameter. Maximum array sizes may be limited by the number of available registers or other resource limits, and compilers are permitted to issue error messages in these cases. However, profiles must support sizes of at least float arr[8], float4 arr[8], and float4x4 arr[4][4].

Fragment profiles are not required to support any operations on arbitrarily sized arrays; only support for vectors and matrices is required.

Function Overloading

Multiple functions may be defined with the same name, as long as the definitions can be distinguished by unqualified parameter types and do not have an open-profile conflict (as described in the section on open functions).

Function-matching rules:

1. Add all visible functions with a matching name in the calling scope to the set of function candidates.
2. Eliminate functions whose profile conflicts with the current compilation profile.
3. Eliminate functions with the wrong number of formal parameters. If a candidate function has excess formal parameters, and each of the excess parameters has a default value, do not eliminate the function.
4. If the set is empty, fail.
5. For each actual parameter expression in sequence (left to right), perform the following:
   1. If the type of the actual parameter matches the unqualified type of the corresponding formal parameter in any function in the set, remove all functions whose corresponding parameter does not match exactly.
   2. If there is a function with a dynamically typed formal argument which is compatible with the actual parameter type, remove all functions whose corresponding parameter is not similarly compatible.
   3. If there is a defined promotion for the type of the actual parameter to the unqualified type of the formal parameter of any function, remove all functions for which this is not true from the set.
4. If there is a valid implicit cast that converts the type of the actual parameter to the unqualified type of the formal parameter of any function, remove all functions for which this is not true from the set.

5. Fail.

6. Choose a function based on profile:
   1. If there is at least one function with a profile that exactly matches the compilation profile, discard all functions that don’t exactly match.
   2. Otherwise, if there is at least one function with a wildcard profile that matches the compilation profile, determine the ‘most specific’ matching wildcard profile in the candidate set. Discard all functions except those with this ‘most specific’ wildcard profile. How ‘specific’ a given wildcard profile name is relative to a particular profile is determined by the profile specification.

7. If the number of functions remaining in the set is not one, then fail.

Global Variables

Global variables are declared and used as in C. Non-static variables may have a semantic associated with them. Uniform non-static variables may have their value set through the run-time API.

Use of Uninitialized Variables

It is incorrect for a program to use an uninitialized static or local variable. However, the compiler is not obligated to detect such errors, even if it would be possible to do so by compile-time data-flow analysis. The value obtained from reading an uninitialized variable is undefined. This same rule applies to the implicit use of a variable that occurs when it is returned by a top-level function. In particular, if a top-level function returns a struct, and some element of that struct is never written, then the value of that element is undefined.

Note: The language designers did not choose to define variables as being initialized to zero because that would result in a performance penalty in cases where the compiler is unable to determine if a variable is properly initialized by the programmer.

Preprocessor

Cg profiles must support the full ANSI C standard preprocessor capabilities: #if, #define, and so on. However, while #include must be supported the mechanism by which the file to be included is located is implementation defined.

Overview of Binding Semantics

In stream-processing architectures, data packets flow between different programmable units. On a GPU, for example, packets of vertex data flow from the application to the vertex program.

Because packets are produced by one program (the application, in this case), and consumed by another (the vertex program), there must be some mechanism for defining the interface between the two. Cg allows the user to choose between two different approaches to defining these interfaces.

The first approach is to associate a binding semantic with each element of the packet. This approach is a bind-by-name approach. For example, an output with the binding semantic FOO is fed to an input with the binding semantic FOO. Profiles may allow the user to define arbitrary identifiers in this “semantic namespace”, or they may restrict the allowed identifiers to a predefined set. Often, these predefined names correspond to the names of hardware registers or API resources.

In some cases, predefined names may control non-programmable parts of the hardware. For example, vertex programs normally compute a position that is fed to the rasterizer, and this position is stored in an output with the binding semantic POSITION.

For any profile, there are two namespaces for predefined binding semantics—the namespace used for in variables and the namespace used for out variables. The primary implication of having two namespaces is that the binding semantic cannot be used to implicitly specify whether a variable is in or out.
The second approach to defining data packets is to describe the data that is present in a packet and allow the compiler to decide how to store it. In Cg, the user can describe the contents of a data packet by placing all of its contents into a `struct`. When a `struct` is used in this manner, we refer to it as a connector. The two approaches are not mutually exclusive, as is discussed later. The connector approach allows the user to rely on a combination of user-specified semantic bindings and compiler-determined bindings.

### Binding Semantics

A binding semantic may be associated with an input to a top-level function or a global variable in one of three ways:

- The binding semantic is specified in the formal parameter declaration for the function. The syntax for formal parameters to a function is:
  ```
  [const] [in | out | inout] <type> <identifier> [: <binding-semantic>] [= <initializer>];
  ```

- If the formal parameter is a `struct`, the binding semantic may be specified with an element of the `struct` when the `struct` is defined:
  ```
  struct <struct-tag> {
    <type> <identifier>[ : <binding-semantic>];
    ...
  };
  ```

- If the input to the function is implicit (a non-static global variable that is read by the function), the binding semantic may be specified when the non-static global variable is declared:
  ```
  [varying [in | out]] <type> <identifier> [ : <binding-semantic>];
  ```

  If the non-static global variable is a `struct`, the binding semantic may be specified when the `struct` is defined, as described in the second bullet above.

- A binding semantic may be associated with the output of a top-level function in a similar manner:
  ```
  <type> <identifier> ( <parameter-list> ) [: <binding-semantic>] {
  ```

Another method available for specifying a semantic for an output value is to return a `struct`, and to specify the binding semantic(s) with elements of the `struct` when the `struct` is defined. In addition, if the output is a formal parameter, then the binding semantic may be specified using the same approach used to specify binding semantics for inputs.

### Aliasing of Semantics

Semantics must honor a copy-on-input and copy-on-output model. Thus, if the same input binding semantic is used for two different variables, those variables are initialized with the same value, but the variables are not aliased thereafter. Output aliasing is illegal, but implementations are not required to detect it. If the compiler does not issue an error on a program that aliases output binding semantics, the results are undefined.

### Additional Details for Binding Semantics

The following are somewhat redundant, but provide extra clarity:

- Semantic names are case-insensitive.
- Semantics attached to parameters to non-main functions are ignored.
- Input semantics may be aliased by multiple variables.
- Output semantics may not be aliased.

### Using a Structure to Define Binding Semantics (Connectors)

Cg profiles may optionally allow the user to avoid the requirement that a binding semantic be specified for every non-uniform input (or output) variable to a top-level program. To avoid this requirement, all the non-uniform variables should be included within a single `struct`. The compiler automatically allocates the elements of this structure to
hardware resources in a manner that allows any program that returns this struct to interoperate with any program that uses this struct as an input.

It is not required that all non-uniform inputs be included within a single struct in order to omit binding semantics. Binding semantics may be omitted from any input or output, and the compiler performs automatic allocation of that input or output to a hardware resource. However, to guarantee interoperability of one program’s output with another program’s input when automatic binding is performed, it is necessary to put all of the variables in a single struct.

It is permissible to explicitly specify a binding semantic for some elements of the struct, but not others. The compiler’s automatic allocation must honor these explicit bindings. The allowed set of explicitly specified binding semantics is defined by the allocation-rule identifier. The most common use of this capability is to bind variables to hardware registers that write to, or read from, non-programmable parts of the hardware. For example, in a typical vertex-program profile, the output struct would contain an element with an explicitly specified POSITION semantic. This element is used to control the hardware rasterizer.

Defining Binding Semantics via an external API

It may be possible to define binding semantics on inputs and outputs by using an external API that manipulates the programs environment. The Cg Runtime API is such an API that allows this, and others may exist.

How Programs Receive and Return Data

A program is a non-static function that has been designated as the main entry point at compilation time. The varying inputs to the program come from this top-level function’s varying in parameters, and any global varying variables that do not have an out modifier. The uniform inputs to the program come from the top-level function’s uniform in parameters and from any non-static global variables that are referenced by the top-level function or by any functions that it calls. The output of the program comes from the return value of the function (which is always implicitly varying), from any out parameters, which must also be varying, and from any varying out global variables that are written by the program.

Parameters to a program of type sampler* are implicitly const.

Statements and Expressions

Statements are expressed just as in C, unless an exception is stated elsewhere in this document. Additionally,

- if, while, and for require bool expressions in the appropriate places.
- Assignment is performed using =. The assignment operator returns a value, just as in C, so assignments may be chained.
- The new discard statement terminates execution of the program for the current data element (such as the current vertex or current fragment) and suppresses its output. Vertex profiles may choose to omit support for discard.

Minimum Requirements for if, while, for

The minimum requirements are as follows:

- All profiles should support if, but such support is not strictly required for older hardware.
- All profiles should support for and while loops if the number of loop iterations can be determined at compile time. “Can be determined at compile time” is defined as follows: The loop-iteration expressions can be evaluated at compile time by use of intra-procedural constant propagation and folding, where the variables through which constant values are propagated do not appear as lvalues within any kind of control statement (if, for, or while) or ?: construct. Profiles may choose to support more general constant propagation techniques, but such support is not required.
- Profiles may optionally support fully general for and while loops.

New Vector Operators

These new operators are defined for vector types:
• Vector construction operator: `typeID(...)`:
  This operator builds a vector from multiple scalars or shorter vectors:
  – `float4(scalar, scalar, scalar, scalar)`
  – `float4(float3, scalar)`

• Matrix construction operator: `typeID(...)`:
  This operator builds a matrix from multiple rows.
  Each row may be specified either as multiple scalars or as any combination of scalars and vectors with the
  appropriate size, e.g.
  `float3x3(1, 2, 3, 4, 5, 6, 7, 8, 9)`
  `float3x3(float3, float3, float3)`
  `float3x3(1, float2, float3, 1, 1, 1)`

• Vector swizzle operator: `(.)`
  `a = b.xxyz; // A swizzle operator example`
  – At least one swizzle character must follow the operator.
  – There are three sets of swizzle characters and they may not be mixed: Set one is `xyzw = 0123`, set two
    is `rgba = 0123`, and set three is `stpq = 0123`.
  – The vector swizzle operator may only be applied to vectors or to scalars.
  – Applying the vector swizzle operator to a scalar gives the same result as applying the operator to a vector
    of length one. Thus, `myscalar.xxx` and `float3(myscalar, myscalar, myscalar)` yield
    the same value.
  – If only one swizzle character is specified, the result is a scalar not a vector of length one. Therefore, the
    expression `b.y` returns a scalar.
  – Care is required when swizzling a constant scalar because of ambiguity in the use of the decimal point
    character. For example, to create a three-vector from a scalar, use one of the following: `(1).xxx` or
    `1.xxx` or `1.0.xxx` or `1.0f.xxx`
  – The size of the returned vector is determined by the number of swizzle characters. Therefore, the size of the
    result may be larger or smaller than the size of the original vector. For example, `float2(0,1).xxyy` and
    `float4(0,0,1,1)` yields the same result.

• Matrix swizzle operator:
  For any matrix type of the form `<type><rows>x<columns>`, the notation: `<matrixObject>._<row><col>[_<row><col>][...]]` can be used to access individual matrix elements (in the case
  of only one `<row>,<col>` pair) or to construct vectors from elements of a matrix (in the case of more than one
  `<row>,<col>` pair). The row and column numbers are zero-based.
  For example:
  `float4x4 myMatrix;`
  `float myFloatScalar;`
  `float4 myFloatVec4;`

  // Set myFloatScalar to myMatrix[3][2]
  myFloatScalar = myMatrix._m32;

  // Assign the main diagonal of myMatrix to myFloatVec4
  myFloatVec4 = myMatrix._m00_m11_m22_m33;

  For compatibility with the D3DMatrix data type, Cg also allows one-based swizzles, using a form with the `m`
  omitted after the `_`: `<matrixObject>._<row><col>[...]]` In this form, the indexes for `<row>` and
  `<col>` are one-based, rather than the C standard zero-based. So, the two forms are functionally equivalent:

  `float4x4 myMatrix;`
  `float4 myVec;`

  // These two statements are functionally equivalent:
  myVec = myMatrix._m00_m23_m11_m31;
  myVec = myMatrix._11_34_22_42;
Because of the confusion that can be caused by the one-based indexing, its use is strongly discouraged. Also one-based indexing and zero-based indexing cannot be mixed in a single swizzle.

The matrix swizzles may only be applied to matrices. When multiple components are extracted from a matrix using a swizzle, the result is an appropriately sized vector. When a swizzle is used to extract a single component from a matrix, the result is a scalar.

- The write-mask operator: (.) It can only be applied to an lvalue that is a vector or matrix. It allows assignment to particular elements of a vector or matrix, leaving other elements unchanged. It looks exactly like a swizzle, with the additional restriction that a component cannot be repeated.

**Arithmetic Precision and Range**

Some hardware may not conform exactly to IEEE arithmetic rules. Fixed-point data types do not have IEEE-defined rules.

Optimizations are permitted to produce slightly different results than unoptimized code. Constant folding must be done with approximately the correct precision and range, but is not required to produce bit-exact results. It is recommended that compilers provide an option either to forbid these optimizations or to guarantee that they are made in bit-exact fashion.

**Operator Precedence**

Cg uses the same operator precedence as C for operators that are common between the two languages.

The swizzle and write-mask operators (.) have the same precedence as the structure member operator (. ) and the array index operator [ ].

**Operator Enhancements**

The standard C arithmetic operators (+, -, *, /, %, unary -) are extended to support vectors and matrices. Sizes of vectors and matrices must be appropriately matched, according to standard mathematical rules. Scalar-to-vector promotion, as described earlier, allows relaxation of these rules.

- \( M[n][m] \)  
  Matrix with \( n \) rows and \( m \) columns
- \( V[n] \)  
  Vector with \( n \) elements
- \(-V[n] \rightarrow V[n]\)  
  Unary vector negate
- \(-M[n] \rightarrow M[n]\)  
  Unary matrix negate
- \( V[n] \ast V[n] \rightarrow V[n]\)  
  Componentwise \( \ast \)
- \( V[n] / V[n] \rightarrow V[n]\)  
  Componentwise \(/\)
- \( V[n] \% V[n] \rightarrow V[n]\)  
  Componentwise \(\%\)
- \( V[n] + V[n] \rightarrow V[n]\)  
  Componentwise \(+\)
- \( V[n] - V[n] \rightarrow V[n]\)  
  Componentwise \(-\)
Componentwise -
M[n][m] * M[n][m] -> M[n][m]
Componentwise *
M[n][m] / M[n][m] -> M[n][m]
Componentwise /
M[n][m] % M[n][m] -> M[n][m]
Componentwise %
M[n][m] + M[n][m] -> M[n][m]
Componentwise +
M[n][m] - M[n][m] -> M[n][m]
Componentwise -

Operators

Boolean
&& || !

Boolean operators may be applied to bool packed bool vectors, in which case they are applied in elementwise fashion to produce a result vector of the same size. Each operand must be a bool vector of the same size.

Both sides of && and || are always evaluated; there is no short-circuiting as there is in C.

Comparisons
< > <= >= != ==

Comparison operators may be applied to numeric vectors. Both operands must be vectors of the same size. The comparison operation is performed in elementwise fashion to produce a bool vector of the same size.

Comparison operators may also be applied to bool vectors. For the purpose of relational comparisons, true is treated as one and false is treated as zero. The comparison operation is performed in elementwise fashion to produce a bool vector of the same size.

Comparison operators may also be applied to numeric or bool scalars.

Arithmetic
+ - * / % ++ -- unary- unary+

The arithmetic operator % is the remainder operator, as in C. It may only be applied to two operands of cint or int types.

When / or % is used with cint or int operands, C rules for integer / and % apply.

The C operators that combine assignment with arithmetic operations (such as +=) are also supported when the corresponding arithmetic operator is supported by Cg.

Conditional Operator
?:

If the first operand is of type bool, one of the following must hold for the second and third operands:

• Both operands have compatible structure types.
• Both operands are scalars with numeric or bool type.
- Both operands are vectors with numeric or bool type, where the two vectors are of the same size, which is less than or equal to four.

If the first operand is a packed vector of bool, then the conditional selection is performed on an elementwise basis. Both the second and third operands must be numeric vectors of the same size as the first operand.

Unlike C, side effects in the expressions in the second and third operands are always executed, regardless of the condition.

**Miscellaneous Operators**

(typecast) ,

Cg supports C’s typecast and comma operators.

**Reserved Words**

The following are currently used reserved words in Cg. A ‘*’ indicates that the reserved word is case-insensitive.

- __[anything] (i.e. any identifier with two underscores as a prefix)
- asm*
- asm_fragment
- auto
- bool
- break
- case
- catch
- char
- class
- column_major
- compile
- const
- const_cast
- continue
- decl*
- default
- delete
- discard
- do
- double
- dword*
- dynamic_cast
- else
- emit
enum
explicit
extern
false
fixed
float*
for
friend
get
goto
half
if
in
inline
inout
int
interface
long
matrix*
mutable
namespace
new
operator
out
packed
pass*
pixelfragment*
pixelshader*
private
protected
public
register
reinterpret_cast
return
row_major
sampler
sampler_state
sampler1D
sampler2D
sampler3D
samplerCUBE
shared
short
signed
sizeof
static
static_cast
string*
struct
switch
technique*
template
texture*
texture1D
texture2D
texture3D
textureCUBE
textureRECT
this
throw
true
try
typedef
typeid
typename
uniform
union
unsigned
using
vector*
vertexfragment*
vertexshader*
Cg Standard Library Functions

Cg provides a set of built-in functions and structures to simplify GPU programming. These functions are similar in spirit to the C standard library functions, providing a convenient set of common functions.

The Cg Standard Library functions are documented in the Cg Standard Library index.

VERTEX PROGRAM PROFILES

A few features of the Cg language that are specific to vertex program profiles are required to be implemented in the same manner for all vertex program profiles.

Mandatory Computation of Position Output

Vertex program profiles may (and typically do) require that the program compute a position output. This homogeneous clip-space position is used by the hardware rasterizer and must be stored in a program output with an output binding semantic of \texttt{POSITION} (or \texttt{HPOS} for backward compatibility).

Position Invariance

In many graphics APIs, the user can choose between two different approaches to specifying per-vertex computations: use a built-in configurable “fixed-function” pipeline or specify a user-written vertex program. If the user wishes to mix these two approaches, it is sometimes desirable to guarantee that the position computed by the first approach is bit-identical to the position computed by the second approach. This “position invariance” is particularly important for multipass rendering.

Support for position invariance is optional in Cg vertex profiles, but for those vertex profiles that support it, the following rules apply:

- Position invariance with respect to the fixed function pipeline is guaranteed if two conditions are met:
  - A \texttt{#pragma position_invariant <top-level-function-name>} appears before the body of the top-level function for the vertex program.
  - The vertex program computes position as follows:
    \[
    \text{OUT\_POSITION} = \text{mul}(\text{MVP}, \text{IN\_POSITION})
    \]
    where:
    \text{OUT\_POSITION}
    is a variable (or structure element) of type \texttt{float4} with an output binding semantic of \texttt{POSITION} or \texttt{HPOS}.
    \text{IN\_POSITION}
    is a variable (or structure element) of type \texttt{float4} with an input binding semantic of \texttt{POSITION}.
    \text{MVP}
    is a uniform variable (or structure element) of type \texttt{float4x4} with an input binding semantic that causes it to track the fixed-function modelview-projection matrix. (The name of this binding semantic is currently profile-specific – for OpenGL profiles, the semantic \texttt{state.matrix.mvp} is recommended).

- If the first condition is met but not the second, the compiler is encouraged to issue a warning.
- Implementations may choose to recognize more general versions of the second condition (such as the variables being copy propagated from the original inputs and outputs), but this additional generality is not required.
Binding Semantics for Outputs

As shown in Table 10, there are two output binding semantics for vertex program profiles:

Table 10  Vertex Output Binding Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Homogeneous clip-space position; fed to rasterizer.</td>
<td>float4</td>
<td>Undefined</td>
</tr>
<tr>
<td>PSIZE</td>
<td>Point size</td>
<td>float</td>
<td>Undefined</td>
</tr>
</tbody>
</table>

Profiles may define additional output binding semantics with specific behaviors, and these definitions are expected to be consistent across commonly used profiles.

FRAGMENT PROGRAM PROFILES

A few features of the Cg language that are specific to fragment program profiles are required to be implemented in the same manner for all fragment program profiles.

Binding semantics for outputs

As shown in Table 11, there are three output binding semantics for fragment program profiles:

Table 11  Fragment Output Binding Semantics

<table>
<thead>
<tr>
<th>Name</th>
<th>Meaning</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>RGBA output color</td>
<td>float4</td>
<td>Undefined</td>
</tr>
<tr>
<td>COLOR0</td>
<td>Same as COLOR</td>
<td>float</td>
<td>Interpolated depth from rasterizer (in range [0,1])</td>
</tr>
<tr>
<td>DEPTH</td>
<td>Fragment depth value</td>
<td>float</td>
<td>Interpolated depth from rasterizer (in range [0,1])</td>
</tr>
</tbody>
</table>

Profiles may define additional output binding semantics with specific behaviors, and these definitions are expected to be consistent across commonly used profiles.

If a program desires an output color alpha of 1.0, it should explicitly write a value of 1.0 to the \( W \) component of the COLOR output. The language does *not* define a default value for this output.

Note: If the target hardware uses a default value for this output, the compiler may choose to optimize away an explicit write specified by the user if it matches the default hardware value. Such defaults are not exposed in the language.)

In contrast, the language does define a default value for the DEPTH output. This default value is the interpolated depth obtained from the rasterizer. Semantically, this default value is copied to the output at the beginning of the execution of the fragment program.

As discussed earlier, when a binding semantic is applied to an output, the type of the output variable is not required to match the type of the binding semantic. For example, the following is legal, although not recommended:

```c
struct myfragoutput {
    float2 mycolor : COLOR;
}
```

In such cases, the variable is implicitly copied (with a typecast) to the semantic upon program completion. If the variable’s vector size is shorter than the semantic’s vector size, the larger-numbered components of the semantic receive their default values if applicable, and otherwise are undefined. In the case above, the \( R \) and \( G \) components of the output color are obtained from mycolor, while the \( B \) and \( A \) components of the color are undefined.
2.1 cgc

NAME
cgc - cg compiler driver

SYNOPSIS
cgc [options] file ...

DESCRIPTION
cgc is the standalone Cg compiler which translates Cg or GLSL programs into OpenGL or DirectX shader assembly code, or OpenGL or DirectX shading language code.

OPTIONS

Basic options

- entry name
  Sets the entry function for the shader to compile. Defaults to main.
-o file
  Sets the output file to be written. Default outputs to stdout.
-A file
  Sets the listing file, where error and warning messages are written. Defaults to stderr.
-profile name
  Selects the target profile, specifying the shader language to be generated.
-profileopts opt1,opt2,...
-po opt1,opt2,...
  Sets one or more profile specific options.
-noentry
  Sets check only mode, where no shader is compiled, but all the code in the input file is checked for syntactic correctness.
Language options

-oglsl
Sets the source language to GLSL.

-ogles
Sets the source language to OpenGL/ES GLSL.

-strict
-nostrict
Enable or disable strict typechecking, where most questionable constructs will be flagged as warnings.

-glsWerter
Like -strict but in addition, unportable GLSL constructs will be flagged as errors.

-nowarn
Disable all warnings.

-nowarn=N,N,...
Disable one or more specific numbered warnings.

-fx
-nofx
Enables or disables FX parsing mode, where FX keywords are recognized. Defaults to on in Cg mode and off in GLSL mode.

-nostdlib
Disable the standard library.

-no_uniform_blocks
Don’t allow defining constant buffers with the uniform keyword. Support BUFFER semantics on structs instead.

Code Generation Options

-fastmath
-nofastmath
Enable or disable optimizations that may change/lose precision in low order bits, such as associative transforms like (a + b) + c instead of a + (b + c). Default is -fastmath.

-fastprecision
-nofastprecision
Enable or disable optimzations doing operations at lower precision than is specified when the result is later converted to a lower precision or the operands were originally in lower precision. Default is -nofastprecision.

-bestprecision
Always do things in the best possible precision; only use lower precision operations if there is no possibility of difference. Implies -nofastmath and -nofastprecision.

-unroll all|none|count=N
Control loop unrolling. **-unroll all** will force unrolling of all loops that can be unrolled, while **-unroll none** will prevent unrolling except if code cannot otherwise be generated in the current profiles (so it will have no effect in profiles that don’t support looping). **unroll count=N** will unroll loops if the estimate of the resulting code is less than $N$ instructions. The estimate does not take into account further optimizations that may be done after unrolling, so it might be quite inaccurate.

- **-inline all|none|count=N**

Control function inlining. Setting **-inline none** will additionally disable inlining of functions with an explicit inline keyword, which are otherwise always inlined. Setting **-inline count=0** will effectively disable inlining of all functions that do not have an explicit inline keyword.

- **-ifcvt all|none|count=N**

Control if conversion (replacement of small if/else blocks with conditional assignments).

- **-O N**

Sets the optimization level of the compiler, from 0 (lowest) to 3 (highest). Higher values may produce better code and will cause compile time to increase. Default is **-O1**.

- **-looplimit N**

Assume loops that the compiler cannot determine an upper bound on the number of iterations may loop as many as $N$ iterations. This may require generating extra code for such loops in some profiles.

- **-d3d**

Generate code compatible with the Direct3D specification.

- **-MaxInstInBasicBlock N**

break basic blocks after $N$ instructions. This has an effect on local optimizations that don’t cross basic block boundaries and may avoid bad compile time blowups in the presence of huge basic blocks due to algorithms that are non-linear in the basic block size.

- **-maxunrollcount N**

Deprecated. Don’t unroll loops with more than $N$ iterations. Use the -unroll option instead, which provides better fine-grained control.

**Preprocessor Options**

- **-DMACRO[=VALUE]**

Sets a preprocessor macro. If VALUE is not specified it defaults to 1.

- **-Idirectory**

Adds a directory to the end of the search path for #include files. The default search path is empty.

- **-E**

Don’t compile, just preprocess the input.

- **-P**

With -E, suppresses the generation of #line directives in the output.

- **-C**

With -E, preserves comments in the output.

- **-MG**

Ignore #include files that can’t be found, rather than issuing an error.

- **-M**
-MM
-MD
-MMD
-MP
-MF file
-MT target
-MQ target

Generate dependency information about \#included files. These options are intended to be compatible with the options to gcc.

Miscellaneous Options

-quiet
-q

Supress all ‘noise’ output (copyright notices, indications of which files are being compiled, etc). With -o and -l, should result in no output being produced.

-nocode

Supress final code generation. Will actually run all the way through the compiler (so any errors present should be diagnosed), but don’t produce any actual output code.

-v
—version

Print compiler version information to listing.

-h
Print short option help summary to stdout and exit.

-help
Print longer option help summary to stdout, including all supported profiles and profile options, and exit.

-type type_definition
Set an override type binding for a variable.

-typefile file
Read override type bindings for variables from a file.

-dumpinputbind file
Dump type bindings for all variables to a file. This file may be passed back to the compiler with -typefile.

Debugging options

-debug
Enable the debug builtin function to abort operation of a shader and immediately output a value.

-debuglast
Like --debug, except the shader does not abort; instead it continues and outputs the value of the last debug function called.
-debugdefault=value

Like -debug, except if no debug call is reached, the output will be set to the specified value instead of what the shader normally computes.

-deprecated

Issue errors instead of warnings for any deprecated features used.

**PROFILES**

A *profile* specifies the output language of the cg compiler (either a shader assembly dialect, or a shading language). Each profile has its own set of *profile options* that can be set for it, though many related profiles have similar or identical options. Profiles can be grouped by program type, API, or GPU generation.

**DirectX profiles**

ds_5_0, gs_4_0, gs_5_0, hlslf, hlslv, hs_5_0, ps_1_1, ps_1_2, ps_1_3, ps_2_0, ps_2_x, ps_3_0, ps_4_0, ps_5_0, vs_1_1, vs_2_0, vs_2_x, vs_3_0, vs_4_0, vs_5_0

**OpenGL profiles**

arbfp1, arbvvp1, fp20, fp30, fp30unlimited, fp40, fp40unlimited, glslf, glslg, glslv, gp4fp, gp4gp, gp4vp, gp5fp, gp5gp, gp5tcp, gp5tep, gp5vp, vp20, vp30, vp40

**Fragment profiles**

arbfp1, fp20, fp30, fp30unlimited, fp40, fp40unlimited, glslf, gp4fp, gp5fp, hlslf, ps_1_1, ps_1_2, ps_1_3, ps_2_0, ps_2_x, ps_3_0, ps_4_0, ps_5_0

**Geometry profiles**

glslg, gp4gp, gp5gp, gs_4_0, gs_5_0

**Vertex profiles**

arbvp1, glslv, gp4vp, gp5vp, hlslv, vp20, vp30, vp40, vs_1_1, vs_2_0, vs_2_x, vs_3_0, vs_4_0, vs_5_0

**GeForce 3/4 Series profiles**

fp20, vp20

**GeForce 5 Series profiles**

fp30, vp30

**GeForce 6/7 Series profiles**

fp40, vp40

**GeForce 8/9/100/200/300 Series, OpenGL 3.x Quadro profiles**

gp4fp, gp4gp, gp4vp

**GeForce 400 Series, OpenGL 4.x Quadro profiles**

gp5fp, gp5gp, gp5tcp, gp5tep, gp5vp

**Profile options**

Here is a complete list of all profiles and their corresponding profile options

*arbfp1*

Targets the **ARB_frameprogram** OpenGL extension
-po ARB_draw_buffers
Use the **ARB_draw_buffers** option for multiple renderbuffer targets (MRT). This is the default

-po ATI_draw_buffers
Use the **ATI_draw_buffers** option for multiple renderbuffer targets (MRT).

- po MaxDrawBuffers=N
Set the maximum number of renderbuffer targets. Default is 1

- po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 32

- po MaxTexIndirections=N
Sets the maximum number of texture indirections allowed in the output program. Default is 1024

- po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 1024

- po NumMathInstructionSlots=N
Sets the maximum number of non-texture instructions in the output program. Default is 1024

- po NumTemps=N
Sets the maximum number of TEMP registers in the output program. Default is 32

- po NumTexInstructionSlots=N
Sets the maximum number of texture instructions in the output program. Default is 1024

*arbvp1*

Targets the **ARB_vertex_program** OpenGL extension

- po MaxAddressRegs=N
Sets the maximum number of ADDRESS registers in the output program. Default is 1

- po MaxInstructions=N
Sets the maximum number of instructions in the output program. Default is 1024

- po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 96

- po NumTemps=N
Sets the maximum number of TEMP registers in the output program. Default is 32

- po PosInv
Generate position invariant code (same as fixed-function) for POSITION output

*fp20*

Targets the **NV_register_combiners2** and **NV_texture_shader** OpenGL extensions

*fp30*

Targets the **NV_fragment_program** OpenGL extension
-po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 256
-po NumTemps=N
Sets the maximum number of temporaries in the output program. Default is 32

fp30unlimited
Same as $fp30$ with various hardware limits on registers and instructions lifted
-po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 4194304
-po NumTemps=N
Sets the maximum number of temporaries in the output program. Default is 512

$fp40$
Targets the $NV\_fragment\_program2$ OpenGL extension
-po appleKilWAR
Work around various bugs with KIL instructions in the OSX-tiger implementation of $NV\_fragment\_program2$
-po ARB_draw_buffers
Use the $ARB\_draw\_buffers$ option for multiple renderbuffer targets (MRT). This is the default
-po ATI_draw_buffers
Use the $ATI\_draw\_buffers$ option for multiple renderbuffer targets (MRT).
-po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is infinite
-po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is infinite
-po NumTemps=N
Sets the maximum number of TEMP registers in the output program. Default is infinite
-po OutColorPrec=N
If N is 3 or 4, force output to fp16 precision. If N is 2, force output to fp32 precision.

fp40unlimited
Same as $fp40$ with various hardware limits on registers and instructions lifted
-po appleKilWAR
Work around various bugs with KIL instructions in the OSX-tiger implementation of $NV\_fragment\_program2$
-po ARB_draw_buffers
Use the $ARB\_draw\_buffers$ option for multiple renderbuffer targets (MRT). This is the default
-po ATI_draw_buffers
Use the $ATI\_draw\_buffers$ option for multiple renderbuffer targets (MRT).
-po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 1024

```-po NumInstructionSlots=N```

Sets the maximum number of instructions in the output program. Default is 4194304

```-po NumTemps=N```

Sets the maximum number of TEMP registers in the output program. Default is 512

```-po OutColorPrec=N```

If $N$ is 3 or 4, force output to fp16 precision. If $N$ is 2, force output to fp32 precision.

**generic**

Produces a dump of the program in a non-executable format

`glsf, glslg` and `glslv`

Targets the OpenGL Shading language (GLSL) v1.10. `glsf` targets fragment programs while `glslv` targets vertex programs

```version=val```

GLSL version to target. Supported versions are 100, 110, 120, 130, 140, 150, 330, 400 and 410.

`userTexCoord`

Use user-defined varying instead of `gl_TexCoord`.

`ATI_draw_buffers`

Use `ATI_draw_buffers` extension for MRT.

`EXT_gpu_shader4`

Use `EXT_gpu_shader4` extension where useful.

**gp4fp**

Targets the `NV_gpu_program4` and `NV_fragment_program4` OpenGL extensions.

```-po fastimul```

Assume integer multiply inputs have at most 24 significant bits.

```-po NV_shader_buffer_load```

Use the `NV_shader_buffer_load` OpenGL extension.

```-po NV_parameter_buffer_object2```

Use the `NV_parameter_buffer_object2` OpenGL extension.

```-po PaBO2```

Use the `NV_parameter_buffer_object2` OpenGL extension.

```-po ARB_draw_buffers```

Use the `ARB_draw_buffers` option for multiple renderbuffer targets (MRT). This is the default

```-po ATI_draw_buffers```

Use the `ATI_draw_buffers` option for multiple renderbuffer targets (MRT).

```-po pixel_center_integer```

Use integer pixel centers.
-po origin_upper_left
  Use upper left pixel origin.

\textit{gp4gp}

Targets the \texttt{NV\_gpu\_program4} and \texttt{NV\_geometry\_program4} OpenGL extensions.
- po POINT
- po LINE
- po LINE\_ADJ
- po TRIANGLE
- po TRIANGLE\_ADJ
  Set the input primitive type for the geometry program
- po POINT\_OUT
- po LINE\_OUT
- po TRIANGLE\_OUT
  Set the output primitive type for the geometry program
- po Vertices=$N$
  Set the number of vertices output by the geometry program

\textit{gp4vp}

Targets the \texttt{NV\_gpu\_program4} and \texttt{NV\_vertex\_program4} OpenGL extensions.
- po PosInv
  Generate position invariant code (same as fixed-function) for POSITION output

\textit{gp5fp}

Targets the \texttt{NV\_gpu\_program5} OpenGL extension.
- po fastimul
  Assume integer multiply inputs have at most 24 significant bits.
- po NV\_shader\_buffer\_load
  Use the \texttt{NV\_shader\_buffer\_load} OpenGL extension.
- po NV\_parameter\_buffer\_object2
  Use the \texttt{NV\_parameter\_buffer\_object2} OpenGL extension.
- po PaBO2
  Use the \texttt{NV\_parameter\_buffer\_object2} OpenGL extension.
- po ARB\_draw\_buffers
  Use the \texttt{ARB\_draw\_buffers} option for multiple renderbuffer targets (MRT). This is the default
- po ATI\_draw\_buffers
  Use the \texttt{ATI\_draw\_buffers} option for multiple renderbuffer targets (MRT).
- po pixel\_center\_integer
Use the `ARB_fragment_coord_conventions` OpenGL extension to specify integer pixel centers.

-`po origin_upper_left`

Use the `ARB_fragment_coord_conventions` OpenGL extension to specify upper left pixel origin.

-`po NV_early_fragment_tests`

Perform depth and stencil tests prior to fragment program invocation.

**gp5gp**

Targets the `NV_gpu_program5` OpenGL extension.

-`po POINT`
-`po LINE`
-`po LINE_ADJ`
-`po TRIANGLE`
-`po TRIANGLE_ADJ`

Set the input primitive type for the geometry program

-`po POINT_OUT`
-`po LINE_OUT`
-`po TRIANGLE_OUT`

Set the output primitive type for the geometry program

-`po Vertices=N`

Set the number of vertices output by the geometry program

**gp5tcp**

Targets the `NV_tessellation_program` and `NV_gpu_program5` OpenGL extensions.

**gp5tep**

Targets the `NV_tessellation_program` and `NV_gpu_program5` OpenGL extensions.

**gp5vp**

Targets the `NV_gpu_program5` OpenGL extension.

-`po PosInv`

Generate position invariant code (same as fixed-function) for POSITION output

**hlslf hlsv**

Targets Microsoft High-Level Shading Language (HLSL). `hlslf` targets pixel programs while `hlsv` targets vertex programs

**ps_1_1 ps_1_2 ps_1_3**

Targets DirectX pixel programs

-`po MaxPixelShaderValue=N`

Maximum absolute value representable in a pixel shader. Default is 1.
**ps_2.0 ps_2.x**

Targets DirectX pixel programs

- `po MaxDrawBuffers=N`
  Set the maximum number of renderbuffer targets. Default is 1
- `po NumInstructionSlots=N`
  Sets the maximum number of instructions in the output program. Default is 96 or 512
- `po NumTemps=N`
  Sets the maximum number of temporaries in the output program. Default is 12 or 32

**ps_3.0**

Targets DirectX pixel programs

- `po MaxDrawBuffers=N`
  Set the maximum number of renderbuffer targets. Default is 1
- `po MaxLocalParams=N`
  Set the maximum number of uniform parameter slots available. Default is 224
- `po NumInstructionSlots=N`
  Sets the maximum number of instructions in the output program. Default is 32768
- `po NumTemps=N`
  Sets the maximum number of temporaries in the output program. Default is 32
- `po OutColorPrec=N`
  If $N$ is 3 or 4, force output to fp16 precision. If $N$ is 2, force output to fp32 precision.

**vp20**

Targets the **NV_vertex_program** OpenGL extension

- `po MaxLocalParams=N`
  Set the maximum number of uniform parameter slots available. Default is 96
- `po PosInv`
  Generate position invariant code (same as fixed-function) for POSITION output

**vp30**

Targets the **NV_vertex_program2** OpenGL extension

- `po MaxLocalParams=N`
  Set the maximum number of uniform parameter slots available. Default is 256
- `po PosInv`
  Generate position invariant code (same as fixed-function) for POSITION output

**vp40**

Targets the **NV_vertex_program3** OpenGL extension
-po MaxAddressRegs=N
Sets the maximum number of ADDRESS registers in the output program. Default is 2
-po MaxInstructions=N
Sets the maximum number of instructions in the output program. Default is 2048
-po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 544
-po NumTemps=N
Sets the maximum number of TEMP registers in the output program. Default is 32
-po PosInv
Generate position invariant code (same as fixed-function) for POSITION output

vs_1_1
Targets DirectX vertex programs
- po dcls
Output dx9-style dcls statements
- po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 96
- po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 128
- po NumTemps=N
Sets the maximum number of temporaries in the output program. Default is 12

vs_2_0 vs_2_x
Targets DirectX vertex programs
- po dcls
Output dx9-style dcls statements
- po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 256
- po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 256
- po NumTemps=N
Sets the maximum number of temporaries in the output program. Default is 12

vs_3_0
Targets DirectX vertex programs
- po dcls
Output dx9-style dcls statements
- po MaxLocalParams=N
Set the maximum number of uniform parameter slots available. Default is 256
-po NumInstructionSlots=N
Sets the maximum number of instructions in the output program. Default is 32768
-po NumTemps=N
Sets the maximum number of temporaries in the output program. Default is 32

ENVIRONMENT

SEE ALSO

Cg Language Specification, arbf1, arbyp1, fp20, fp30, fp40, glslf, glslv, gp4fp, gp4gp, gp4vp, hlslf, hlslv, vp20, vp30, vp40

2.2 cgfxcat

NAME
cgfxcat - dump a Cg effect or program file to standard out

SYNOPSIS

DESCRIPTION
cgfxcat loads an effect or program file and calls every query operation available in the core runtime on the resulting Cg object. The values returned by the queries are written to standard out.

The source code for cgfxcat is found under examples/Tools/cgfxcat.

OPTIONS

file
The effect or program file to be processed. cgfxcat can handle effect files, program files, or precompiled program files.

Files with a .cgfx or .fx extension will be loaded using cgCreateEffectFromFile. Files with a .cg or .hlslf extension will be loaded using cgCreateProgramFromFile. Any other extension is passed to cgGetProfile and if a valid CGprofile is found the file is loaded using cgCreateProgramFromFile with the program_type set to CG_OBJECT. You can override these built-in file extension mappings with the -effect, -program, or -object options.

-effect
Treat file as an effect file, ignoring its extension.

-program
Treat file as a program file, ignoring its extension.

-object
Treat file as a precompiled program file, ignoring its extension.

-profile profile
Use the CGprofile returned by cgGetProfile for profile as the profile argument of cgCreateProgramFromFile when compiling file.

-entry entry
Use **entry** as the **entry** argument of **cgCreateProgramFromFile** when compiling **file**.

- **gl**
  Register states with **cgGLRegisterStates** rather than using the generic state code built into **cgfxcat**.

SEE ALSO

cgCreateEffectFromFile, cgCreateProgramFromFile, cgGetProfile, cgGLRegisterStates

### 2.3 cginfo

**NAME**

cginfo - print Cg library information

**SYNOPSIS**

cginfo [-help] [-profiles] [/path/to/library]

**DESCRIPTION**

cginfo will print a Cg library’s version string to the standard output along with the path to the library from which the version was retrieved. The list of supported profiles can also be displayed.

The source code for cginfo is found under examples/Tools/cginfo. This example demonstrates how to load the Cg library and get the address of a symbol from the library for all of the platforms on which Cg is supported.

**OPTIONS**

- **-help**
  Print a description of the command line options understood by cginfo on the standard output.

- **-profiles**
  Also print a list of profiles supported by this library on the standard output. Note that the APIs to enumerate supported profiles were introduced in Cg 2.2. If the library being queried is 2.1 or earlier then this option will have no effect.

- **/path/to/library**
  Platform specific path the library to be queried. If no explicit library path is given the platform specific rules for finding shared libraries will be used to locate a copy of the Cg library to query.

SEE ALSO

Cg_language
CHAPTER THREE

CG PROFILES

3.1 arbfp1

NAME

arbfp1 - OpenGL fragment profile for multi-vendor ARB_fragment_program extension

SYNOPSIS

arbfp1

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by GeForce FX and other DirectX 9 GPUs. This profile is supported by any OpenGL implementation that conformantly implements ARB_fragment_program.

The compiler output for this profile conforms to the assembly format defined by ARB_fragment_program.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES

Requires OpenGL support for the multi-vendor ARB_fragment_program extension. This extension is supported by GeForce FX and later GPUs. ATI GPUs also support this extension.

OpenGL Extension Specifications

GL_ARB_fragment_program

PROFILE OPTIONS

NumTemps=n

Number of temporaries to use, 32 by default.

NumInstructionSlots=n

Maximum allowable (static) instructions. Not an issue for NVIDIA GPUs.

NumTexInstructionSlots=n

Maximum number of texture instructions to generate. Not an issue for NVIDIA GPUs, but important for ATI GPUs (set it to 32).
NumMathInstructionSlots=n
Maximum number of math instructions to generate. Not an issue for NVIDIA GPUs, but important for
ATI GPUs (set it to 64).
MaxTexIndirections=n
Maximum number of texture indirections. Not an issue for NVIDIA GPUs, but important for ATI GPUs
(set it to 4).
ATI_draw_buffers
When multiple draw buffers are used, use the **ATI_draw_buffers** syntax so the generated code
says **OPTION ATI_draw_buffers**. The default, if this option is not specified, is to use
**ARB_draw_buffers**.
ARB_draw_buffers
When multiple draw buffers are used, use the **ARB_draw_buffers** syntax so the generated code says
**OPTION ARB_draw_buffers**. This option is the default.
MaxDrawBuffers=n
Maximum draw buffers for use with **ARB_draw_buffers**. Set to 1 for NV3x GPUs. Use up to 4 for
NV4x or ATI GPUs.
MaxLocalParams=n
Maximum allowable local parameters.
pixel_center_integer=b
Boolean to enable integer pixel centers.
origin_upper_left=b
Boolean to enable upper left pixel origin.

DATA TYPES
to-be-written

SEMANTICS

VARYING INPUT SEMANTICS
to-be-written

UNIFORM INPUT SEMANTICS
to-be-written

OUTPUT SEMANTICS
to-be-written

STANDARD LIBRARY ISSUES
to-be-written

SEE ALSO

`arbvp1`
3.2 arbvp1

NAME
arbvp1 - OpenGL vertex profile for multi-vendor ARB_vertex_program extension

SYNOPSIS
arbvp1

DESCRIPTION
This OpenGL profile corresponds to the per-vertex functionality introduced by GeForce3. This profile is supported by any OpenGL implementation that conformantly implements ARB_vertex_program.

The compiler output for this profile conforms to the assembly format defined by ARB_vertex_program.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is supported; but texture accesses are not supported.

3D API DEPENDENCIES
Requires OpenGL support for the multi-vendor ARB_vertex_program extension. These extensions were introduced by GeForce3 and Quadro DCC GPUs. ATI GPUs also support this extension.

OpenGL Extension Specifications
GL_ARB_vertex_program

PROFILE OPTIONS

NumTemp=\textup{n}
Number of temporaries to use, 32 by default.

MaxInstructions=\textup{n}
Maximum allowable (static) instructions.

MaxLocalParams=\textup{n}
Maximum allowable local parameters.

MaxAddressRegs=\textup{n}
Maximum allowable address registers.

PosInv=b
Boolean to enable position invariance.

DATA TYPES
to-be-written

SEMANTICS

VARYING INPUT SEMANTICS
to-be-written
UNIFORM INPUT SEMANTICS
to-be-written

OUTPUT SEMANTICS
to-be-written

STANDARD LIBRARY ISSUES
to-be-written

SEE ALSO
arbfp1

3.3 ds_5_0

NAME
ds_5_0 - Translation profile to DirectX 11’s High Level Shader Language for domain shaders.

SYNOPSIS
ds_5_0

DESCRIPTION
This Direct3D profile translates Cg into DirectX 11’s High Level Shader Language (HLSL11) for domain shaders. The compiler output for this profile conforms to the textual high-level language defined by DirectX 11’s High Level Shading Language.

The limitations of the ds_5_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES
Requires Direct3D 11 support.


PROFILE OPTIONS
-po pad16 This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL11 normally uses.

DATA TYPES
In general, the Cg data types translate to the HLSL11 data types with the same name.

float, half, fixed

These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

int

This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.
SEMANTICS

INPUT SEMANTICS

The varying input semantics in the ds_5_0 profile correspond to the respectively named varying output semantics of the hs_5_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4) (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
</tbody>
</table>

UNIFORM INPUT SEMANTICS

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Clip-space position (SV_Position)</td>
</tr>
<tr>
<td>HPOS</td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>Front primary color (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Front secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
<tr>
<td>TEX#</td>
<td>TEX# is translated to TEXCOORD#</td>
</tr>
<tr>
<td>FOGC</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
<tr>
<td>PSIZE</td>
<td>Point size</td>
</tr>
<tr>
<td>PSIZ2</td>
<td></td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL11 for domain shaders. In general, the Cg and HLSL11 standard libraries are very similar.

SEE ALSO

ps_5_0, vs_5_0, gs_5_0, hs_5_0
3.4 fp20

NAME

fp20 - OpenGL fragment profile for NV2x (GeForce3, GeForce4 Ti, Quadro DCC, etc.)

SYNOPSIS

fp20

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by GeForce3.

The capabilities of this profile are quite limited.

The compiler output for this profile conforms to the `nvparse` file format for describing `NV_register_combiners` and `NV_texture_shader` state configurations.

3D API DEPENDENCIES

Requires OpenGL support for `NV_texture_shader`, `NV_texture_shader2`, and `NV_register_combiners2` extensions. These extensions were introduced by GeForce3 and Quadro DCC GPUs.

Some standard library functions may require `NV_texture_shader3`. This extension was introduced by GeForce4 Ti and Quadro4 XGL GPUs.

OpenGL Extension Specifications

- `GL_NV_register_combiners`
- `GL_NV_register_combiners2`
- `GL_NV_texture_shader`
- `GL_NV_texture_shader2`

PROFILE OPTIONS

None.

DATA TYPES

- fixed

  The `fixed` data type corresponds to a native signed 9-bit integers normalized to the [-1.0,+1.0] range.

- float

- half

  In most cases, the `float` and `half` data types are mapped to `fixed` for math operations.

  Certain built-in standard library functions corresponding to `NV_texture_shader` operations operate at 32-bit floating-point precision.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the `fp20` profile correspond to the respectively named varying output semantics of the `vp20` profile.
<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Name Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>

**OUTPUT SEMANTICS**

| COLOR                  | Output color (float4) |
| COLOR0                 |                        |
| COL                    |                        |
| COL0                   |                        |
| DEPTH                  | Output depth (float)   |
| DEPR                   |                        |

**STANDARD LIBRARY ISSUES**

There are a lot of standard library issues with this profile.

Because the ‘fp20’ profile has limited capabilities, not all of the Cg standard library functions are supported. The list below presents the Cg standard library functions that are supported by this profile. See the standard library documentation for descriptions of these functions.

dot(floatN, floatN)
lerp(floatN, floatN, floatN)
lerp(floatN, floatN, float)
tex1D(sampler1D, float)
tex1D(sampler1D, float2)
tex1Dproj(sampler1D, float2)
tex1Dproj(sampler1D, float3)
tex2D(sampler2D, float2)
tex2D(sampler2D, float3)
tex2Dproj(sampler2D, float3)
texRECT(samplerRECT, float4)
texRECT(samplerRECT, float2)
texRECT(samplerRECT, float3)
texRECTproj(samplerRECT, float3)
texRECTproj(samplerRECT, float4)
tex3D(sampler3D, float3)
tex3Dproj(sampler3D, float4)
texCUBE(samplerCUBE, float3)
texCUBEproj(samplerCUBE, float4)

Note: The non-projective texture lookup functions are actually done as projective lookups on the underlying hardware. Because of this, the ‘w’ component of the texture coordinates passed to these functions from the application or vertex program must contain the value 1.

Texture coordinate parameters for projective texture lookup functions must have swizzles that match the swizzle done by the generated texture shader instruction. While this may seem burdensome, it is intended to allow ‘fp20’ profile programs to behave correctly under other pixel shader profiles. The list below shows the swizzles required on the texture coordinate parameter to the projective texture lookup functions.

<table>
<thead>
<tr>
<th>Texture lookup function</th>
<th>Texture coordinate swizzle</th>
</tr>
</thead>
<tbody>
<tr>
<td>tex1Dproj</td>
<td>.xw/.ra</td>
</tr>
<tr>
<td>tex2Dproj</td>
<td>.xyw/.rga</td>
</tr>
<tr>
<td>texRECTproj</td>
<td>.xyw/.rga</td>
</tr>
<tr>
<td>tex3Dproj</td>
<td>.xyzw/.rgba</td>
</tr>
<tr>
<td>texCUBEproj</td>
<td>.xyzw/.rgba</td>
</tr>
</tbody>
</table>

TEXTURE SHADER OPERATIONS

In order to take advantage of the more complex hard-wired shader operations provided by NV_texture_shader, a collection of built-in functions implement the various shader operations.

offsettex2D
offsettexRECT

offsettex2D(uniform sampler2D tex,
float2 st,
float4 prevlookup,
uniform float4 m)

offsettexRECT(uniform samplerRECT tex,
float2 st,
float4 prevlookup,
uniform float4 m)

Performs the following

```c
float2 newst = st + m.xy * prevlookup.xx + m.zw * prevlookup.yy;
return tex2D/RECT(tex, newst);
```

where ‘st’ are texture coordinates associated with sampler ‘tex’, ‘prevlookup’ is the result of a previous texture operation, and ‘m’ is the offset texture matrix. This function can be used to generate the ‘offset_2d’ or ‘offset_rectangle’ NV_texture_shader instructions.

offsettex2DScaleBias
offsettexRECTScaleBias

offsettex2DScaleBias(uniform sampler2D tex,
float2 st,
float4 prevlookup,
uniform float4 m,
uniform float scale,
uniform float bias)

offsettexRECTScaleBias(uniform samplerRECT tex,
float2 st,
float4 prevlookup,
uniform float4 m,
uniform float scale,
uniform float bias)

Performs the following
float2 newst = st + m.xy * prevlookup.xx + m.zw * prevlookup.yy;
float4 result = tex2D/RECT(tex, newst);
return result * saturate(prevlookup.z * scale + bias);

where ‘st’ are texture coordinates associated with sampler ‘tex’, ‘prevlookup’ is the result of a previous texture operation, ‘m’ is the offset texture matrix, ‘scale’ is the offset texture scale and ‘bias’ is the offset texture bias. This function can be used to generate the ‘offset_2d_scale’ or ‘offset_rectangle_scale’ NV_texture_shader instructions.

tex1D_dp3(sampler1D tex, float3 str, float4 prevlookup

tex1D_dp3(sampler1D tex, float3 str, float4 prevlookup

Performs the following
return tex1D(tex, dot(str, prevlookup.xyz));

where ‘str’ are texture coordinates associated with sampler ‘tex’ and ‘prevlookup’ is the result of a previous texture operation. This function can be used to generate the ‘dot_product_1d’ NV_texture_shader instruction.

tex2D_dp3x2
texRECT_dp3x2

tex2D_dp3x2(uniform sampler2D tex, float3 str, float4 intermediate_coord, float4 prevlookup)
texRECT_dp3x2(uniform samplerRECT tex, float3 str, float4 intermediate_coord, float4 prevlookup)

Performs the following
float2 newst = float2(dot(intermediate_coord.xyz, prevlookup.xyz),
dot(str, prevlookup.xyz));
return tex2D/RECT(tex, newst);

where ‘str’ are texture coordinates associated with sampler ‘tex’, ‘prevlookup’ is the result of a previous texture operation and ‘intermediate_coord’ are texture coordinates associated with the previous texture unit. This function can be used to generate the ‘dot_product_2d’ or ‘dot_product_rectangle’ NV_texture_shader instruction combinations.

tex3D_dp3x3
texCUBE_dp3x3
\texttt{tex3D\_dp3x3(sampler3D\ tex,\
float3\ str,\
float4\ intermediate\_coord1,\
float4\ intermediate\_coord2,\
float4\ prevlookup)}

\texttt{texCUBE\_dp3x3(samplerCUBE\ tex,\
float3\ str,\
float4\ intermediate\_coord1,\
float4\ intermediate\_coord2,\
float4\ prevlookup)}

Performs the following

\[
\text{float3}\ newst = \text{float3}(\text{dot}(\text{intermediate\_coord1}.xyz, \text{prevlookup}.xyz),\
\text{dot}(\text{intermediate\_coord2}.xyz, \text{prevlookup}.xyz),\
\text{dot}(\text{str}, \text{prevlookup}.xyz));
\]

\textbf{return} \ \text{tex3D/CUBE(tex, newst)};

where ‘str’ are texture coordinates associated with sampler ‘tex’, ‘prevlookup’ is the result of a previous
texture operation, ‘intermediate\_coord1’ are texture coordinates associated with the ‘n-2’ texture unit and
‘intermediate\_coord2’ are texture coordinates associated with the ‘n-1’ texture unit. This function can
be used to generate the ‘dot\_product\_3d’ or ‘dot\_product\_cube\_map’ \textbf{NV\_texture\_shader} instruction
combinations.

\texttt{texCUBE\_reflect\_dp3x3}

\texttt{texCUBE\_reflect\_dp3x3(uniform\ samplerCUBE\ tex,\
float4\ strq,\
float4\ intermediate\_coord1,\
float4\ intermediate\_coord2,\
float4\ prevlookup)}

Performs the following

\[
\text{float3}\ E = \text{float3}(\text{intermediate\_coord2}.w, \text{intermediate\_coord1}.w,\
\text{strq}.w);\
\text{float3}\ N = \text{float3}(\text{dot}(\text{intermediate\_coord1}.xyz, \text{prevlookup}.xyz),\
\text{dot}(\text{intermediate\_coord2}.xyz, \text{prevlookup}.xyz),\
\text{dot}(\text{strq}.xyz, \text{prevlookup}.xyz));
\]

\textbf{return} \ \text{texCUBE(tex, 2 * \text{dot}(N, E) / \text{dot}(N, N) * N - E)};

where ‘strq’ are texture coordinates associated with sampler ‘tex’, ‘prevlookup’ is the result of a previous
texture operation, ‘intermediate\_coord1’ are texture coordinates associated with the ‘n-2’ texture unit and
‘intermediate\_coord2’ are texture coordinates associated with the ‘n-1’ texture unit. This function can
be used to generate the ‘dot\_product\_reflect\_cube\_map\_eye\_from\_qs’ \textbf{NV\_texture\_shader} instruction
combination.

\texttt{texCUBE\_reflect\_eye\_dp3x3}

\texttt{texCUBE\_reflect\_eye\_dp3x3(uniform\ samplerCUBE\ tex,\
float3\ str,\
float4\ intermediate\_coord1,\
float4\ intermediate\_coord2,\
float4\ prevlookup,\
uniform\ float3\ eye)}

Performs the following
float3 \( N = \) float3(dot(intermediate_coord1.xyz, prevlookup.xyz), dot(intermediate_coord2.xyz, prevlookup.xyz), dot(coords.xyz, prevlookup.xyz));

\[ \text{return } \text{texCUBE(tex, } 2 \times \text{dot(N, E)} / \text{dot(N, N)} \times \text{N - E)}; \]

where 'strq' are texture coordinates associated with sampler 'tex', 'prevlookup' is the result of a previous texture operation, 'intermediate_coord1' are texture coordinates associated with the 'n-2' texture unit, 'intermediate_coord2' are texture coordinates associated with the 'n-1' texture unit and 'eye' is the eye-ray vector. This function can be used to generate the 'dot_product_reflect_cube_map_const_eye' NV\_texture\_shader instruction combination.

tex\_dp3x2\_depth

\[ \text{tex_dp3x2_depth(float3 str, } \]
\[ \quad \text{float4 intermediate_coord, } \]
\[ \quad \text{float4 prevlookup) } \]

Performs the following

\[ \text{float } z = \text{dot(intermediate_coord.xyz, prevlookup.xyz); } \]
\[ \text{float } w = \text{dot(str, prevlookup.xyz); } \]
\[ \text{return } z / w; \]

where 'str' are texture coordinates associated with the 'n'th texture unit, 'intermediate_coord' are texture coordinates associated with the 'n-1' texture unit and 'prevlookup' is the result of a previous texture operation. This function can be used in conjunction with the 'DEPTH' varying out semantic to generate the 'dot_product_depth_replace' NV\_texture\_shader instruction combination.

EXPERIMENTS

The following examples illustrate how a developer can use Cg to achieve NV\_texture\_shader/NV\_register\_combiners functionality.

Example 1

\begin{verbatim}
struct VertexOut {
    float4 color : COLOR0;
    float4 texCoord0 : TEXCOORD0;
    float4 texCoord1 : TEXCOORD1;
};

float4 main(VertexOut IN,
    uniform sampler2D diffuseMap,
    uniform sampler2D normalMap) : COLOR
{
    float4 diffuseTexColor = tex2D(diffuseMap, IN.texCoord0.xy);
    float4 normal = 2 \times (\text{tex2D(normalMap, IN.texCoord1.xy) - 0.5});
    float3 light_vector = 2 \times (IN.color.rgb - 0.5);
    float4 dot_result = saturate(dot(light_vector, normal.xyz).xxxx);
    return dot_result \times diffuseTexColor;
}
\end{verbatim}

Example 2

\begin{verbatim}
struct VertexOut {
    float4 texCoord0 : TEXCOORD0;
    float4 texCoord1 : TEXCOORD1;
    float4 texCoord2 : TEXCOORD2;
    float4 texCoord3 : TEXCOORD3;
}
\end{verbatim}
float4 main(VertexOut IN, 
    uniform sampler2D normalMap, 
    uniform sampler2D intensityMap, 
    uniform sampler2D colorMap) : COLOR
{
    float4 normal = 2 * (tex2D(normalMap, IN.texCoord0.xy) - 0.5);
    float2 intensCoord = float2(dot(IN.texCoord1.xyz, normal.xyz), 
                                dot(IN.texCoord2.xyz, normal.xyz));
    float4 intensity = tex2D(intensityMap, intensCoord);
    float4 color = tex2D(colorMap, IN.texCoord3.xy); 
    return color * intensity;
}

SEE ALSO
vp20

3.5 fp30

NAME
fp30 - OpenGL fragment profile for NV3x (GeForce FX, Quadro FX, etc.)

SYNOPSIS
fp30

DESCRIPTION
This OpenGL profile corresponds to the per-fragment functionality introduced by the GeForce FX and Quadro FX line of NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by NV fragment program.

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES
Requires OpenGL support for the NV fragment program extension. These extensions were introduced by the GeForce FX and Quadro FX GPUs.

OpenGL Extension Specifications
GL_NV_fragment_program

PROFILE OPTIONS

NumInstructionSlots=val
How many instructions the compiler should assume it can use.

NumTemps=val
How many temporaries the compiler should assume it can use.
DATA TYPES

fixed

The fixed data type corresponds to a native signed fixed-point integers with the range \([-2.0,+2.0)\), sometimes called \(fx/2\). This type provides 10 fractional bits of precision.

half

The half data type corresponds to a floating-point encoding with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called \(s10e5\).

float

The float data type corresponds to a standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called \(s10e5\).

SEMANTICS

VARYING INPUT SEMANTICS

The varying input semantics in the fp30 profile correspond to the respectively named varying output semantics of the vp30 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td></td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>
UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Semantic</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Output color (float4)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>DEPTH</td>
<td>Output depth (float)</td>
</tr>
<tr>
<td>DEPR</td>
<td></td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are supported.

SEE ALSO

vp30

3.6 fp40

NAME

fp40 - OpenGL fragment profile for NVIDIA GeForce 6/7 Series, NV4x-based Quadro FX

SYNOPSIS

fp40

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by the GeForce 6800 and other NV4x-based NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by NV_fragment_program2.

Data-dependent loops are allowed with a limit of 256 iterations maximum. Four levels of nesting are allowed.

Conditional expressions can be supported with data-dependent branching.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES

Requires OpenGL support for the NV_fragment_program2 extension. These extensions were introduced by the GeForce 6800 and other NV4x-based GPUs.

OpenGL Extension Specifications

GL_NV_fragment_program2
PROFILE OPTIONS

None.

DATA TYPES

fixed

The fixed data type corresponds to a native signed fixed-point integers with the range [-2.0,+2.0), sometimes called fx12. This type provides 10 fractional bits of precision.

half

The half data type corresponds to a floating-point encoding with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float

The half data type corresponds to a standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

SEMANTICS

VARYING INPUT SEMANTICS

The varying input semantics in the fp40 profile correspond to the respectively named varying output semantics of the vp40 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td></td>
</tr>
</tbody>
</table>
FOGP          Input fog color (XYZ) and factor (W)
FOG
FACE         Polygon facing.
            +1 for front-facing polygon or line or point
            -1 for back-facing polygon

UNIFORM INPUT SEMANTICS
Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>COLOR</th>
<th>Output color (float4)</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>DEPTH</td>
<td>Output depth (float)</td>
</tr>
<tr>
<td>DEPR</td>
<td></td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES
Functions that compute partial derivatives are supported.

SEE ALSO
vp40

3.7 glslf

NAME
gslf - OpenGL fragment profile for the OpenGL Shading Language (GLSL)

SYNOPSIS
glslf

DESCRIPTION
This OpenGL profile corresponds to the per-fragment functionality introduced by the OpenGL Shading Language.
The compiler output for this profile conforms to the language grammar defined by the OpenGL Shading Language specification.

3D API DEPENDENCIES
Requires support for OpenGL 2.0.
PROFILE OPTIONS

version=val

GLSL version to target. Supported versions are 100, 110, 120, 130, 140, 150, 330, 400 and 410.

userTexCoord

Use user-defined varying instead of gl_TexCoord.

ATI_draw_buffers

Use ATI_draw_buffers extension for MRT.

EXT_gpu_shader4

Use EXT_gpu_shader4 extension where useful.

DATA TYPES

The Cg half and fixed data types are both mapped to float because GLSL lacks first-class half and fixed data types.

SEMANTICS

VARYING INPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
<th>GLSL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Primary color (float4)</td>
<td>gl_Color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
<td>gl_SecondaryColor</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXCOORD</td>
<td>Texture coordinate set 0</td>
<td>gl_TexCoord[0]</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
<td>gl_TexCoord[#]</td>
</tr>
<tr>
<td>TEX#</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FACE</td>
<td>Front/back facing (+1/-1)</td>
<td>gl_FrontFacing</td>
</tr>
</tbody>
</table>

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

The following standard fragment output semantics are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
<th>GLSL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Output color (float4)</td>
<td>gl_FragColor</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
COLOR0-COLOR7  Output color (float4)  gl_FragData[n]
COLOR0-COL7    for draw buffers 0 to 7
DEPTH          Output depth (float)  gl_FragDepth
DEPR

SEE ALSO

oglsl, glslg, glslv, cgGLSetContextGLSLVersion

3.8 glslg

NAME

glslg - OpenGL geometry profile for the OpenGL Shading Language (GLSL)

SYNOPSIS

glslg

DESCRIPTION

This OpenGL profile corresponds to the geometry shader functionality introduced by the EXT_geometry_shader4
multi-vendor OpenGL extension.

The compiler output for this profile conforms to the language grammar defined by the OpenGL Shading Language
specification.

3D API DEPENDENCIES

Requires support for OpenGL 2.0 and the EXT_geometry_shader4 extension.

OpenGL Extension Specifications

GL_EXT_geometry_shader4

PROFILE OPTIONS

version=val

GLSL version to target. Supported versions are 100, 110, 120, 130, 140, 150, 330, 400 and 410.

userTexCoord

Use user-defined varying instead of gl_TexCoord.

ATI_draw_buffers

Use ATI_draw_buffers extension for MRT.

EXT_gpu_shader4

Use EXT_gpu_shader4 extension where useful.

DATA TYPES

SEMANTICS

to-be-written
VARYING INPUT SEMANTICS
UNIFORM INPUT SEMANTICS
OUTPUT SEMANTICS
SEE ALSO

gls1, glslv, glslf, cgGLSetContextGLSLVersion

3.9 glsl

NAME
gls1 - OpenGL Shading Language profiles

SYNOPSIS
gls1

DESCRIPTION
gls1 corresponds not to a single profile but to a family of profiles that include glslv, glslg and glslf. For more details refer to each profile documentation.

OpenGL Specifications

http://www.opengl.org/documentation/specs/

GLSL Specifications

http://www.opengl.org/documentation/glsl/
The OpenGL Shading Language Specification v1.10.59
The OpenGL Shading Language Specification v1.20.8

GLSL VERSIONS

Available GLSL versions:

**CG_GL_GLSL_DEFAULT**
The default GLSL version, corresponding to GLSL used for Cg 3.0 and before.

**CG_GL_GLSL_100**
GLSL v1.00 corresponding to ARB_shader_objects extension to OpenGL 1.0. Deprecated by core/forward-compatible OpenGL 3.0, supported by ARB_compatibility

**CG_GL_GLSL_110**
As specified in The OpenGL Shading Language, Language Version 1.10. Supported by OpenGL 2.0 onwards. Deprecated by core/forward-compatible OpenGL 3.0, supported by ARB_compatibility

**CG_GL_GLSL_120**
As specified in The OpenGL Shading Language, Language Version 1.20. Supported by OpenGL 2.1 onwards. Deprecated by core/forward-compatible OpenGL 3.0, supported by ARB_compatibility

STANDARD LIBRARY ISSUES

Cg standard library routines are available.

3.9. glsl
### 3.10 glslv

**NAME**

`glslv` - OpenGL vertex profile for the OpenGL Shading Language (GLSL)

**SYNOPSIS**

`glslv`

**DESCRIPTION**

This OpenGL profile corresponds to the per-vertex functionality introduced by the OpenGL Shading Language.

The compiler output for this profile conforms to the language grammar defined by the OpenGL Shading Language specification.

**3D API DEPENDENCIES**

Requires OpenGL support for **OpenGL 2.0**.

**PROFILE OPTIONS**

- `version=val`
  - GLSL version to target. Supported versions are **100, 110, 120, 130, 140, 150, 330, 400** and **410**.
  - `userTexCoord`
    - Use user-defined varying instead of `gl_TexCoord`.
  - `ATI_draw_buffers`
    - Use `ATI_draw_buffers` extension for MRT.
  - `EXT_gpu_shader4`
    - Use `EXT_gpu_shader4` extension where useful.

**DATA TYPES**

The Cg half and fixed data types are both mapped to float because GLSL lacks first-class half and fixed data types.

**SEMANTICS**

**VARYING INPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
<th>GLSL Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position</td>
<td><code>gl_Vertex</code></td>
</tr>
<tr>
<td>ATTR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
<td><code>gl_Normal</code></td>
</tr>
<tr>
<td>ATTR2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4)</td>
<td><code>gl_Color</code></td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
COLOR1
SPECULAR
ATTR4

FOGOORD
ATTR5

TEXCOORD#
ATTR8
ATTR9
ATTR10
ATTR11
ATTR12
ATTR13
ATTR14
ATTR15

UNIFORM INPUT SEMANTICS

OUTPUT SEMANTICS

Binding Semantics Name   Corresponding Data   GLSL Equivalent
POSITION                  Clip-space position   gl_Position
HPOS                      
COLOR                     Front primary color    gl_FrontColor
COLOR0                    
COLOR0                    
COLOR                     
COL0                      
COL                       

COLOR1                    Front secondary color    gl_FrontSecondaryColor
COL1                      
BCOL0                     Back primary color    gl_BackColor
BCOL1                     Back secondary color    gl_BackSecondaryColor
CLPV                      Clip vertex          gl_ClipVertex
TEXCOORD#                 Texture coordinate set # gl_MultiTexCoord#
TEX#                      
FOGC                      Fog coordinate        gl_FogCoord
FOG                       
PSIZE                     Point size           gl_PointSize
PSIZ                      

STANDARD LIBRARY ISSUES

Vertex texture fetches are supported only if the OpenGL implementation advertises a positive value for the implementation-dependent GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS limit.
SEE ALSO

glsl, glslg, glslf, cgGLSetContextGLSLVersion

3.11 gp4fp

NAME
gp4fp - OpenGL fragment profile for NVIDIA GeForce 8/9/100/200/300 Series, OpenGL 3.x Quadro

SYNOPSIS
gp4fp

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by NVIDIA’s 4th generation of assembly instruction sets.

The compiler output for this profile conforms to the assembly format defined by NV_gpu_program4 and ARB_fragment_program.

Note that the NV_gpu_program4 extension has its fragment domain-specific aspects documented in the NV_fragment_program4 specification.

Data-dependent loops and branching are allowed.

Relative indexing of uniform arrays is supported.

Parameter buffer objects (also known as “constant buffers” in DirectX 10 or “bindable uniform” in GLSL’s EXT_bindable_uniform extension) provide a way to source uniform values from OpenGL buffer objects.

Texture accesses include support for texture arrays (see the EXT_texture_array OpenGL extension for more details) and texture buffer objects (see the EXT_texture_buffer_object extension for details). Texture results can be either conventional floating-point vectors or integer vectors (see the EXT_texture_integer extension for details).

If the system supports the NV_explicit_multisample OpenGL extension, renderbuffers can be bound as textures, which allows texture access to individual samples stored in multi-sample renderbuffer.

3D API DEPENDENCIES

Requires OpenGL support for the NV_gpu_program4 extension. This extension was introduced by the GeForce 8800 and other G8x-based GPUs. Renderbuffer sampler access requires NV_explicit_multisample extension.

OpenGL Extension Specifications

Programmability:

GL_NV_gpu_program4
GL_NV_fragment_program4

New texture samplers:

GL_EXT_texture_array
GL_EXT_texture_buffer_object

New integer texture formats:

GL_EXT_texture_integer

Draw buffers:
GL_ARB_drawBuffers
GL_ATI_drawBuffers
Optional renderbuffer textures:
GL_NV_explicit_multisample

PROFILE OPTIONS

Fragment Domain-specific GP4 Options

ARB_draw_buffers=val
ATI_draw_buffers=val

Indicates that the ARB_draw_buffers or ATI_draw_buffers OpenGL extension is supported and what the extension’s implementation dependent value of GL_MAX_DRAW_BUFFERS_ARB or GL_MAX_DRAW_BUFFERS_ATI is.

When specified, the compiler generates the “OPTION ARB_draw_buffers;” or “OPTION ATI_draw_buffers;” in the compiled code to enable output to multiple draw buffers. Output to multiple draw buffers is done by specifying output parameters with the COLOR1, COLOR2, etc. semantics.

GPUs that support these extensions typically support up to 4 buffers.

These options are useful in the rare situation you want to control the specific OPTION name used. For example, Apple drivers support the ARB_draw_buffers extension but not the ATI_draw_buffers extension.

The CgGL runtime routine cgGLSetOptimalOptions will automatically add the appropriate option based on querying the current OpenGL context’s extension support (prefering the ARB extension) and specify the proper limit.

DATA TYPES

Samplers

This profile has additional samplers for texture arrays (1D and 2D) and texture buffers.

Standard OpenGL textures formats (GL_RGBA8, etc.) return floating-point sampled results, but new signed and unsigned integer texture formats require samplers the return signed and unsigned integer vectors respectively. Sampler variants for fetching signed and unsigned integer vectors are prefixed by i and u respectively. Your application is required to make sure the bound textures have the appropriate texture format. So a 3D texture specified with the GL_RGBA32UI_EXT internal format (see the EXT_texture_integer OpenGL extension) must be used with a usampler3D sampler. Otherwise, texture sampling returns undefined results.

sampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns float vectors.
isampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns int vectors.
usampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns unsigned int vectors.
sampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.
isampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.

usampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

tsampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns float vectors.
isampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns int vectors.
usampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns unsigned int vectors.

tsampler2DARRAY
2D array texture unit corresponding to OpenGL’s GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.
isampler2DARRAY
2D array texture unit corresponding to OpenGL’s GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.
usampler2DARRAY
2D array texture unit corresponding to OpenGL’s GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

tsampler3D
3D texture unit corresponding to OpenGL’s GL_TEXTURE_3D target. Sampling returns float vectors.
isampler3D
3D texture unit corresponding to OpenGL’s GL_TEXTURE_3D target. Sampling returns int vectors.
usampler3D
3D texture unit corresponding to OpenGL’s GL_TEXTURE_3D target. Sampling returns unsigned int vectors.

tsamplerCUBE
Cube map texture unit corresponding to OpenGL’s GL_TEXTURE_CUBE_MAP target. Sampling returns float vectors.
isamplerCUBE
Cube map texture unit corresponding to OpenGL’s GL_TEXTURE_CUBE_MAP target. Sampling returns int vectors.
usamplerCUBE
Cube map texture unit corresponding to OpenGL’s GL_TEXTURE_CUBE_MAP target. Sampling returns unsigned int vectors.

tsamplerRECT
Rectangle texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RECTANGLE\_ARB} target. Sampling returns \texttt{float} vectors.

\texttt{isamplerRECT}

Rectangle texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RECTANGLE\_ARB} target. Sampling returns \texttt{int} vectors.

\texttt{isamplerRECT}

Rectangle texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RECTANGLE\_ARB} target. Sampling returns \texttt{unsigned int} vectors.

\texttt{samplerBUF}

Buffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_BUFFER\_EXT} target provided by the \texttt{EXT\_texture\_buffer\_object} extension. Sampling returns \texttt{float} vectors.

\texttt{isamplerBUF}

Buffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_BUFFER\_EXT} target provided by the \texttt{EXT\_texture\_buffer\_object} extension. Sampling returns \texttt{int} vectors.

\texttt{usamplerBUF}

Buffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_BUFFER\_EXT} target provided by the \texttt{EXT\_texture\_buffer\_object} extension. Sampling returns \texttt{unsigned int} vectors.

\texttt{samplerRBUF}

Renderbuffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RENDERBUFFER\_NV} target provided by the \texttt{NV\_explicit\_multisample} extension. Sampling returns \texttt{float} vectors.

\texttt{isamplerRBUF}

Renderbuffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RENDERBUFFER\_NV} target provided by the \texttt{NV\_explicit\_multisample} extension. Sampling returns \texttt{int} vectors.

\texttt{usamplerRBUF}

Renderbuffer texture unit corresponding to OpenGL’s \texttt{GL\_TEXTURE\_RENDERBUFFER\_NV} target provided by the \texttt{NV\_explicit\_multisample} extension. Sampling returns \texttt{unsigned int} vectors.

\textbf{Floating-point}

\texttt{float}

32-bit IEEE floating-point

\texttt{half}

32-bit IEEE floating-point

\texttt{double}

32-bit IEEE floating-point

\texttt{fixed}

Floating-point restricted to \([-2,2)\) range.

\textbf{Integer}

This profile supports “true” integer data types. Shifting and bitwise operators are supported for integer data types.
int
32-bit signed integer
unsigned int
32-bit unsigned integer
short
16-bit signed integer
unsigned short
16-bit unsigned integer
char
8-bit signed integer
unsigned char
8-bit unsigned integer

SEMANTICS

VARYING INPUT SEMANTICS

Interpolated Input Semantics
The varying input semantics in the \texttt{gp4fp} profile correspond to the respectively named varying output semantics of the \texttt{gp4vp} profile (or \texttt{gp4gp} if a geometry shader is present).

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>WP00S</td>
<td>Window position (with lower-left origin)</td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
</tbody>
</table>
TEX6
TEXCOORD6

Input texture coordinate sets 6

TEX7
TEXCOORD7

Input texture coordinate sets 7

FOGP
FOG

Input fog color (XYZ) and factor (W)

Interpolation Semantic Modifiers

A number of interpolation semantic modifiers control how interpolation happens for the interpolated input semantics above. These modifiers are suffixed to the semantic name with a "." (period) separator. Without a modifier, perspective-correct interpolation applies.

<table>
<thead>
<tr>
<th>Semantic Modifier Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CENTROID</td>
<td>Interpolate at the centroid of the covered samples (only applies when rendering to multisampled surface)</td>
</tr>
<tr>
<td>FLAT</td>
<td>Flat shading using provoking vertex’s value</td>
</tr>
<tr>
<td>NOPERSPECTIVE</td>
<td>Interpolate without perspective correction</td>
</tr>
</tbody>
</table>

Examples:

float4 a : TEXCOORD0
float4 b : TEXCOORD1.CENTROID
float4 c : TEXCOORD2.FLAT
float4 d : TEXCOORD3.NOPERSPECTIVE

Per-primitive Input Semantics

FACE

Polygon facing.
+1 for front-facing polygon or line or point
-1 for back-facing polygon

PRIMITIVEID

Primitive ID (int)

If a geometry program is active, parameters given the PRIMITIVEID semantic obtained their integer value from the primitive ID value emitted by the geometry program for the provoking vertex. If no geometry program is active, the value is the number of primitives processed by the rasterizer since the last time glBegin was called (directly or indirectly via vertex array functions). The first primitive generated after a glBegin is numbered zero, and the primitive ID counter is incremented after every individual point, line, or polygon primitive is processed. For polygons drawn in point or line mode, the primitive ID counter is incremented only once, even though multiple points or lines may be drawn. For QUADS and QUAD_STRIP primitives that are decomposed into triangles, the primitive ID is incremented after each complete quad is processed. For POLYGON primitives, the primitive ID counter is zero. The primitive ID is zero for fragments generated by DrawPixels or Bitmap. Restarting a primitive topology using the primitive restart index has no effect on the primitive ID counter.

UNIFORM INPUT SEMANTICS

Buffer Semantics

gp4 profiles can specify that uniforms be specified to reside within binable buffers.

Example of automatic, compiler-determined specification of a uniform’s location within a buffer:
uniform float2 location : BUFFER[3];  // compiler positions within buffer 3
uniform float4 brickColor : BUFFER[3];  // compiler positions within buffer 3

Example of absolute byte offset specification of a uniform’s location within a buffer:
uniform float4 mustBeHere : BUFFER[7][20];  // locate 20 bytes into buffer 7

Constant Register Semantics

C0–C255    Constant register [0..255].
            The aliases c0–c255 (lowercase) are also accepted.

If used with a variable that requires more than one constant register (e.g. a matrix), the semantic specifies the first register that is used.
Example:
uniform float4 array[20] : C14;  // uses c14 through c33

Texture Unit Semantics

TEXUNIT0–TEXUNIT31  Texture image unit

Example:
uniform sampler2DARRAY texArray : TEXUNIT7;

OUTPUT SEMANTICS

COLOR         Output color (float4 or int4)
COL
COLOR0–COLOR7 Output color (float4 or int4) for draw buffers 0 to 7
COL0–COL7

DEPTH         Output depth (float)
DEPR

STANDARD LIBRARY ISSUES

Raw Cast from Floating-point to Integer Functions

It is possible to convert the raw bit patterns of IEEE single-precision floating-point to 32-bit unsigned integer.
floatToRawIntBits, floatToIntBits, intBitsToFloat

Texture Array Functions

New sampler data types for texture arrays and texture buffers lead to new standard library routines to access these samplers.

New standard library functions are used to access 1D texture array samplers (sampler1DARRAY).
tex1DARRAY, tex1DARRAYbias, tex1DARRAYcmpbias, tex1DARRAYlod, tex1DARRAYcmplod, tex1DARRAYproj

The dimensions of a texture array level can be determined.
tex1DARRAYsize

New standard library functions are used to access 2D texture array samplers (sampler2DARRAY).
tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYproj
The dimensions of a texture array level can be determined.

\[ \text{tex2DARRAYsize} \]

SEE ALSO

gp4, gp4vp, gp4gp, texBUF, texBUFsize, texRBUF, texRBUFsize, floatToIntBits, floatToRawIntBits, intBitsToFloat, tex1DARRAY, tex1DARRAYbias, tex1DARRAYcmpbias, tex1DARRAYlod, tex1DARRAYcmlod, tex1DARRAYproj, tex1DARRAYsize, tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYcmlod, tex2DARRAYproj, tex2DARRAYsize

### 3.12 gp4gp

#### NAME

gp4gp - OpenGL geometry profile for NVIDIA GeForce 8/9/100/200/300 Series, OpenGL 3.x Quadro

#### SYNOPSIS

gp4gp

#### DESCRIPTION

This OpenGL profile corresponds to the per-primitive functionality introduced by NVIDIA's 4th generation of assembly instruction sets.

The compiler output for this profile conforms to the assembly format defined by \texttt{NV\_gpu\_program4} and \texttt{ARB\_vertex\_program}.

Note that the \texttt{NV\_gpu\_program4} extension has its geometry domain-specific aspects documented in the \texttt{NV\_geometry\_program4} specification.

Data-dependent loops and branching are allowed.

Relative indexing of uniform arrays is supported.

Parameter buffer objects (also known as “constant buffers” in DirectX 10 or “bindable uniform” in GLSL’s \texttt{EXT\_bindable\_uniform} extension) provide a way to source uniform values from OpenGL buffer objects.

Texture accesses include support for texture arrays (see the \texttt{EXT\_texture\_array} OpenGL extension for more details) and texture buffer objects (see the \texttt{EXT\_texture\_buffer\_object} extension for details). Texture results can be either conventional floating-point vectors or integer vectors (see the \texttt{EXT\_texture\_integer} extension for details).

If the system supports the \texttt{NV\_explicit\_multisample} OpenGL extension, renderbuffers can be bound as textures, which allows texture access to individual samples stored in multi-sample renderbuffer.

#### 3D API DEPENDENCIES

Requires OpenGL support for the \texttt{NV\_gpu\_program4} extension. This extension was introduced by the GeForce 8800 and other G8x-based GPUs. Renderbuffer sampler access requires \texttt{NV\_explicit\_multisample} extension.

#### OpenGL Extension Specifications

Programmability:

\[ \text{GL\_NV\_gpu\_program4} \]
\[ \text{GL\_NV\_geometry\_program4} \]

New texture samplers:

\[ \text{GL\_EXT\_texture\_array} \]
GL_EXT_texture_buffer_object

New integer texture formats:
GL_EXT_texture_integer

Optional renderbuffer textures:
GL_NV_explicit_multisample

PROFILE OPTIONS

Geometry Domain-specific GP4 Options

Normally the options below to specify primitive input and output type are better specified as a profile modifier preceding the Cg entry function.

Vertices=val

Maximum number of output vertices. Emitting more than this number of vertices results in the extra vertices being discarded. Example “-po Vertices=3”

On NVIDIA GPUs, the throughput for geometry shaders is inverse proportional to the maximum number of vertices output times the number of scalar components per vertex. For this reason, keep the maximum number of output vertices as small as possible for best performance.

POINT

The entry function inputs point primitives. Example: “-po POINT”

If an output primitive is not also specified with a command line profile option, POINT_OUT is assumed.

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

LINE

The entry function inputs line primitives. Example: “-po LINE”

If an output primitive is not also specified with a command line profile option, LINE_OUT is assumed.

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

LINE_ADJ

The entry function inputs line adjacency primitives. Example: “-po LINE_ADJ”

If an output primitive is not also specified with a command line profile option, LINE_OUT is assumed.

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

TRIANGLE

The entry function inputs triangle primitives. Example: “-po TRIANGLE”

If an output primitive is not also specified with a command line profile option, TRIANGLE_OUT is assumed.

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

TRIANGLE_ADJ

The entry function inputs triangle adjacency primitives. Example: “-po TRIANGLE_ADJ”

If an output primitive is not also specified with a command line profile option, TRIANGLE_OUT is assumed.

POINT_OUT

The entry function outputs point primitives. Example: “-po POINT_OUT”
Normally this is option is better specified as a profile modifier preceding the Cg entry function.

LINE_OUT
The entry function outputs line primitives. Example: “-po LINE_OUT”

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

TRIANGLE_OUT
The entry function outputs triangle primitives. Example: “-po TRIANGLE_OUT”

Normally this is option is better specified as a profile modifier preceding the Cg entry function.

DATA TYPES

Samplers

This profile has additional samplers for texture arrays (1D and 2D) and texture buffers.

Standard OpenGL textures formats (GL_RGBA8, etc.) return floating-point sampled results, but new signed and unsigned integer texture formats require samplers the return signed and unsigned integer vectors respectively. Sampler variants for fetching signed and unsigned integer vectors are prefixed by i and u respectively. Your application is required to make sure the bound textures have the appropriate texture format. So a 3D texture specified with the GL_RGBA32UI_EXT internal format (see the EXT_texture_integer OpenGL extension) must be used with a sampler3D sampler. Otherwise, texture sampling returns undefined results.

sampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns float vectors.

isampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns int vectors.

usampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns unsigned int vectors.

sampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.

isampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.

usampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

sampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns float vectors.

isampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns int vectors.

usampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns unsigned int vectors.
sampler2DARRAY
2D array texture unit corresponding to OpenGL's GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.

isampler2DARRAY
2D array texture unit corresponding to OpenGL's GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.

usampler2DARRAY
2D array texture unit corresponding to OpenGL's GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

sampler3D
3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns float vectors.

isampler3D
3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns int vectors.

usampler3D
3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns unsigned int vectors.

samplerCUBE
Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns float vectors.

isamplerCUBE
Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns int vectors.

usamplerCUBE
Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns unsigned int vectors.

samplerRECT
Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns float vectors.

isamplerRECT
Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns int vectors.

usamplerRECT
Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns unsigned int vectors.

samplerBUF
Buffer texture unit corresponding to OpenGL’s GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns float vectors.

isamplerBUF
Buffer texture unit corresponding to OpenGL’s GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns int vectors.
usamplerBUF

Buffer texture unit corresponding to OpenGL’s GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns unsigned int vectors.

samplerRBUF

Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns float vectors.

isamplerRBUF

Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns int vectors.

usamplerRBUF

Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns unsigned int vectors.

Floating-point

float

32-bit IEEE floating-point

half

32-bit IEEE floating-point
double

32-bit IEEE floating-point
fixed

Floating-point restricted to [-2,2) range.

Integer

This profile supports “true” integer data types. Shifting and bitwise operators are supported for integer data types.

int

32-bit signed integer
unsigned int

32-bit unsigned integer
short

16-bit signed integer
unsigned short

16-bit unsigned integer
char

8-bit signed integer
unsigned char

8-bit unsigned integer
SEMANTICS

VARYING INPUT SEMANTICS

Primitive Instance Input Semantic

Within a batch of primitives (defined in OpenGL by a glBegin/glEnd sequence or an implied glBegin/glEnd performed by glDrawElements, glDrawArrays, etc.) a counter tracks each assembled primitive instance. The geometry shader has access to this counter through the INSTANCEID semantic.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSTANCEID</td>
<td>Integer instance ID of the primitive within the current batch</td>
</tr>
</tbody>
</table>

The first primitive generated after a glBegin is numbered zero, and the instance ID counter is incremented after every individual point, line, or polygon primitive is processed. For QUADS and QUAD_STRIP primitives that are decomposed into triangles, the instance ID is incremented after each complete quad is processed. For POLYGON primitives, the instance ID counter is zero. Restarting a primitive topology using the primitive restart index has no effect on the instance ID counter.

Example:

```
int primID : INSTANCEID
```

Vertex Instance Input Semantic

The geometry shader can identify the vertex index (when using vertex buffer objects) that sourced each vertex making up the primitive.

The vertex instance input semantic must be declared with the `AttribArray` template-style syntax because a geometry shader accepts an attribute array of vertex instance IDs.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>VERTEXID</td>
<td>Integer ID of the vertex’s index for vertex pulling</td>
</tr>
</tbody>
</table>

The vertex ID is equal to value effectively passed to glArrayElement (or routines that implicitly call glArrayElements such as glDrawElements or glDrawArrays) when the vertex is specified, and is defined only if vertex arrays are used with buffer objects (VBOs).

Example defining a varying parameter for the position vertex attribute:

```
AttribArray<int> vertexID : VERTEXID
```

Vertex Attribute Input Semantics

For geometry shader profiles such as gp4gp, varying parameters with vertex attribute input semantics must be declared with the `AttribArray` template-style syntax because a geometry shader accepts an attribute array of vertex attributes (rather than individual vertex attributes as a vertex shader does).

Example defining a varying parameter for the position vertex attribute:

```
AttribArray<float4> position : POSITION
```

The set of binding semantics for varying input vertex attributes to gp4gp consists of POSITION, BLENDWEIGHT, NORMAL, COLOR0, COLOR1, TESSFACTOR, PSIZE, BLENDINDICES, and TEXCOORD0-TEXCOORD7. One can also use TANGENT and BINORMAL instead of TEXCOORD6 and TEXCOORD7.

Additionally, a set of binding semantics ATTR0-ATTR15 can be used. These binding semantics map to NV_gpu_program4 input attribute parameters as described in the NV_geometry_program4 document.
The two sets act as aliases to each other on NVIDIA GPUs excluding Apple Macs. ATI GPUs and NVIDIA Mac GPUs do not alias the conventional vertex attributes with the generic attributes.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, ATTR0</td>
<td>Input Vertex, Generic Attribute 0</td>
</tr>
<tr>
<td>BLENDWEIGHT, ATTR1</td>
<td>Input vertex weight, Generic Attribute 1</td>
</tr>
<tr>
<td>NORMAL, ATTR2</td>
<td>Input normal, Generic Attribute 2</td>
</tr>
<tr>
<td>DIFFUSE, COLOR0, ATTR3</td>
<td>Input primary color, Generic Attribute 3</td>
</tr>
<tr>
<td>SPECULAR, COLOR1, ATTR4</td>
<td>Input secondary color, Generic Attribute 4</td>
</tr>
<tr>
<td>TESSFACTOR, FOGCOORD, ATTR5</td>
<td>Input fog coordinate, Generic Attribute 5</td>
</tr>
<tr>
<td>PSIZE, ATTR6</td>
<td>Input point size, Generic Attribute 6</td>
</tr>
<tr>
<td>BLENDINDICES, ATTR7</td>
<td>Generic Attribute 7</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7, ATTR8-ATTR15</td>
<td>Input texture coordinates (texcoord0-texcoord7)</td>
</tr>
<tr>
<td>TANGENT, ATTR14</td>
<td>Generic Attribute 14</td>
</tr>
<tr>
<td>BINORMAL, ATTR15</td>
<td>Generic Attribute 15</td>
</tr>
<tr>
<td>VERTEXID</td>
<td>Input vertex ID (integer)</td>
</tr>
<tr>
<td>PRIMITIVEID</td>
<td>Input primitive ID (integer)</td>
</tr>
</tbody>
</table>

These vertex attribute semantics should match with the semantics of the outputs of the vertex shader feeding the geometry shader.

**UNIFORM INPUT SEMANTICS**

**Buffer Semantics**

gp4 profiles can specify that uniforms be specified to reside within binable buffers.

Example of automatic, compiler-determined specification of a uniform’s location within a buffer:

```cpp
uniform float2 location : BUFFER[3]; // compiler positions within buffer 3
uniform float4 brickColor : BUFFER[3]; // compiler positions within buffer 3
```

Example of absolute byte offset specification of a uniform’s location within a buffer:

```cpp
uniform float4 mustBeHere : BUFFER[7][20]; // locate 20 bytes into buffer 7
```

**Constant Register Semantics**

C0-C255                  Constant **register [0..255]**.
The aliases c0-c255 (lowercase) are also accepted.

If used with a variable that requires more than one constant register (e.g. a matrix), the semantic specifies the first register that is used.

Example:
uniform float4 array[20] : C14; // uses c14 through c33

Texture Unit Semantics

TEXUNIT0-TEXUNIT31  Texture image unit

Example:

uniform sampler2DARRAY texArray : TEXUNIT7;

OUTPUT SEMANTICS

These vertex attribute output semantics match the output semantics for gp4vp.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7,</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>TEX0-TEX7</td>
<td></td>
</tr>
<tr>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
</tr>
<tr>
<td>PRIMITIVEID</td>
<td>Output primitive ID (integer)</td>
</tr>
<tr>
<td>LAYER</td>
<td>Output layer (integer)</td>
</tr>
<tr>
<td></td>
<td>Output 0 through 5 for cube map faces</td>
</tr>
<tr>
<td></td>
<td>Output 0 through N for 2D texture arrays</td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

Raw Cast from Floating-point to Integer Functions

It is possible to convert the raw bit patterns of IEEE single-precision floating-point to 32-bit unsigned integer.

\text{floatToRawIntBits}, \text{floatToIntBits}, \text{intBitsToFloat}

Texture Array Functions

New sampler data types for texture arrays and texture buffers lead to new standard library routines to access these samplers.

New standard library functions are used to access 1D texture array samplers (sampler1DARRAY).

\text{tex1DARRAY}, \text{tex1DARRAYbias}, \text{tex1DARRAYcmpbias}, \text{tex1DARRAYlod}, \text{tex1DARRAYcmplod}, \text{tex1DARRAYproj}

The dimensions of a texture array level can be determined.

\text{tex1DARRAYsize}
New standard library functions are used to access 2D texture array samplers (sampler2DARRAY).

tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYproj

The dimensions of a texture array level can be determined.

tex2DARRAYsize

SEE ALSO

gp4, gp4vp, gp4fp, texBUF, texBUFSIZE, texRBUF, texRBUFSIZE, floatToRawIntBits, floatToIntBits, intBitsToFloat, tex1DARRAY, tex1DARRAYbias, tex1DARRAYcmpbias, tex1DARRAYlod, tex1DARRAYcmplod, tex1DARRAYproj, tex1DARRAYsize, tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYproj, tex2DARRAYsize

3.13 gp4

NAME

gp4 - OpenGL profiles for NVIDIA GeForce 8/9/100/200/300 Series, OpenGL 3.x Quadro

SYNOPSIS

gp4

DESCRIPTION

gp4 corresponds not to a single profile but to a family of profiles that include gp4vp, gp4gp and gp4fp. Since most of the functionality exposed in these profiles is almost identical and defined by NV_gpu_program4 these profiles are named the gp4 profiles. For more details refer to each profile documentation.

OpenGL Extension Specifications

GL_NV_gpu_program4

PROFILE OPTIONS

Common GP4 Options

fastimul

Assume integer multiply inputs have at most 24 significant bits.

Disabled by default.

Example: “-po fastimul”

PaBO2=val

NV_parameter_buffer_object2=val

Use the NV_parameter_buffer_object2 (PaBO2) extension.

By default the NV_parameter_buffer_object (PaBO) extension is used for buffer layout, implied by the NV_gpu_program4 extension.

Example: “-po NV_parameter_buffer_object2=1”

GL_NV_parameter_buffer_object

GL_NV_parameter_buffer_object2
SEE ALSO

gp4vp, gp4gp, gp4fp

3.14 gp4vp

NAME

gp4vp - OpenGL vertex profile for NVIDIA GeForce 8/9/100/200/300 Series, OpenGL 3.x Quadro

SYNOPSIS

gp4vp

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by NVIDIA’s 4th generation of assembly instruction sets.

The compiler output for this profile conforms to the assembly format defined by NV_gpu_program4 and ARB_vertex_program.

Note that the NV_gpu_program4 extension has its vertex domain-specific aspects documented in the NV_vertex_program4 specification.

Data-dependent loops and branching are allowed.

Relative indexing of uniform arrays is supported.

Texture accesses are supported. While the prior vp40 profile has substantial limitations on vertex texturing, the gp4vp profile eliminates all the limitations.

Texture accesses include support for texture arrays (see the EXT_texture_array OpenGL extension for more details) and texture buffer objects (see the EXT_texture_buffer_object extension for details). Texture results can be either conventional floating-point vectors or integer vectors (see the EXT_texture_integer extension for details).

If the system supports the NV_explicit_multisample OpenGL extension, renderbuffers can be bound as textures, which allows texture access to individual samples stored in multi-sample renderbuffer.

3D API DEPENDENCIES

Requires OpenGL support for the NV_gpu_program4 extension. This extension was introduced by the GeForce 8800 and other G8x-based GPUs. Renderbuffer sampler access requires NV_explicit_multisample extension.

OpenGL Extension Specifications

Programmability:

GL_NV_gpu_program4
GL_NV_vertex_program4

New texture samplers:

GL_EXT_texture_array
GL_EXT_texture_buffer_object

New integer texture formats:

GL_EXT_texture_integer

Optional renderbuffer textures:
GL_NV_explicit_multisample

PROFILE OPTIONS

DATA TYPES

Samplers

This profile has additional samplers for texture arrays (1D and 2D) and texture buffers.

Standard OpenGL textures formats (GL_RGBA8, etc.) return floating-point sampled results, but new signed and unsigned integer texture formats require samplers the return signed and unsigned integer vectors respectively. Sampler variants for fetching signed and unsigned integer vectors are prefixed by i and u respectively. Your application is required to make sure the bound textures have the appropriate texture format. So a 3D texture specified with the GL_RGBA32UI_EXT internal format (see the EXT_texture_integer OpenGL extension) must be used with a usampler3D sampler. Otherwise, texture sampling returns undefined results.

sampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns float vectors.

isampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns int vectors.

usampler1D
1D texture unit corresponding to OpenGL’s GL_TEXTURE_1D target. Sampling returns unsigned int vectors.

sampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.

isampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.

usampler1DARRAY
1D array texture unit corresponding to OpenGL’s GL_TEXTURE_1D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

sampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns float vectors.

isampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns int vectors.

usampler2D
2D texture unit corresponding to OpenGL’s GL_TEXTURE_2D target. Sampling returns unsigned int vectors.

sampler2DARRAY
2D array texture unit corresponding to OpenGL’s GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns float vectors.

isampler2DARRAY
2D array texture unit corresponding to OpenGL’s GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns int vectors.
usampler2DARRAY

2D array texture unit corresponding to OpenGL's GL_TEXTURE_2D_ARRAY_EXT target provided by the EXT_texture_array extension. Sampling returns unsigned int vectors.

sampler3D

3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns float vectors.

isampler3D

3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns int vectors.

usampler3D

3D texture unit corresponding to OpenGL's GL_TEXTURE_3D target. Sampling returns unsigned int vectors.

samplerCUBE

Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns float vectors.

isamplerCUBE

Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns int vectors.

usamplerCUBE

Cube map texture unit corresponding to OpenGL's GL_TEXTURE_CUBE_MAP target. Sampling returns unsigned int vectors.

samplerRECT

Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns float vectors.

isamplerRECT

Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns int vectors.

isamplerRECT

Rectangle texture unit corresponding to OpenGL's GL_TEXTURE_RECTANGLE_ARB target. Sampling returns unsigned int vectors.

samplerBUF

Buffer texture unit corresponding to OpenGL's GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns float vectors.

isamplerBUF

Buffer texture unit corresponding to OpenGL's GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns int vectors.

usamplerBUF

Buffer texture unit corresponding to OpenGL's GL_TEXTURE_BUFFER_EXT target provided by the EXT_texture_buffer_object extension. Sampling returns unsigned int vectors.

samplerRBUF

Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns float vectors.
Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns int vectors.

Renderbuffer texture unit corresponding to OpenGL’s GL_TEXTURE_RENDERBUFFER_NV target provided by the NV_explicit_multisample extension. Sampling returns unsigned int vectors.

**Floating-point**

- float
- 32-bit IEEE floating-point
- half
- 32-bit IEEE floating-point
- double
- 32-bit IEEE floating-point
- fixed

Floating-point restricted to [-2,2) range.

**Integer**

This profile supports “true” integer data types. Shifting and bitwise operators are supported for integer data types.

- int
- 32-bit signed integer
- unsigned int
- 32-bit unsigned integer
- short
- 16-bit signed integer
- unsigned short
- 16-bit unsigned integer
- char
- 8-bit signed integer
- unsigned char
- 8-bit unsigned integer

**SEMANTICS**

**VARYING INPUT SEMANTICS**

**Vertex Attribute Input Semantics**

For geometry shader profiles such as gp4gp, varying parameters with vertex attribute input semantics must be declared with the AttribArray template-style syntax because a geometry shader accepts an attribute array of vertex attributes (rather than individual vertex attributes as a vertex shader does).

Example defining a varying parameter for the position vertex attribute:
AttribArray<float4> position : POSITION

The set of binding semantics for varying input vertex attributes to gp4gp consists of POSITION, BLENDWEIGHT, NORMAL, COLOR0, COLOR1, TESSFACTOR, PSIZE, BLENDINDICES, and TEXCOORD0-TEXCOORD7. One can also use TANGENT and BINORMAL instead of TEXCOORD6 and TEXCOORD7.

Additionally, a set of binding semantics ATTR0-ATTR15 can be used. These binding semantics map to NV_gpu_program4 input attribute parameters as described in the NVGeometryProgram4 document.

The two sets act as aliases to each other on NVIDIA GPUs excluding Apple Macs. ATI GPUs and NVIDIA Mac GPUs do not alias the conventional vertex attributes with the generic attributes.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, ATTR0</td>
<td>Input Vertex, Generic Attribute 0</td>
</tr>
<tr>
<td>BLENDWEIGHT, ATTR1</td>
<td>Input vertex weight, Generic Attribute 1</td>
</tr>
<tr>
<td>NORMAL, ATTR2</td>
<td>Input normal, Generic Attribute 2</td>
</tr>
<tr>
<td>DIFFUSE, COLOR0, ATTR3</td>
<td>Input primary color, Generic Attribute 3</td>
</tr>
<tr>
<td>SPECULAR, COLOR1, ATTR4</td>
<td>Input secondary color, Generic Attribute 4</td>
</tr>
<tr>
<td>TESSFACTOR, FOGCOORD, ATTR5</td>
<td>Input fog coordinate, Generic Attribute 5</td>
</tr>
<tr>
<td>PSIZE, ATTR6</td>
<td>Input point size, Generic Attribute 6</td>
</tr>
<tr>
<td>BLENDINDICES, ATTR7</td>
<td>Generic Attribute 7</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7, ATTR8-ATTR15</td>
<td>Input texture coordinates (texcoord0-texcoord7)</td>
</tr>
<tr>
<td>TANGENT, ATTR14</td>
<td>Generic Attribute 14</td>
</tr>
<tr>
<td>BINORMAL, ATTR15</td>
<td>Generic Attribute 15</td>
</tr>
<tr>
<td>VERTEXID</td>
<td>Input vertex ID (integer)</td>
</tr>
<tr>
<td></td>
<td>Vertex index when using vertex arrays</td>
</tr>
<tr>
<td>INSTANCEID</td>
<td>Integer instance ID of the primitive within the current batch</td>
</tr>
</tbody>
</table>

These vertex attribute semantics should match with the semantics of the outputs of the vertex shader feeding the geometry shader.

**UNIFORM INPUT SEMANTICS**

**Buffer Semantics**

gp4 profiles can specify that uniforms be specified to reside within binable buffers.

Example of automatic, compiler-determined specification of a uniform’s location within a buffer:

```c
uniform float2 location : BUFFER[3];  // compiler positions within buffer 3
uniform float4 brickColor : BUFFER[3];  // compiler positions within buffer 3
```

Example of absolute byte offset specification of a uniform’s location within a buffer:
uniform float4 mustBeHere : BUFFER[7][20];  // locate 20 bytes into buffer 7

**Constant Register Semantics**

C0-C255  Constant register [0..255].
The aliases c0-c255 (lowercase) are also accepted.

If used with a variable that requires more than one constant register (e.g. a matrix), the semantic specifies the first register that is used.

Example:

uniform float4 array[20] : C14;  // uses c14 through c33

**Texture Unit Semantics**

TEXUNIT0-TEXUNIT31  Texture image unit

Example:

uniform sampler2DARRAY texArray : TEXUNIT7;

**OUTPUT SEMANTICS**

These vertex attribute output semantics match the output semantics for gp4vp.

<table>
<thead>
<tr>
<th>Binding Semantics</th>
<th>Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td></td>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td></td>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td></td>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td></td>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td></td>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td></td>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td></td>
<td>TEXCOORD0-TEXCOORD7, TEX0-TEX7</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td></td>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

**Raw Cast from Floating-point to Integer Functions**

It is possible to convert the raw bit patterns of IEEE single-precision floating-point to 32-bit unsigned integer.

*floatToRawIntBits, floatToIntBits, intBitsToFloat*

**Texture Array Functions**

New sampler data types for texture arrays and texture buffers lead to new standard library routines to access these samplers.

New standard library functions are used to access 1D texture array samplers (sampler1DARRAY).
The dimensions of a texture array level can be determined.

tex1DARRAYsize

New standard library functions are used to access 2D texture array samplers (sampler2DARRAY).

tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYproj

The dimensions of a texture array level can be determined.

tex2DARRAYsize

SEE ALSO
gp4, gp4gp, gp4fp, texBUF, texBUFSIZE, texRBUF, texRBUFsize, floatToRawIntBits, floatToIntBits, intBitsToFloat, tex1DARRAY, tex1DARRAYbias, tex1DARRAYcmpbias, tex1DARRAYlod, tex1DARRAYcmplod, tex1DARRAYproj, tex1DARRAYsize, tex2DARRAY, tex2DARRAYbias, tex2DARRAYlod, tex2DARRAYproj, tex2DARRAYsize

3.15 gp5fp

NAME
gp5fp - OpenGL fragment profile for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro

SYNOPSIS
gp5fp

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by NVIDIA’s 5th generation of assembly instruction sets. It extends gp4fp.

The compiler output for this profile conforms to the assembly format defined by NV_gpu_program5 and ARB_fragment_program.

3D API DEPENDENCIES

Requires OpenGL support for the NV_gpu_program5 extension.

OpenGL Extension Specifications

GL_NV_gpu_program5

PROFILE OPTIONS

All gp4fp profile options are also available in gp5fp.

SEE ALSO
gp4fp, gp5, gp5tcp, gp5tep, gp5vp, gp5gp

3.16 gp5gp

NAME
gp5gp - OpenGL geometry profile for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro
SYNOPSIS

gp5gp

DESCRIPTION

This OpenGL profile corresponds to the per-primitive functionality introduced by NVIDIA’s 5th generation of assembly instruction sets. It extends gp4gp.

Tessellation patch input primitives are supported.

Instanced invocations are supported.

The compiler output for this profile conforms to the assembly format defined by NV_gpu_program5 and ARB_vertex_program.

Note that the NV_gpu_program5 extension has its geometry domain-specific aspects documented in the NV_geometry_program4 specification.

3D API DEPENDENCIES

Requires OpenGL support for the NV_gpu_program5 extension.

OpenGL Extension Specifications

GL_NV_gpu_program5
GL_NV_geometry_program4

PROFILE OPTIONS

All gp4gp profile options are also available in gp5gp.

Invocations=val

The number of instanced invocations. Example: “Invocations=2”

PATCH_1 .. PATCH_32

Patch input primitive of size 1 to 32. Example: “PATCH_16”

SEE ALSO

gp4gp, gp5, gp5tcp, gp5tep, gp5vp, gp5fp

3.17 gp5

NAME

gp5 - OpenGL profiles for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro

SYNOPSIS

gp5

DESCRIPTION

gp5 corresponds not to a single profile but to a family of profiles that include gp5vp, gp5tcp, gp5tep, gp5gp and gp5fp. Since most of the functionality exposed in these profiles is almost identical and defined by NV_gpu_program5 these profiles are named the gp5 profiles. For more details refer to each profile documentation.
OpenGL Extension Specifications

GL_NV_gpu_program5

PROFILE OPTIONS

All \textit{gp4} profile options are also available in \textit{gp5}.

\begin{itemize}
  \item \texttt{PaBO2=\textit{val}}
  \item \texttt{NV\_parameter\_buffer\_object2=\textit{val}}
\end{itemize}

Use the \texttt{NV\_parameter\_buffer\_object2 (PaBO2)} extension.

By default the \texttt{PaBO2} extension is used for \textit{gp5} buffer layout. Alternatively the \texttt{NV\_parameter\_buffer\_object (PaBO)} buffer layout can be specified: \texttt{“-po NV\_parameter\_buffer\_object2=0”}

\begin{itemize}
  \item GL_NV\_parameter\_buffer\_object
  \item GL_NV\_parameter\_buffer\_object2
\end{itemize}

SEE ALSO

\textit{gp5tcp}, \textit{gp5tep}, \textit{gp5vp}, \textit{gp5gp}, \textit{gp5fp}

3.18 \textit{gp5tcp}

NAME

\textit{gp5tcp} - OpenGL tessellation control profile for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro

SYNOPSIS

\textit{gp5tcp}

DESCRIPTION

This OpenGL profile corresponds to tessellation control functionality introduced by NVIDIA’s 5th generation of assembly instruction sets.

The compiler output for this profile conforms to the assembly format defined by \texttt{NV\_tessellation\_program}, \texttt{NV\_gpu\_program5} and \texttt{ARB\_vertex\_program}.

3D API DEPENDENCIES

Requires OpenGL support for the \texttt{NV\_gpu\_program5} extension.

OpenGL Extension Specifications

\begin{itemize}
  \item GL_NV\_gpu\_program5
  \item GL_NV\_tessellation\_program5
\end{itemize}

PROFILE OPTIONS

All \textit{gp4} profile options are also available in \textit{gp5tcp}.
**Tessellation Control Domain-specific GP5 Options**

PATCH_1 ... PATCH_32

Specify the number of control points associated with each patch. Must be in the range [1,32]. The input varying `AttribArrays` parameter size is also this value.

The tessellation control shader will fail to load if PATCH_1 ... PATCH_32 is not specified as a profile option for the primitive type.

Usually the primitive input parameter is specified as a profile modifier preceding the Cg entry function.

**SEMANTICS**

**VARYING INPUT SEMANTICS**

The varying input semantics in the `gp5tcp` profile correspond to the respectively named varying output semantics of the `gp5vp` profile. The exception is the `CONTROLPOINT_ID` semantic.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTROLPOINT_ID</td>
<td>An input integer corresponding to the control point’s input order.</td>
</tr>
</tbody>
</table>

**VARYING OUTPUT SEMANTICS**

The varying output semantics in the `gp5tcp` profile correspond to the respectively named varying input semantics of the `gp5tep` profile. The two exceptions are the LOD parameters consumed by the tessellation primitive generator.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>EDGETESS</td>
<td>Output <code>float</code> array corresponding to outer tessellation level. (Size of array = 3 for triangle patches, 4 for quad patches.)</td>
</tr>
<tr>
<td>INNERTESS</td>
<td>Output <code>float</code> array corresponding to inner tessellation level. (Size of array = 1 for triangle patches, 2 for quad patches.)</td>
</tr>
<tr>
<td>PATCH</td>
<td>Output per-patch attribute.</td>
</tr>
</tbody>
</table>

An index does not need to be specified for other attributes. The output attributes are mapped to their corresponding control point array entry for subsequent shader stages.

For example the following will simply pass-through the control point’s position to the evaluation shader:

```c
void main(
    in int id : CONTROLPOINT_ID,
    in AttribArray<float3> inPositions : POSITION,
    out position : POSITION)
{
    position = inPositions[id];
}
```

**SEE ALSO**

`gp5`, `gp5tep`, `gp5vp`, `gp5gp`, `gp5fp`

### 3.19 gp5tep

**NAME**

`gp5tep` - OpenGL tessellation evaluation profile for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro
SYNOPSIS

gpstep

DESCRIPTION

This OpenGL profile corresponds to tessellation evaluation functionality introduced by NVIDIA’s 5th generation of assembly instruction sets.

The compiler output for this profile conforms to the assembly format defined by `NV_tessellation_program`, `NV_gpu_program5` and `ARB_vertex_program`.

3D API DEPENDENCIES

Requires OpenGL support for the `NV_gpu_program5` extension.

OpenGL Extension Specifications

- GL_NV_gpu_program5
- GL_NV_tessellation_program5

PROFILE OPTIONS

All `gp4` profile options are also available in `gpstep`.

Tessellation Evaluation Domain-specific GP5 Options

Normally the options below to specify primitive input and output parameters are better specified as a profile modifier preceding the Cg entry function. The tessellation evaluation shader will fail to load if either TRIANGLES, QUADS, or ISOLINES is not specified, and PATCH_1 ... PATCH_32 is not specified as profile options for the primitive type.

- PATCH_1 ... PATCH_32
  This specifies the number of control points associated with this patch. Must be 1 - 32 control points. This option affects the size of the AttribArrays associated with the input varying parameters.

- TRIANGLES
  Specifies that the input patch is a triangular patch. This affects the data type for the input varying semantics UV and TESSCOORD.

- QUADS
  Specifies that the input patch is a quadrangular patch. This affects the data type for the input varying semantics UV and TESSCOORD.

- ISOLINES
  The patch is drawn as a set of lines going in one parametric direction.

- POINT_MODE
  Specifies that the tessellation primitive generator will emit points for each vertex in the subdivided primitive instead of lines or triangles.

- ORDER_CCW
  Specifies counter clock-wise ordering of the output triangles emitted by the tessellation primitive generator. ORDER_CW is used by default.

- ORDER_CW
  Specifies clock-wise ordering of the output triangles emitted by the tessellation primitive generator. ORDER_CW is used by default.
SPACE_FRACTIONAL_EVEN

Specifies that the spacing of vertices emitted by the tessellation primitive generator can be fractional (non-integer) values, and the even spacing algorithm is used to perform the tessellation. Default is SPACE_EQUAL.

SPACE_FRACTIONAL_ODD

Specifies that the spacing of vertices emitted by the tessellation primitive generator can be fractional (non-integer) values, and the odd spacing algorithm is used to perform the tessellation. SPACE_EQUAL is used by default.

SPACE_EQUAL

Specifies that the spacing of vertices emitted by the tessellation primitive generator are clamped to integer values. This is the default option.

SEMANTICS

VARYING INPUT SEMANTICS

The varying input semantics in the gp5tep profile correspond to the respectively named varying output semantics of the gp5vp profile. All input semantics must use the AttribArray template-style syntax. The reason for this is because an evaluation shader must be able to see outputs of all control points of the patch to evaluate its unique position/outputs. The size of this AttribArray is determined by the PATCH_1 ... PATCH_32 profile parameter.

Example defining an input varying parameter which are output attributes of the control points’ gp5tcp invocations:

AttribArray< float3 > position : POSITION

There are additional input semantics specific to a gp5tep shader and these do not use an AttribArray type for input. They are as follows:

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TESSCOORD</td>
<td>Input u,v,w coordinates for the current vertex in a triangle patch.</td>
</tr>
<tr>
<td>UV</td>
<td>Input u,v coordinates for the current vertex in a quad patch.</td>
</tr>
<tr>
<td>PATCH</td>
<td>Input per-patch attribute.</td>
</tr>
</tbody>
</table>

SEE ALSO

gp5, gp5tcp, gp5vp, gp5gp, gp5fp

3.20 gp5vp

NAME

gp5vp - OpenGL vertex profile for NVIDIA GeForce 400/500 Series, OpenGL 4.x Quadro

SYNOPSIS

gp5vp

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by NVIDIA’s 5th generation of assembly instruction sets. It extends gp4vp.
The compiler output for this profile conforms to the assembly format defined by NV_gpu_program5 and ARB_vertex_program.

3D API DEPENDENCIES
Requires OpenGL support for the NV_gpu_program5 extension.

OpenGL Extension Specifications
   GL_NV_gpu_program5

PROFILE OPTIONS
All gp4vp profile options are also available in gp5vp.

SEE ALSO
gp4vp, gp5, gp5tcp, gp5tep, gp5gp, gp5fp

3.21 gs_4_0

NAME
gs_4_0 - Translation profile to DirectX 10’s High Level Shader Language for geometry shaders.

SYNOPSIS
   gs_4_0

DESCRIPTION
This Direct3D profile translates Cg into DirectX 10’s High Level Shader Language (HLSL10) for geometry shaders.

The compiler output for this profile conforms to the textual high-level language defined by DirectX 10’s High Level Shading Language.

The limitations of the gs_4_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES
Requires Direct3D 10 support.


PROFILE OPTIONS
   -po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL10 normally uses.

DATA TYPES
In general, the Cg data types translate to the HLSL10 data types with the same name.

   float, half, fixed

   These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

   int
This integral data type operates like an integer over the $-2^{24}$ to $2^{24}$ range. The result of int by int division is an integer (rounding down) to match C.

**SEMAPHTICS**

**INPUT SEMANTICS**

The varying input semantics in the gs_4_0 profile correspond to the respectively named varying output semantics of the vs_4_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4) (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Clip-space position (SV_Position)</td>
</tr>
<tr>
<td>HPOS</td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>Front primary color (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COLO</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Front secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
<tr>
<td>TEX#</td>
<td>TEX# is translated to TEXCOORD#</td>
</tr>
<tr>
<td>FOGC</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
<tr>
<td>PSIZE</td>
<td>Point size</td>
</tr>
<tr>
<td>PSIZ</td>
<td></td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

This profile is limited to standard library support available in HLSL10 for vertex shaders. In general, the Cg and HLSL10 standard libraries are very similar.
SEE ALSO

vs_4_0, ps_4_0

3.22 gs_5_0

NAME
gs_5_0 - Translation profile to DirectX 11’s High Level Shader Language for geometry shaders.

SYNOPSIS
gs_5_0

DESCRIPTION
This Direct3D profile translates Cg into DirectX 11’s High Level Shader Language (HLSL11) for geometry shaders.
The compiler output for this profile conforms to the textual high-level language defined by DirectX 11’s High Level Shading Language.
The limitations of the gs_5_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES
Requires Direct3D 11 support.


PROFILE OPTIONS
-po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL11 normally uses.

DATA TYPES
In general, the Cg data types translate to the HLSL11 data types with the same name.

float, half, fixed

These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

int

This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.

SEMANTICS

INPUT SEMANTICS
The varying input semantics in the gs_5_0 profile correspond to the respectively named varying output semantics of the vs_5_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
</tbody>
</table>
COLOR
COLOR0
DIFFUSE
COLOR1
SPECULAR
FOGCOORD
TEXCOORD#

UNIFORM INPUT SEMANTICS

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics</th>
<th>Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Clip-space position (SV_Position)</td>
<td></td>
</tr>
<tr>
<td>HPOS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>Front primary color (SV_Target)</td>
<td></td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Front secondary color</td>
<td></td>
</tr>
<tr>
<td>COL1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
<td></td>
</tr>
<tr>
<td>TEX#</td>
<td>TEX# is translated to TEXCOORD#</td>
<td></td>
</tr>
<tr>
<td>FOGC</td>
<td>Fog coordinate</td>
<td></td>
</tr>
<tr>
<td>FOG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PSIZE</td>
<td>Point size</td>
<td></td>
</tr>
<tr>
<td>PSIZ2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL11 for vertex shaders. In general, the Cg and HLSL11 standard libraries are very similar.

SEE ALSO

ps_5_0, vs_5_0, hs_5_0, ds_5_0

3.23 hlsl10

NAME

hlsl10 - DirectX 10 High Level Shader Language (HLSL) profiles

SYNOPSIS

hlsl10
DESCRIPTION

hlsl10 corresponds not to a single profile but to a family of profiles that include vs_4_0, gs_4_0 and ps_4_0. For more details refer to each profile documentation.

3D API DEPENDENCIES

Requires Direct3D 10 support.


STANDARD LIBRARY ISSUES

Cg standard library routines are available.

SEE ALSO

vs_4_0, gs_4_0, ps_4_0

3.24 hlsl11

NAME

hlsl11 - DirectX 11 High Level Shader Language (HLSL) profiles

SYNOPSIS

hlsl11

DESCRIPTION

hlsl11 corresponds not to a single profile but to a family of profiles that include vs_5_0, hs_5_0, ds_5_0, gs_5_0 and ps_5_0. For more details refer to each profile documentation.

3D API DEPENDENCIES

Requires Direct3D 11 support.


STANDARD LIBRARY ISSUES

Cg standard library routines are available.

SEE ALSO

vs_5_0, hs_5_0, ds_5_0, gs_5_0, ps_5_0

3.25 hlslf

NAME

hlslf - Translation profile to DirectX 9’s High Level Shader Language for pixel shaders.
SYNOPSIS
hlslf

DESCRIPTION
This Direct3D profile translates Cg into DirectX 9’s High Level Shader Language (HLSL) for pixel shaders.
The compiler output for this profile conforms to the textual high-level language defined by DirectX 9’s High Level Shading Language. See:


The limitations of the hlslf profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES
Requires Direct3D 9 support.

PROFILE OPTIONS
None.

DATA TYPES
In general, the Cg data types translate to the HLSL data types with the same name.

half
NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float
The float data type corresponds to a floating-point representation with at least 24 bits.
NVIDIA GPUs supporting hlslf use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.
Older ATI GPUs use 24-bit floating-point.

fixed
The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS
The varying input semantics in the hlslf profile correspond to the respectively named varying output semantics of the vs_2_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR 0</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
</tbody>
</table>
TEXTURE0

TEXT2

TEXTURE2

TEXTURE3

TEXTURE4

TEXTURE5

TEXTURE6

TEXTURE7

FOG

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td></td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td></td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td></td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

COLOR

COLOR0

COLOR1

COLOR

DEPR

Output depth (float)

STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL. In general, the Cg and HLSL standard libraries are very similar.

SEE ALSO

hlslv
3.26 hlslv

NAME

hlslv - Translation profile to DirectX 9’s High Level Shader Language for vertex shaders.

SYNOPSIS

hlslv

DESCRIPTION

This Direct3D profile translates Cg into DirectX 9’s High Level Shader Language (HLSL) for pixel shaders. The compiler output for this profile conforms to the textual high-level language defined by DirectX 9’s High Level Shading Language. See:


The limitations of the hlslv profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES

Requires Direct3D 9 support.

PROFILE OPTIONS

None.

DATA TYPES

In general, the Cg data types translate to the HLSL data types with the same name.

float, half, fixed

These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

int

This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the hlslv profile correspond to the respectively named varying output semantics of the ps_2_0 profile.

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td></td>
<td>Object-space position</td>
</tr>
<tr>
<td>NORMAL</td>
<td></td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td></td>
<td>Primary color (float4)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td></td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
FOGCOORD Fog coordinate

TEXCOORD# Texture coordinate set #

**UNIFORM INPUT SEMANTICS**

*to-be-written*

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Clip-space position</td>
</tr>
<tr>
<td>HPOS</td>
<td></td>
</tr>
<tr>
<td>COLOR</td>
<td>Front primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Front secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
<tr>
<td>TEX#</td>
<td></td>
</tr>
<tr>
<td>FOGC</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
<tr>
<td>PSIZE</td>
<td>Point size</td>
</tr>
<tr>
<td>PSIZ</td>
<td></td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

This profile is limited to standard library support available in HLSL for vertex shaders. In general, the Cg and HLSL standard libraries are very similar.

**SEE ALSO**

*hlslf*

### 3.27 hs_5_0

**NAME**

hs_5_0 - Translation profile to DirectX 11’s High Level Shader Language for hull shaders.

**SYNOPSIS**

vs_5_0

**DESCRIPTION**

This Direct3D profile translates Cg into DirectX 11’s High Level Shader Language (HLSL11) for hull shaders.

The compiler output for this profile conforms to the textual high-level language defined by DirectX 11’s High Level Shading Language.
The limitations of the vs_5_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES

Requires Direct3D 11 support.


PROFILE OPTIONS

-po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL11 normally uses.

DATA TYPES

In general, the Cg data types translate to the HLSL11 data types with the same name.

float, half, fixed

These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

int

This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the hs_5_0 profile correspond to the respectively named varying output semantics of the vs_5_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4) (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
</tbody>
</table>

UNIFORM INPUT SEMANTICS

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Clip-space position (SV_Position)</td>
</tr>
<tr>
<td>HPOS</td>
<td></td>
</tr>
</tbody>
</table>
COLOR Front primary color (SV_Target)
COLOR0
COL0
COL
COLOR1 Front secondary color
COL1
TEXCOORD# Texture coordinate set #
TEX# TEX# is translated to TEXCOORD#
FOGC Fog coordinate
FOG
PSIZE Point size
PSIZ

STANDARD LIBRARY ISSUES
This profile is limited to standard library support available in HLSL11 for hull shaders. In general, the Cg and HLSL11 standard libraries are very similar.

SEE ALSO
ps_5_0, vs_5_0, gs_5_0, ds_5_0

3.28 ps_1_1

NAME
ps_1_1 - Direct3D Shader Model 1.1 fragment profile for DirectX 8

SYNOPSIS
ps_1_1

DESCRIPTION
This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce 2 (NV1x) for DirectX 8. The compiler output for this profile conforms to the textual assembly defined by DirectX 8’s Pixel Shader 1.1 shader format. See:

3D API DEPENDENCIES
Requires Direct3D 8 or 9 support.

PROFILE OPTIONS
*to-be-written*

DATA TYPES
*to-be-written*
SEMANTICS

INPUT SEMANTICS

*to-be-written*

UNIFORM INPUT SEMANTICS

*to-be-written*

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are *not* supported.

SEE ALSO

vs_1_1, ps_1_3, ps_1_2

3.29 ps_1_2

NAME

ps_1_2 - Direct3D Shader Model 1.2 fragment profile for DirectX 8

SYNOPSIS

ps_1_2

DESCRIPTION

This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce 2 (NV1x) for DirectX 8. The compiler output for this profile conforms to the textual assembly defined by DirectX 8's Pixel Shader 1.2 shader format. See:


3D API DEPENDENCIES

Requires Direct3D 8 or 9 support.

PROFILE OPTIONS

*to-be-written*

DATA TYPES

*to-be-written*

SEMANTICS

INPUT SEMANTICS

*to-be-written*

UNIFORM INPUT SEMANTICS

*to-be-written*

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are *not* supported.
SEE ALSO

vs_1_1, ps_1_3, ps_1_1

3.30 ps_1_3

NAME

ps_1_3 - Direct3D Shader Model 1.3 fragment profile for DirectX 8

SYNOPSIS

ps_1_3

DESCRIPTION

This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce 3 (NV2x) for DirectX 8.

The compiler output for this profile conforms to the textual assembly defined by DirectX 8’s Pixel Shader 1.3 shader format. See:


3D API DEPENDENCIES

Requires Direct3D 8 or 9 support.

PROFILE OPTIONS

to-be-written

DATA TYPES

to-be-written

SEMANTICS

INPUT SEMANTICS

to-be-written

UNIFORM INPUT SEMANTICS

to-be-written

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.

SEE ALSO

vs_1_1, ps_1_2, ps_1_1

3.31 ps_2_0

NAME

ps_2_0 - Direct3D Shader Model 2.0 fragment profile for DirectX 9
SYNOPSIS

ps_2_0

DESCRIPTION

This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Pixel Shader 2.0 shader format. See:


3D API DEPENDENCIES

Requires Direct3D 9 support.

PROFILE OPTIONS

NumTemps=$val$
Number of 4-component vector temporaries the target implementation supports.

NumInstructionSlots=$val$
Number of instructions the target implementation supports.

MaxDrawBuffers=$val$
Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

DATA TYPES

half
The half data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called $s10e5$.

float
The float data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting ps_2_0 use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called $s10e5$.

Older ATI GPUs use 24-bit floating-point.

fixed
The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the ps_2_0 profile correspond to the respectively named varying output semantics of the vs_2_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
</tbody>
</table>
COL0
COLOR1 Input secondary color
COL1
TEX0 Input texture coordinate sets 0
TEXCOORD0
TEX1 Input texture coordinate sets 1
TEXCOORD1
TEX2 Input texture coordinate sets 2
TEXCOORD2
TEX3 Input texture coordinate sets 3
TEXCOORD3
TEX4 Input texture coordinate sets 4
TEXCOORD4
TEX5 Input texture coordinate sets 5
TEXCOORD5
TEX6 Input texture coordinate sets 6
TEXCOORD6
TEX7 Input texture coordinate sets 7
TEXCOORD7
FOGP Input fog color (XYZ) and factor (W)
FOG

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

COLOR Output color (float4)
COLOR0
COL0
COL

DEPTH Output depth (float)
DEPR

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.
This profile may have limits on the number of dependent texture fetches.
SEE ALSO

vs_2_0, ps_2_x

## 3.32 ps_2_sw

### NAME

**ps_2_sw** - Direct3D Software Shader for Model 2.0 Extended fragment profile

### SYNOPSIS

`ps_2_sw`

### DESCRIPTION

This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by the software version of DirectX 9’s Pixel Shader 2.0 Extended shader format. See:


This profile is useful for debugging and prototyping.

### 3D API DEPENDENCIES

Requires a reference device.

This profile generates code assuming the following Direct3D 9 pixel shader capability bits are set:

- `D3DD3DPShaderCaps2_0_ArbitrarySwizzle`
- `D3DD3DPShaderCaps2_0_GradientInstructions`
- `D3DD3DPShaderCaps2_0_Predication`
- `D3DD3DPShaderCaps2_0_NonDependentReadLimit`
- `D3DD3DPShaderCaps2_0_NotExInstructionLimit`

### PROFILE OPTIONS

- **NumTemps** = `val`
  Number of 4-component vector temporaries the target implementation supports.

- **NumInstructionSlots** = `val`
  Number of instructions the target implementation supports.

- **MaxDrawBuffers** = `val`
  Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

### DATA TYPES

- **half**
  The **half** data type makes use of the Partial Precision instruction modifier to request less precision. NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called `s10e5`.

- **float**
The `float` data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting `ps_2_sw` use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called `s10e5`.

Older ATI GPUs use 24-bit floating-point.

The `fixed` data type is treated like `half`.

**SEMANTICS**

**INPUT SEMANTICS**

The varying input semantics in the `ps_2_sw` profile correspond to the respectively named varying output semantics of the `vs_2_sw` profile.

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td></td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TEXUNIT0  Texture unit 0
TEXUNIT1  Texture unit 1
...
TEXUNIT15 Texture unit 15

OUTPUT SEMANTICS

COLOR       Output color (float4)
COLOR0      COL0
COLOR
DEPTH      Output depth (float)
DEPR

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.
There are no restrictions on dependent texture reads (up to the instruction limit) for this profile.

SEE ALSO

vs_2_sw, ps_2_x

3.33 ps_2_x

NAME

ps_2_x - Direct3D Shader Model 2.0 Extended fragment profile for DirectX 9

SYNOPSIS

ps_2_x

DESCRIPTION

This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce FX (NV3x) for DirectX 9.
The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Pixel Shader 2.0 Extended shader format. See:


This profile supports static and structured dynamic flow control.

3D API DEPENDENCIES

Requires Direct3D 9 support.

This profile generates code assuming the following Direct3D 9 pixel shader capability bits are set:

D3DD3DPHADERCAPS2_0_ARBITRARYSWIZZLE
D3DD3DPHADERCAPS2_0_GRADIENTINSTRUCTIONS
D3DD3DPHADERCAPS2_0_PREDICATION
D3DD3DPHADERCAPS2_0_NODEPENDENTREADLIMIT
D3DD3DPHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT

3.33. ps_2_x
PROFILE OPTIONS

NumTemps = val
Number of 4-component vector temporaries the target implementation supports.

NumInstructionSlots = val
Number of instructions the target implementation supports.

MaxDrawBuffers = val
Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

DATA TYPES

half
The half data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float
The float data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting ps_2_x use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

Older ATI GPUs use 24-bit floating-point.

fixed
The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the ps_2_x profile correspond to the respectively named varying output semantics of the vs_2_x profile.

<table>
<thead>
<tr>
<th>Binding Semantics</th>
<th>Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>COLOR0</td>
<td>Input primary color</td>
</tr>
<tr>
<td></td>
<td>COLOR0</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COL</td>
<td>COL0</td>
<td>Input secondary color</td>
</tr>
<tr>
<td></td>
<td>COL0</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COLOR1</td>
<td>COLOR1</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>COL1</td>
<td>COLOR1</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEX0</td>
<td>TEXCOORD0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td>TEXCOORD0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEX1</td>
<td>TEXCOORD1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td>TEXCOORD1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEX2</td>
<td>TEXCOORD2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td>TEXCOORD2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEX3</td>
<td>TEXCOORD3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td>TEXCOORD3</td>
<td>Input texture coordinate sets 3</td>
</tr>
</tbody>
</table>
TEXCOORD3
TEX4 Input texture coordinate sets 4
TEXCOORD4
TEX5 Input texture coordinate sets 5
TEXCOORD5
TEX6 Input texture coordinate sets 6
TEXCOORD6
TEX7 Input texture coordinate sets 7
TEXCOORD7
FOGP Input fog color (XYZ) and factor (W)
FOG

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>Texture unit 15</td>
<td></td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR Output color (float4)</td>
</tr>
<tr>
<td>COLOR0</td>
</tr>
<tr>
<td>COL0</td>
</tr>
<tr>
<td>COL</td>
</tr>
<tr>
<td>DEPTH Output depth (float)</td>
</tr>
<tr>
<td>DEPR</td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.

There are no restrictions on dependent texture reads (up to the instruction limit) for this profile.

SEE ALSO

vs_2_x, ps_2_0

3.34 ps_3_0

NAME

ps_3_0 - Direct3D Shader Model 3.0 fragment profile for DirectX 9

SYNOPSIS

ps_3_0
DESCRIPTION
This Direct3D profile corresponds to the per-fragment functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Pixel Shader 3.0 shader format. See:


Data-dependent loops are allowed with a limit of 256 iterations maximum. Four levels of nesting are allowed.

Conditional expressions can be supported with data-dependent branching.

Relative indexing of uniform arrays is not supported; use texture accesses instead.

3D API DEPENDENCIES
Requires Direct3D 9 support.

PROFILE OPTIONS
None.

DATA TYPES

half

The half data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float

float values in ps_3_0 require standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

fixed

The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS
The varying input semantics in the ps_3_0 profile correspond to the respectively named varying output semantics of the vs_3_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
</tbody>
</table>
UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

Binding Semantic Name Corresponding Data
TEXUNIT0 Texture unit 0
TEXUNIT1 Texture unit 1
... Texture unit 15

OUTPUT SEMANTICS

COLOR Output color (float4)
COLOR0
COLO
COL
DEPTH Output depth (float)
DEPR

STANDARD LIBRARY ISSUES

This profile may have limits on the number of dependent texture fetches.

SEE ALSO

vs_3_0

3.35 ps_4_0

NAME

ps_4_0 - Translation profile to DirectX 10’s High Level Shader Language for pixel shaders.
SYNOPSIS

ps_4_0

DESCRIPTION

This Direct3D profile translates Cg into DirectX 10’s High Level Shader Language (HLSL10) for pixel shaders. The compiler output for this profile conforms to the textual high-level language defined by DirectX 10’s High Level Shading Language.

The limitations of the ps_4_0 profile depend on what HLSL10 profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES

Requires Direct3D 10 support.


PROFILE OPTIONS

-po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL10 normally uses.

DATA TYPES

In general, the Cg data types translate to the HLSL10 data types with the same name.

half

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float

The float data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting ps_4_0 use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

Older ATI GPUs use 24-bit floating-point.

fixed

The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the ps_4_0 profile correspond to the respectively named varying output semantics of the vs_4_0 profile.

<table>
<thead>
<tr>
<th>Binding Semantics</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
</tbody>
</table>
**UNIFORM INPUT SEMANTICS**

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Output color (float4)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>DEPTH</td>
<td>Output depth (float)</td>
</tr>
<tr>
<td>DEPR</td>
<td></td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

This profile is limited to standard library support available in HLSL10. In general, the Cg and HLSL10 standard libraries are very similar.

**SEE ALSO**

`vs_4_0`, `gs_4_0`
3.36 ps_5_0

NAME

ps_5_0 - Translation profile to DirectX 11’s High Level Shader Language for pixel shaders.

SYNOPSIS

ps_5_0

DESCRIPTION

This Direct3D profile translates Cg into DirectX 11’s High Level Shader Language (HLSL11) for pixel shaders.

The compiler output for this profile conforms to the textual high-level language defined by DirectX 11’s High Level Shading Language.

The limitations of the ps_5_0 profile depend on what HLSL11 profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES

Requires Direct3D 11 support.


PROFILE OPTIONS

-po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL11 normally uses.

DATA TYPES

In general, the Cg data types translate to the HLSL11 data types with the same name.

half

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called $s10e5$.

float

The float data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting ps_5_0 use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called $s10e5$.

Older ATI GPUs use 24-bit floating-point.

fixed

The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS

The varying input semantics in the ps_5_0 profile correspond to the respectively named varying output semantics of the vs_5_0 profile.
<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td>TEX# translates to TEXCOORD#</td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Output color (float4)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>DEPTH</td>
<td>Output depth (float)</td>
</tr>
<tr>
<td>DEPR</td>
<td></td>
</tr>
</tbody>
</table>
STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL11. In general, the Cg and HLSL11 standard libraries are very similar.

SEE ALSO

hs_5_0, vs_5_0, gs_5_0, ds_5_0

3.37 vp20

NAME

vp20 - OpenGL fragment profile for NV2x (GeForce3, GeForce4 Ti, Quadro DCC, etc.)

SYNOPSIS

vp20

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by GeForce3.

The compiler output for this profile conforms to the assembly format defined by NV_vertex_program1_1 (which assumes NV_vertex_program).

Data-dependent loops are not allowed; all loops must be unrollable.

Conditional expressions are supported without branching so both conditions must be evaluated.

Relative indexing of uniform arrays is supported; but texture accesses are not supported.

3D API DEPENDENCIES

Requires OpenGL support for NV_vertex_program and NV_vertex_program1_1 extensions. These extensions were introduced by GeForce3 and Quadro DCC GPUs.

OpenGL Extension Specifications

    GL_NV_vertex_program
    GL_NV_vertex_program1_1

PROFILE OPTIONS

    PosInv=val
    Non-zero means generates code for position-invariant (with fixed-function) position transformation.
    MaxLocalParams=val
    Maximum number of local parameters the implementation supports.

DATA TYPES

to-be-written

SEMAN TICS

VARYING INPUT SEMANTICS

to-be-written
UNIFORM INPUT SEMANTICS
to-be-written

OUTPUT SEMANTICS
to-be-written

STANDARD LIBRARY ISSUES
to-be-written

SEE ALSO
fp20

3.38 vp30

NAME
vp30 - OpenGL fragment profile for NV3x (GeForce FX, Quadro FX, etc.)

SYNOPSIS
vp30

DESCRIPTION
This OpenGL profile corresponds to the per-vertex functionality introduced by the GeForce FX and Quadro FX line of NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by NV_vertex_program2.

Data-dependent loops and branching are allowed.

Relative indexing of uniform arrays is supported; but texture accesses are not supported.

3D API DEPENDENCIES
Requires OpenGL support for the NV_vertex_program2 extension. These extensions were introduced by the GeForce FX and Quadro FX GPUs.

OpenGL Extension Specifications
GL_NV_vertex_program2

PROFILE OPTIONS
PosInv=val
Non-zero means generates code for position-invariant (with fixed-function) position transformation.

MaxLocalParams=val
Maximum number of local parameters the implementation supports.

DATA TYPES
to-be-written
SEMANTICS

VARYING INPUT SEMANTICS
to-be-written

UNIFORM INPUT SEMANTICS
to-be-written

OUTPUT SEMANTICS
to-be-written

STANDARD LIBRARY ISSUES
to-be-written

SEE ALSO

fp30

3.39 vp40

NAME

vp40 - OpenGL vertex profile for NVIDIA GeForce 6/7 Series, NV4x-based Quadro FX

SYNOPSIS

vp40

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by the GeForce 6800 and other NV4x-based NVIDIA GPUs.

The compiler output for this profile conforms to the assembly format defined by NV_vertex_program3 and ARB_vertex_program.

Data-dependent loops and branching are allowed.

Relative indexing of uniform arrays is supported.

Texture accesses are supported. However substantial limitations on vertex texturing exist for hardware acceleration by NV4x hardware.

NV4x hardware accelerates vertex fetches only for 1-, 3-, and 4-component floating-point textures. NV4x hardware does not accelerated vertex-texturing for cube maps or 3D textures. NV4x does allow non-power-of-two sizes (width and height).

3D API DEPENDENCIES

Requires OpenGL support for the NV_fragment_program3 extension. These extensions were introduced by the GeForce 6800 and other NV4x-based GPUs.

OpenGL Extension Specifications

GL_NV_vertex_program3
GL_ARB_vertex_program
PROFILE OPTIONS

PosInv=val
Non-zero means generates code for position-invariant (with fixed-function) position transformation.

NumTemps=val
Maximum number of temporary registers the implementation supports. Set to the implementaiton-dependent value of GL_MAX_NATIVE_TEMPORARIES_ARB for best performance.

MaxAddressRegs=val
Maximum number of address registers the implementation supports. Set to the implementaiton-dependent value of GL_MAX_NATIVE_ADDRESS_REGISTERS_ARB for best performance.

MaxInstructions=val
Maximum number of instructions the implementation supports. Set to the implementaiton-dependent value of GL_MAX_NATIVE_INSTRUCTIONS_ARB for best performance.

MaxLocalParams=val
Maximum number of local parameters the implementation supports.

DATA TYPES

float
This profile implements the float data type as IEEE 32-bit single precision.

half
double
half and double data types are treated as float.

int
The int data type is supported using floating point operations, which adds extra instructions for proper truncation for divides, modulus and casts from floating point types. Because integer values are simulated with IEEE single-precision floating-point, they are accurate over the range -2^24 to 2^24 but will not overflow like true integers.

fixed
sampler*
This profile does not support any operations on the fixed or sampler* data types, but does provide the minimal partial support that is required for these data types by the core language spec (i.e. it is legal to declare variables using these types, as long as no operations are performed on the variables).

SEMANTICS

VARYING INPUT SEMANTICS

The set of binding semantics for varying input data to vp40 consists of POSITION, BLENDWEIGHT, NORMAL, COLOR0, COLOR1, TESSFACTOR, PSIZE, BLENDINDICES, and TEXCOORD0-TEXCOORD7. One can also use TANGENT and BINORMAL instead of TEXCOORD6 and TEXCOORD7.

Additionally, a set of binding semantics ATTR0-ATTR15 can be used. These binding semantics map to NV_vertex_program3 input attribute parameters.

The two sets act as aliases to each other on NVIDIA GPUs excluding Apple Macs. ATI GPUs and NVIDIA Mac GPUs do not alias the conventional vertex attributes with the generic attributes.
<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, ATTR0</td>
<td>Input Vertex, Generic Attribute 0</td>
</tr>
<tr>
<td>BLENDWEIGHT, ATTR1</td>
<td>Input vertex weight, Generic Attribute 1</td>
</tr>
<tr>
<td>NORMAL, ATTR2</td>
<td>Input normal, Generic Attribute 2</td>
</tr>
<tr>
<td>DIFFUSE, COLOR0, ATTR3</td>
<td>Input primary color, Generic Attribute 3</td>
</tr>
<tr>
<td>SPECULAR, COLOR1, ATTR4</td>
<td>Input secondary color, Generic Attribute 4</td>
</tr>
<tr>
<td>TESSFACTOR, FOGCOORD, ATTR5</td>
<td>Input fog coordinate, Generic Attribute 5</td>
</tr>
<tr>
<td>PSIZE, ATTR6</td>
<td>Input point size, Generic Attribute 6</td>
</tr>
<tr>
<td>BLENDINDICES, ATTR7</td>
<td>Generic Attribute 7</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7, ATTR8-ATTR15</td>
<td>Input texture coordinates (texcoord0-texcoord7)</td>
</tr>
<tr>
<td>TANGENT, ATTR14</td>
<td>Generic Attribute 14</td>
</tr>
<tr>
<td>BINORMAL, ATTR15</td>
<td>Generic Attribute 15</td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

C0-C255  Constant register \([0..255]\).
The aliases c0-c255 (lowercase) are also accepted.

If used with a variable that requires more than one constant register (e.g. a matrix), the semantic specifies the first register that is used.

TEXUNIT0-TEXUNIT15  Texture unit (but only 4 distinct texture units are allowed)

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7, TEX0-TEX7</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>
STANDARD LIBRARY ISSUES
Standard library routines for texture cube map and rectangle samplers are not supported by vp40.

SEE ALSO

fp40

3.40 vs_1_1

NAME
vs_1_1 - Direct3D Shader Model 1.1 vertex profile for DirectX 8

SYNOPSIS
vs_1_1

DESCRIPTION
This Direct3D profile corresponds to the per-vertex functionality introduced by GeForce 2 (NV1x) for DirectX 8. The compiler output for this profile conforms to the textual assembly defined by DirectX 8’s Vertex Shader 1.1 shader format. See:


3D API DEPENDENCIES
Requires Direct3D 8 or 9 support.

PROFILE OPTIONS
to-be-written

DATA TYPES
to-be-written

SEMANTICS

INPUT SEMANTICS
to-be-written

UNIFORM INPUT SEMANTICS
to-be-written

STANDARD LIBRARY ISSUES
Functions that compute partial derivatives are not supported.

SEE ALSO

ps_1_3, ps_1_2, ps_1_1
3.41 vs_2_0

NAME
vs_2_0 - Direct3D Shader Model 2.0 vertex profile for DirectX 9

SYNOPSIS
vs_2_0

DESCRIPTION
This Direct3D profile corresponds to the per-vertex functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Vertex Shader 2.0 shader format. See:


3D API DEPENDENCIES
Requires Direct3D 9 support.

PROFILE OPTIONS

NumTemps=\textit{val}
Number of 4-component vector temporaries the target implementation supports.

NumInstructionSlots=\textit{val}
Number of instructions the target implementation supports.

MaxDrawBuffers=\textit{val}
Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

DATA TYPES

\textit{half}
The \textit{half} data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called \textit{s10e5}.

\textit{float}
The \textit{float} data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting \textit{vs_2_0} use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called \textit{s10e5}.

Older ATI GPUs use 24-bit floating-point.

\textit{fixed}
The \textit{fixed} data type is treated like \textit{half}.
### SEMANTICS

#### INPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td></td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>

#### UNIFORM INPUT SEMANTICS

Eight texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td>TEXUNIT7</td>
<td>Texture unit 7</td>
</tr>
</tbody>
</table>

#### OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
</tbody>
</table>
FOG, FOGC Output fog coordinate
COLOR0, COL0 Output primary color
COLOR1, COL1 Output secondary color
BCOL0 Output backface primary color
BCOL1 Output backface secondary color
TEXCOORD0-TEXCOORD7, TEX0-TEX7 Output texture coordinates
CLP0-CL5 Output Clip distances

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.
This profile may have limits on the number of dependent texture fetches.

SEE ALSO

ps_2_0, vs_2_x

3.42 vs_2_sw

NAME

vs_2_sw - Direct3D Software Shader for Model 2.0 Extended vertex profile

SYNOPSIS

vs_2_sw

DESCRIPTION

This Direct3D profile corresponds to the per-vertex functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by the software version of DirectX 9’s Vertex Shader 2.0 Extended shader format. See:


This profile is useful for debugging and prototyping.

3D API DEPENDENCIES

Requires support for software vertex processing and a reference device.
This profile generates code assuming the following Direct3D 9 Vertex shader capability bits are set:

D3DD3DPSHADERCAPS2_0_ARBITRARYSWIZZLE
D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS
D3DD3DPSHADERCAPS2_0_PREDICATION
D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT
D3DD3DPSHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT
PROFILE OPTIONS

NumTemps=\textit{val}
Number of 4-component vector temporaries the target implementation supports.

NumInstructionSlots=\textit{val}
Number of instructions the target implementation supports.

MaxDrawBuffers=\textit{val}
Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

DATA TYPES

\texttt{half}

The \texttt{half} data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called \textit{s10e5}.

\texttt{float}

The \texttt{float} data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting \texttt{vs_2_sw} use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called \textit{s10e5}.

Older ATI GPUs use 24-bit floating-point.

\texttt{fixed}

The \texttt{fixed} data type is treated like \texttt{half}.

SEMANTICS

INPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
</tbody>
</table>
TEXCOORD
TEX5
TEXCOORD5
TEX6
TEXCOORD6
TEX7
TEXCOORD7
FOGP
FOG

**UNIFORM INPUT SEMANTICS**

Eight texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT7</td>
<td>Texture unit 7</td>
</tr>
</tbody>
</table>

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7,</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>TEX0-TEX7</td>
<td></td>
</tr>
<tr>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

Functions that compute partial derivatives are *not* supported.

There are no restrictions on dependent texture reads (up to the instruction limit) for this profile.

**SEE ALSO**

*ps_2_sw, vs_2_x*
3.43  vs_2_x

NAME

vs_2_x - Direct3D Shader Model 2.0 Extended vertex profile for DirectX 9

SYNOPSIS

vs_2_x

DESCRIPTION

This Direct3D profile corresponds to the per-vertex functionality introduced by GeForce FX (NV3x) for DirectX 9. The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Vertex Shader 2.0 Extended shader format. See:


This profile supports static and structured dynamic flow control.

3D API DEPENDENCIES

Requires Direct3D 9 support.

This profile generates code assuming the following Direct3D 9 Vertex shader capability bits are set:

D3DD3DPHADERCAPS2_0_ARBITRARYSWIZZLE
D3DD3DPHADERCAPS2_0_GRADIENTINSTRUCTIONS
D3DD3DPHADERCAPS2_0_PREDICATION
D3DD3DPHADERCAPS2_0_NODEPENDENTREADLIMIT
D3DD3DPHADERCAPS2_0_NOTEXINSTRUCTIONLIMIT

PROFILE OPTIONS

NumTemps=val

Number of 4-component vector temporaries the target implementation supports.

NumInstructionSlots=val

Number of instructions the target implementation supports.

MaxDrawBuffers=val

Number of draw buffers or Multiple Render Targets (MRT) the target implementation supports.

DATA TYPES

half

The half data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float

The float data type corresponds to a floating-point representation with at least 24 bits.

NVIDIA GPUs supporting vs_2_x use standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.
Older ATI GPUs use 24-bit floating-point.

fixed

The fixed data type is treated like half.

**SEMANTICS**

**INPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
</tr>
<tr>
<td>TEX7</td>
<td>Input texture coordinate sets 7</td>
</tr>
<tr>
<td>TEXCOORD7</td>
<td></td>
</tr>
<tr>
<td>FOGP</td>
<td>Input fog color (XYZ) and factor (W)</td>
</tr>
<tr>
<td>FOG</td>
<td></td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

Eight texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT7</td>
<td>Texture unit 7</td>
</tr>
</tbody>
</table>
OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics</th>
<th>Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
<td></td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
<td></td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
<td></td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
<td></td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
<td></td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
<td></td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
<td></td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7, TEX0-TEX7</td>
<td>Output texture coordinates</td>
<td></td>
</tr>
<tr>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
<td></td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives are not supported.

There are no restrictions on dependent texture reads (up to the instruction limit) for this profile.

SEE ALSO

ps_2_x, vs_2_0

3.44 vs_3_0

NAME

vs_3_0 - Direct3D Shader Model 3.0 vertex profile for DirectX 9

SYNOPSIS

vs_3_0

DESCRIPTION

This Direct3D profile corresponds to the per-vertex functionality introduced by GeForce FX (NV3x) for DirectX 9.

The compiler output for this profile conforms to the textual assembly defined by DirectX 9’s Vertex Shader 3.0 shader format. See:


Data-dependent loops are allowed with a limit of 256 iterations maximum. Four levels of nesting are allowed.

Conditional expressions can be supported with data-dependent branching.

Relative indexing of uniform arrays is not supported; use texture accesses instead.
3D API DEPENDENCIES

Requires Direct3D 9 support.

PROFILE OPTIONS

None.

DATA TYPES

half

The half data type makes use of the Partial Precision instruction modifier to request less precision.

NVIDIA GPUs may use half-precision floating-point when the Partial Precision instruction modifier is specified. Half-precision floating-point is encoded with a sign bit, 10 mantissa bits, and 5 exponent bits (biased by 16), sometimes called s10e5.

float

float values in vs_3.0 require standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

fixed

The fixed data type is treated like half.

SEMANTICS

INPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding</th>
<th>Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>COLOR</td>
<td></td>
<td>Input primary color</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COL0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td></td>
<td>Input secondary color</td>
</tr>
<tr>
<td>COL1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX0</td>
<td></td>
<td>Input texture coordinate sets 0</td>
</tr>
<tr>
<td>TEXCOORD0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX1</td>
<td></td>
<td>Input texture coordinate sets 1</td>
</tr>
<tr>
<td>TEXCOORD1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX2</td>
<td></td>
<td>Input texture coordinate sets 2</td>
</tr>
<tr>
<td>TEXCOORD2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX3</td>
<td></td>
<td>Input texture coordinate sets 3</td>
</tr>
<tr>
<td>TEXCOORD3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX4</td>
<td></td>
<td>Input texture coordinate sets 4</td>
</tr>
<tr>
<td>TEXCOORD4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX5</td>
<td></td>
<td>Input texture coordinate sets 5</td>
</tr>
<tr>
<td>TEXCOORD5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TEX6</td>
<td></td>
<td>Input texture coordinate sets 6</td>
</tr>
<tr>
<td>TEXCOORD6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**UNIFORM INPUT SEMANTICS**

Eight texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT7</td>
<td>Texture unit 7</td>
</tr>
</tbody>
</table>

**OUTPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0-TEXCOORD7,</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>TEX0-TEX7</td>
<td></td>
</tr>
<tr>
<td>CLP0-CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>

**STANDARD LIBRARY ISSUES**

This profile may have limits on the number of dependent texture fetches.

**SEE ALSO**

*ps_3_0*

**3.45 vs_4_0**

**NAME**

*vs_4_0* - Translation profile to DirectX 10’s High Level Shader Language for vertex shaders.
SYNOPSIS

vs_4_0

DESCRIPTION

This Direct3D profile translates Cg into DirectX 10’s High Level Shader Language (HLSL10) for vertex shaders.

The compiler output for this profile conforms to the textual high-level language defined by DirectX 10’s High Level Shading Language.

The limitations of the vs_4_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

3D API DEPENDENCIES

Requires Direct3D 10 support.


PROFILE OPTIONS

-po pad16  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL10 normally uses.

DATA TYPES

In general, the Cg data types translate to the HLSL10 data types with the same name.

float, half, fixed

These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

int

This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.

SEMANTICS

INPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4) (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
</tbody>
</table>
UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>

OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0–TEXCOORD7,</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>TEX0–TEX7</td>
<td></td>
</tr>
<tr>
<td>CLP0–CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL10 for vertex shaders. In general, the Cg and HLSL10 standard libraries are very similar.

SEE ALSO

gs_4_0, ps_4_0

3.46 vs_5_0

NAME

vs_5_0 - Translation profile to DirectX 11’s High Level Shader Language for vertex shaders.

SYNOPSIS

vs_5_0

DESCRIPTION

This Direct3D profile translates Cg into DirectX 11’s High Level Shader Language (HLSL11) for vertex shaders.
The compiler output for this profile conforms to the textual high-level language defined by DirectX 11’s High Level Shading Language.

The limitations of the vs_5_0 profile depend on what HLSL profile to which the translated HLSL code is compiled.

**3D API DEPENDENCIES**

Requires Direct3D 11 support.


**PROFILE OPTIONS**

- **-po pad16**  This will add padding variables to the cbuffer declarations to match the 16 byte padding the GP4 OpenGL profiles use. This makes sure each variable in the cbuffer uses an entire float4 constant instead of the tight packing HLSL11 normally uses.

**DATA TYPES**

In general, the Cg data types translate to the HLSL11 data types with the same name.

- **float, half, fixed**
  
  These numeric data types all correspond to standard IEEE 754 single-precision floating-point encoding with a sign bit, 23 mantissa bits, and 8 exponent bits (biased by 128), sometimes called s10e5.

- **int**
  
  This integral data type operates like an integer over the -2^24 to 2^24 range. The result of int by int division is an integer (rounding down) to match C.

**SEMANATICS**

**INPUT SEMANTICS**

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION</td>
<td>Object-space position (SV_Position)</td>
</tr>
<tr>
<td>NORMAL</td>
<td>Object-space normal</td>
</tr>
<tr>
<td>COLOR</td>
<td>Primary color (float4) (SV_Target)</td>
</tr>
<tr>
<td>COLOR0</td>
<td></td>
</tr>
<tr>
<td>DIFFUSE</td>
<td></td>
</tr>
<tr>
<td>COLOR1</td>
<td>Secondary color (float4)</td>
</tr>
<tr>
<td>SPECULAR</td>
<td></td>
</tr>
<tr>
<td>FOGCOORD</td>
<td>Fog coordinate</td>
</tr>
<tr>
<td>TEXCOORD#</td>
<td>Texture coordinate set #</td>
</tr>
</tbody>
</table>

**UNIFORM INPUT SEMANTICS**

Sixteen texture units are supported:

<table>
<thead>
<tr>
<th>Binding Semantic Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>TEXUNIT0</td>
<td>Texture unit 0</td>
</tr>
<tr>
<td>TEXUNIT1</td>
<td>Texture unit 1</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>TEXUNIT15</td>
<td>Texture unit 15</td>
</tr>
</tbody>
</table>
OUTPUT SEMANTICS

<table>
<thead>
<tr>
<th>Binding Semantics Name</th>
<th>Corresponding Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>POSITION, HPOS</td>
<td>Output position</td>
</tr>
<tr>
<td>PSIZE, PSIZ</td>
<td>Output point size</td>
</tr>
<tr>
<td>FOG, FOGC</td>
<td>Output fog coordinate</td>
</tr>
<tr>
<td>COLOR0, COL0</td>
<td>Output primary color</td>
</tr>
<tr>
<td>COLOR1, COL1</td>
<td>Output secondary color</td>
</tr>
<tr>
<td>BCOL0</td>
<td>Output backface primary color</td>
</tr>
<tr>
<td>BCOL1</td>
<td>Output backface secondary color</td>
</tr>
<tr>
<td>TEXCOORD0–TEXCOORD7, TEX0–TEX7</td>
<td>Output texture coordinates</td>
</tr>
<tr>
<td>CLP0–CL5</td>
<td>Output Clip distances</td>
</tr>
</tbody>
</table>

STANDARD LIBRARY ISSUES

This profile is limited to standard library support available in HLSL11 for vertex shaders. In general, the Cg and HLSL11 standard libraries are very similar.

SEE ALSO

*ps_5_0, hs_5_0, gs_5_0, ds_5_0*
4.1 cgAddStateEnumerant

NAME
cgAddStateEnumerant - associates an integer enumerant value as a possible value for a state

SYNOPSIS

#include <Cg/cg.h>

void cgAddStateEnumerant( CGstate state, const char * name, int value );

PARAMETERS

state
The state to which to associate the name and value.

name
The name of the enumerant.

value
The value of the enumerant.

RETURN VALUES

None.

DESCRIPTION

cgAddStateEnumerant associates a given named integer enumerant value with a state definition. When that state is later used in a pass in an effect file, the value of the state assignment can optionally be given by providing a named enumerant defined with cgAddStateEnumerant. The state assignment will then take on the value provided when the enumerant was defined.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.
HISTORY

cgAddStateEnumerant was introduced in Cg 1.4.

SEE ALSO

cgCreateState, cgCreateArrayState, cgCreateSamplerState, cgCreateArraySamplerState, cgGetName

4.2 cgCallStateResetCallback

NAME

cgCallStateResetCallback - calls the state resetting callback function for a state assignment

SYNOPSIS

#include <Cg/cg.h>

CGbool cgCallStateResetCallback( CGstateassignment sa );

PARAMETERS

sa

The state assignment handle.

RETURN VALUES

Returns the boolean value returned by the callback function. It should be CG_TRUE upon success.

Returns CG_TRUE if no callback function was defined.

DESCRIPTION

cgCallStateResetCallback calls the graphics state resetting callback function for the given state assignment.

The semantics of “resetting state” will depend on the particular graphics state manager that defined the valid state assignments; it will generally either mean that graphics state is reset to what it was before the pass, or that it is reset to the default value. The OpenGL state manager in the OpenGL Cg runtime implements the latter approach.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgCallStateResetCallback was introduced in Cg 1.4.

SEE ALSO

cgResetPassState, cgSetStateCallbacks, cgCallStateSetCallback, cgCallStateValidateCallback
### 4.3 cgCallStateSetCallback

**NAME**

cgCallStateSetCallback - calls the state setting callback function for a state assignment

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGbool cgCallStateSetCallback( CGstateassignment sa );
```

**PARAMETERS**

- `sa` - The state assignment handle.

**RETURN VALUES**

Returns the boolean value returned by the callback function. It should be `CG_TRUE` upon success.

Returns `CG_TRUE` if no callback function was defined.

**DESCRIPTION**

cgCallStateSetCallback calls the graphics state setting callback function for the given state assignment.

**EXAMPLES**

*to-be-written*

**ERRORS**

`CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.

**HISTORY**

cgCallStateSetCallback was introduced in Cg 1.4.

**SEE ALSO**

cgSetPassState, cgSetStateCallbacks, cgCallStateResetCallback, cgCallStateValidateCallback

### 4.4 cgCallStateValidateCallback

**NAME**

cgCallStateValidateCallback - calls the state validation callback function for a state assignment

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGbool cgCallStateValidateCallback( CGstateassignment sa );
```
PARAMETERS

sa
The state assignment handle.

RETURN VALUES

Returns the boolean value returned by the validation function. It should be CG_TRUE upon success.
Returns CG_TRUE if no callback function was defined.

DESCRIPTION

cgCallStateValidateCallback calls the state validation callback function for the given state assignment. The validation callback will return CG_TRUE or CG_FALSE depending on whether the current hardware and driver support the graphics state set by the state assignment.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgCallStateValidateCallback was introduced in Cg 1.4.

SEE ALSO

cgSetStateCallbacks, cgCallStateSetCallback, cgCallStateResetCallback

4.5 cgCombinePrograms

NAME

cgCombinePrograms - combine programs from different domains

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCombinePrograms( int n,
                                          const CGprogram * exeList );

PARAMETERS

n
The number of program objects in exeList.
exeList
An array of two or more executable programs, each from a different domain.

RETURN VALUES

Returns a handle to the newly created program on success.
Returns NULL if an error occurs.
DESCRIPTION

cgCombinePrograms will take a set of n programs and combine them into a single CGprogram. This allows a single call to BindProgram (instead of a BindProgram for each individual program) and provides optimizations between the combined set of program inputs and outputs.

EXAMPLES

CGprogram p1 = cgCreateProgram(context, CG_SOURCE, vSrc, vProfile, vEntryName, NULL);
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NULL);

CGprogram programs[] = {p1, p2};
CGprogram combined = cgCombinePrograms(2, programs);

cgDestroyProgram(p1);
cgDestroyProgram(p2);

cgGLBindProgram(combined); /* Assuming cgGL runtime */
/* Render... */

ERRORS

CG_INVALID_DIMENSION_ERROR is generated if n less than or equal to 1 or n is greater than 3.
CG_INVALID_PARAMETER_ERROR is generated if exeList is NULL.
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if one of the programs in exeList is invalid.
The errors listed in cgCreateProgram might also be generated.

HISTORY

cgCombinePrograms was introduced in Cg 1.5.

SEE ALSO

cgCombinePrograms2, cgCombinePrograms3, cgCombinePrograms4, cgCombinePrograms5, cgCreateProgram, cgGLBindProgram, cgD3D9BindProgram

4.6 cgCombinePrograms2

NAME

cgCombinePrograms2 - combine programs from two different domains

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCombinePrograms2( const CGprogram program1,
                                const CGprogram program2 );

PARAMETERS

    program1
An executable program from one domain.

program2

An executable program from a different domain.

RETURN VALUES

Returns a handle to the newly created program on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCombinePrograms2 takes two programs from different domains and combines them into a single CGprogram. This is a convenience function for cgCombinePrograms.

EXAMPLES

CGprogram p1 = cgCreateProgram(context, CG_SOURCE, vSrc, vProfile, vEntryName, NULL);
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NULL);

CGprogram combined = cgCombinePrograms2(p1, p2);

cgDestroyProgram(p1);
cgDestroyProgram(p2);

cgGLBindProgram(combined); /* Assuming cgGL runtime */
/* Render... */

ERRORS

The errors listed in cgCombinePrograms might be generated.

HISTORY

cgCombinePrograms2 was introduced in Cg 1.5.

SEE ALSO

cgCombinePrograms, cgCombinePrograms3, cgCombinePrograms4, cgCombinePrograms5

4.7 cgCombinePrograms3

NAME

cgCombinePrograms3 - combine programs from three different domains

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCombinePrograms3( const CGprogram program1,
                                const CGprogram program2,
                                const CGprogram program3 );
PARAMETERS

program1
An executable program from one domain.

program2
An executable program from a second domain.

program3
An executable program from a third domain.

RETURN VALUES

Returns a handle to the newly created program on success.
Returns NULL if an error occurs.

DESCRIPTION

cgCombinePrograms3 takes three programs from different domains and combines them into a single CGprogram. This is a convenience function for cgCombinePrograms.

EXAMPLES

CGprogram p1 = cgCreateProgram(context, CG_SOURCE, vSrc, vProfile, vEntryName, NULL);
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NULL);
CGprogram p3 = cgCreateProgram(context, CG_SOURCE, gSrc, gProfile, gEntryName, NULL);

CGprogram combined = cgCombinePrograms3(p1, p2, p3);
cgDestroyProgram(p1);
cgDestroyProgram(p2);
cgDestroyProgram(p3);

cgGLBindProgram(combined); /* Assuming cgGL runtime */
/* Render... */

ERRORS

The errors listed in cgCombinePrograms might be generated.

HISTORY

cgCombinePrograms3 was introduced in Cg 1.5.

SEE ALSO

cgCombinePrograms, cgCombinePrograms2, cgCombinePrograms4, cgCombinePrograms5

4.8 cgCombinePrograms4

NAME

cgCombinePrograms4 - combine programs from three different domains
SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCombinePrograms4( const CGprogram program1, const CGprogram program2, const CGprogram program3, const CGprogram program4 );

PARAMETERS

program1
An executable program from one domain.

program2
An executable program from a second domain.

program3
An executable program from a third domain.

program4
An executable program from a fourth domain.

RETURN VALUES

Returns a handle to the newly created program on success.

 Returns NULL if an error occurs.

DESCRIPTION

cgCombinePrograms4 takes four programs from different domains and combines them into a single CGprogram. This is a convenience function for cgCombinePrograms.

EXAMPLES

CGprogram p1 = cgCreateProgram(context, CG_SOURCE, vSrc, vProfile, vEntryName, NULL);
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NULL);
CGprogram p3 = cgCreateProgram(context, CG_SOURCE, tcSrc, tcProfile, tcEntryName, NULL);
CGprogram p4 = cgCreateProgram(context, CG_SOURCE, teSrc, teProfile, teEntryName, NULL);

CGprogram combined = cgCombinePrograms4(p1, p2, p3, p4);

cgDestroyProgram(p1);
cgDestroyProgram(p2);
cgDestroyProgram(p3);
cgDestroyProgram(p4);

cgGLBindProgram(combined); /* Assuming cgGL runtime */

/* Render... */

ERRORS

The errors listed in cgCombinePrograms might be generated.
HISTORY

cgCombinePrograms4 was introduced in Cg 3.0.

SEE ALSO

cgCombinePrograms, cgCombinePrograms2, cgCombinePrograms3, cgCombinePrograms5

4.9 cgCombinePrograms5

NAME

cgCombinePrograms5 - combine programs from three different domains

SYNOPSIS

```c
#include <Cg/cg.h>

CGprogram cgCombinePrograms5(
    const CGprogram program1,
    const CGprogram program2,
    const CGprogram program3,
    const CGprogram program4,
    const CGprogram program5 );
```

PARAMETERS

- **program1**
  An executable program from one domain.

- **program2**
  An executable program from a second domain.

- **program3**
  An executable program from a third domain.

- **program4**
  An executable program from a fourth domain.

- **program5**
  An executable program from a fifth domain.

RETURN VALUES

Returns a handle to the newly created program on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCombinePrograms5 takes five programs from different domains and combines them into a single CGprogram. This is a convenience function for cgCombinePrograms.

EXAMPLES
CGprogram p1 = cgCreateProgram(context, CG_SOURCE, vSrc, vProfile, vEntryName, NULL);
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NULL);
CGprogram p3 = cgCreateProgram(context, CG_SOURCE, gSrc, gProfile, gEntryName, NULL);
CGprogram p4 = cgCreateProgram(context, CG_SOURCE, tcSrc, tcProfile, tcEntryName, NULL);
CGprogram p5 = cgCreateProgram(context, CG_SOURCE, teSrc, teProfile, teEntryName, NULL);

CGprogram combined = cgCombinePrograms5(p1, p2, p3, p4, p5);

cgDestroyProgram(p1);
cgDestroyProgram(p2);
cgDestroyProgram(p3);
cgDestroyProgram(p4);
cgDestroyProgram(p5);

cgGLBindProgram(combined); /* Assuming cgGL runtime */

/* Render... */

ERRORS

The errors listed in cgCombinePrograms might be generated.

HISTORY

cgCombinePrograms5 was introduced in Cg 3.0.

SEE ALSO

cgCombinePrograms, cgCombinePrograms2, cgCombinePrograms3, cgCombinePrograms4

4.10 cgCompileProgram

NAME

cgCompileProgram - compile a program object

SYNOPSIS

#include <Cg/cg.h>

void cgCompileProgram( CGprogram program );

PARAMETERS

program

The program object to compile.

RETURN VALUES

None.
DESCRIPTION

cgCompileProgram compiles the specified Cg program for its target profile. A program must be compiled before it can be loaded (by the API-specific part of the runtime). It must also be compiled before its parameters can be inspected.

The compiled program can be retrieved as a text string by passing CG_COMPILED_PROGRAM to cgGetProgramString.

Certain actions invalidate a compiled program and the current value of all of its parameters. If one of these actions is performed, the program must be recompiled before it can be used. A program is invalidated if the program source is modified, if the compile arguments are modified, or if the entry point is changed.

If one of the parameter bindings for a program is changed, that action invalidates the compiled program, but does not invalidate the current value of the program’s parameters.

EXAMPLES

```c
if(!cgIsProgramCompiled(program))
    cgCompileProgram(program);
```

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_COMPILER_ERROR is generated if compilation fails.

HISTORY

cgCompileProgram was introduced in Cg 1.1.

SEE ALSO

cgIsProgramCompiled, cgCreateProgram, cgGetNextParameter, cgIsParameter, cgGetProgramString

4.11 cgConnectParameter

NAME

cgConnectParameter - connect two parameters

SYNOPSIS

```c
#include <Cg/cg.h>

void cgConnectParameter( CGparameter from,
                        CGparameter to );
```

PARAMETERS

from

    The source parameter.

to

    The destination parameter.

RETURN VALUES

None.
DESCRIPTION

cgConnectParameter connects a source (from) parameter to a destination (to) parameter. The resulting connection forces
the value and variability of the destination parameter to be identical to the source parameter. A source parameter
may be connected to multiple destination parameters but there may only be one source parameter per destination
parameter.

cgConnectParameter may be used to create an arbitrarily deep tree. A runtime error will be thrown
if a cycle is inadvertently created. For example, the following code snipped would generate a
CG_BIND_CREATES_CYCLE_ERROR:

```c
CGcontext context = cgCreateContext();
CGparameter Param1 = cgCreateParameter(context, CG_FLOAT);
CGparameter Param2 = cgCreateParameter(context, CG_FLOAT);
CGparameter Param3 = cgCreateParameter(context, CG_FLOAT);

cgConnectParameter(Param1, Param2);
cgConnectParameter(Param2, Param3);
cgConnectParameter(Param3, Param1); /* This will generate the error */
```

If the source type is a complex type (e.g., struct, or array) the topology and member types of both parameters must be
identical. Each correlating member parameter will be connected.

Both parameters must be of the same type unless the source parameter is a struct type, the destination parameter is an
interface type, and the struct type implements the interface type. In such a case, a copy of the parameter tree under the
source parameter will be duplicated, linked to the original tree, and placed under the destination parameter.

If an array parameter is connected to a resizable array parameter the destination parameter array will automatically be
resized to match the source array.

The source parameter may not be a program parameter. Also the variability of the parameters may not be
CG_VARYING.

EXAMPLES

```c
CGparameter TimeParam1 = cgGetNamedParameter(program1, "time");
CGparameter TimeParam2 = cgGetNamedParameter(program2, "time");
CGparameter SharedTime = cgCreateParameter(context,
  cgGetParameterType(TimeParam1));

cgConnectParameter(SharedTime, TimeParam1);
cgConnectParameter(SharedTime, TimeParam2);

cgSetParameter1f(SharedTime, 2.0);
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if either of the from or to parameters are invalid handles.

CG_PARAMETER_IS_NOT_SHARED_ERROR is generated if the source parameter is a program parameter.

CG_BIND_CREATES_CYCLE_ERROR is generated if the connection will result in a cycle.

CG_PARAMETERS_DO_NOT_MATCH_ERROR is generated if the parameters do not have the same type or the
topologies do not match.

CG_ARRAY_TYPES_DO_NOT_MATCH_ERROR is generated if the type of two arrays being connected do not
match.

CG_ARRAY_DIMENSIONS_DO_NOT_MATCH_ERROR is generated if the dimensions of two arrays being
connected do not match.
HISTORY
cgConnectParameter was introduced in Cg 1.2.

SEE ALSO
cgGetConnectedParameter, cgGetConnectedToParameter, cgDisconnectParameter

4.12 cgCopyEffect

NAME
cgCopyEffect - make a copy of an effect

SYNOPSIS
#include <Cg/cg.h>

CGeffect cgCopyEffect( CGeffect effect );

PARAMETERS

  effect
    The effect object to be copied.

RETURN VALUES

Returns a copy of effect on success.
Returns NULL if effect is invalid or the copy fails.

DESCRIPTION

cgCopyEffect creates a new effect object that is a copy of effect and adds it to the same context as effect.
Note: Currently cgCopyEffect does not work and therefore will always returns NULL.

EXAMPLES

CGeffect effectCopy = cgCopyEffect(effect);

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY
cgCopyEffect was introduced in Cg 2.0.

SEE ALSO
cgCreateEffect, cgCreateEffectFromFile, cgDestroyEffect
4.13 cgCopyProgram

NAME
cgCopyProgram - make a copy of a program object

SYNOPSIS
#include <Cg/cg.h>

CGprogram cgCopyProgram( CGprogram program );

PARAMETERS

program
The program object to copy.

RETURN VALUES

Returns a copy of program on success.
Returns NULL if program is invalid or the copy fails.

DESCRIPTION

cgCopyProgram creates a new program object that is a copy of program and adds it to the same context as program. cgCopyProgram is useful for creating a new instance of a program whose parameter properties have been modified by the run-time API.

EXAMPLES

CGprogram programCopy = cgCopyProgram(program);

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgCopyProgram was introduced in Cg 1.1.
cgCopyProgram is operational as of Cg 3.0.

SEE ALSO

cgCreateProgram, cgDestroyProgram

4.14 cgCreateArraySamplerState

NAME

cgCreateArraySamplerState - create an array-typed sampler state definition

SYNOPSIS
#include <Cg/cg.h>

CGstate cgCreateArraySamplerState( CGcontext context,
    const char * name,
    CGtype type,
    int nelements );

PARAMETERS

c context
The context in which to define the sampler state.

c name
The name of the new sampler state.

c type
The type of the new sampler state.

c nelements
The number of elements in the array.

RETURN VALUES

Returns a handle to the newly created CGstate.

Returns NULL if there is an error.

DESCRIPTION

cgCreateArraySamplerState adds a new array-typed sampler state definition to context. All state in sampler_state blocks must have been defined ahead of time via a call to cgCreateSamplerState or cgCreateArraySamplerState before adding an effect file to the context.

Applications will typically call cgSetStateCallbacks shortly after creating a new state with cgCreateArraySamplerState.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_PARAMETER_ERROR is generated if name is NULL or not a valid identifier, if type is not a simple scalar, vector, or matrix-type, or if nelements is not a positive number.

HISTORY

cgCreateArraySamplerState was introduced in Cg 1.4.

SEE ALSO

cgCreateSamplerState, cgGetName, cgGetType, cgIsState, cgSetStateCallbacks, cgRegisterStates
4.15 cgCreateArrayState

NAME

cgCreateArrayState - create an array-typed state definition

SYNOPSIS

#include <Cg/cg.h>

CGstate cgCreateArrayState( CGcontext context,
                             const char * name,
                             CGtype type,
                             int nelements );

PARAMETERS

class INSIDECLASS 


context
        The context in which to define the state.

name
        The name of the new state.

type
        The type of the new state.

nelements
        The number of elements in the array.

RETURN VALUES

Returns a handle to the newly created CGstate.
Returns NULL if there is an error.

DESCRIPTION

cgCreateArrayState adds a new array-typed state definition to context. Before a CgFX file is added to a context, all
state assignments in the file must have previously been defined via a call to cgCreateState or cgCreateArrayState.
Applications will typically call cgSetStateCallbacks shortly after creating a new state with cgCreateArrayState.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_PARAMETER_ERROR is generated if name is NULL or not a valid identifier, if type is not a
simple scalar, vector, or matrix-type, or if nelements is not a positive number.

HISTORY

cgCreateArrayState was introduced in Cg 1.4.

SEE ALSO

cgGetStateContext, cgGetStateName, cgGetStateType, cgIsState, cgSetStateCallbacks, cgGLRegisterStates
4.16 cgCreateBuffer

NAME

cgCreateBuffer - create a buffer object managed by the runtime

SYNOPSIS

#include <Cg/cg.h>

CGbuffer cgCreateBuffer( CGcontext context,
                         int size,
                         const void *data,
                         CGbufferusage bufferUsage );

PARAMETERS

context
The context to which the new buffer will be added.

size
The length in bytes of the buffer to create.

data
Pointer to initial buffer data. NULL will fill the buffer with zero.

bufferUsage
Indicates the intended usage method of the buffer.

Can be one of the following types:

* CG_BUFFER_USAGE_STREAM_DRAW
* CG_BUFFER_USAGE_STREAM_READ
* CG_BUFFER_USAGE_STREAM_COPY
* CG_BUFFER_USAGE_STATIC_DRAW
* CG_BUFFER_USAGE_STATIC_READ
* CG_BUFFER_USAGE_STATIC_COPY
* CG_BUFFER_USAGE_DYNAMIC_DRAW
* CG_BUFFER_USAGE_DYNAMIC_READ
* CG_BUFFER_USAGE_DYNAMIC_COPY

RETURN VALUES

Returns a CGbuffer handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateBuffer creates a runtime managed Cg buffer object.

There is no way to query the 3D API-specific resource for a managed buffer. cgGLCreateBuffer should be used if the application wishes to later query the 3D API-specific resource for the buffer.
EXAMPLES

CGbuffer myBuffer = cgCreateBuffer( myCgContext, sizeof( float ) * 16, initialData, CG_BUFFER_USAGE_STATIC_DRAW );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgCreateBuffer was introduced in Cg 2.0.

SEE ALSO

cgGLCreateBuffer, cgDestroyBuffer

4.17 cgCreateContext

NAME
cgCreateContext - create a context

SYNOPSIS

#include <Cg/cg.h>

CGcontext cgCreateContext( void );

PARAMETERS

None.

RETURN VALUES

Returns a valid CGcontext on success.
Returns NULL if context creation fails.

DESCRIPTION

cgCreateContext creates a Cg context object and returns its handle. A Cg context is a container for Cg programs. All Cg programs must be added to a Cg context.

EXAMPLES

CGcontext context = cgCreateContext();

ERRORS

CG_MEMORY_ALLOC_ERROR is generated if a context couldn’t be created.

HISTORY

cgCreateContext was introduced in Cg 1.1.

SEE ALSO

cgDestroyContext, cgSetContextBehavior, cgGetContextBehavior, cgCreateProgram, cgCreateEffect, cgCreateState
4.18 cgCreateEffect

NAME

cgCreateEffect - create an effect object from a source string

SYNOPSIS

#include <Cg/cg.h>

CGeffect cgCreateEffect( CGcontext context,
                          const char * source,
                          const char ** args );

PARAMETERS

context
The context to which the new effect will be added.

source
A string containing the effect’s source code.

args
If args is not NULL it is assumed to be an array of null-terminated strings that will be passed directly to
the compiler as arguments. The last value of the array must be a NULL.

RETURN VALUES

Returns a CGeffect handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateEffect generates a new CGeffect object and adds it to the specified Cg context.

If an error occurs cgGetLastListing can be called to retrieve any warning or error messages from the compilation
process.

EXAMPLES

Creating an effect:

char *effectSource = ...;
CGcontext context = cgCreateContext();
CGeffect effect = cgCreateEffect(context,
                                   effectSource,
                                   NULL);

Iterating through the techniques in an effect:

CGtechnique technique = cgGetFirstTechnique(effect);
while (technique) {
    // Do something with each technique
    technique = cgGetNextTechnique(technique);
}
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_COMPILER_ERROR is generated if compilation fails.

HISTORY

cgCreateEffect was introduced in Cg 1.4.

SEE ALSO

cgCreateContext, cgCreateEffectFromFile, cgGetLastListing, cgGetFirstTechnique, cgGetTechniqueEffect, cgGetFirstEffect

4.19 cgCreateEffectAnnotation

NAME

cgCreateEffectAnnotation - create an effect annotation

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgCreateEffectAnnotation( CGeffect effect,
                                           const char * name,
                                           CGtype type );

PARAMETERS

effect
   The effect to which the new annotation will be added.

name
   The name of the new annotation.

type
   The type of the new annotation.

RETURN VALUES

Returns the new CGannotation handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateEffectAnnotation adds a new annotation to the effect.

EXAMPLES

/* create a float annotation named "Apple" for CGeffect effect */
CGannotation ann = cgCreateEffectAnnotation( effect, "Apple", CG_FLOAT );
ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

CG_DUPLICATE_NAME_ERROR is generated if name is already used by an annotation for this effect.

CG_INVALID_ENUMERANT_ERROR is generated if type is not CG_INT, CG_FLOAT, CG_BOOL, or CG_STRING.

HISTORY

cgCreateEffectAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetNamedEffectAnnotation, cgGetFirstEffectAnnotation, cgGetNextAnnotation

4.20 cgCreateEffectFromFile

NAME

cgCreateEffectFromFile - create an effect object from a file

SYNOPSIS

#include <Cg/cg.h>

CGeffect cgCreateEffectFromFile( CGcontext context,
const char * filename,
const char ** args );

PARAMETERS

ccontext

The context to which the new effect will be added.

filename

Name of a file that contains the effect’s source code.

args

If args is not NULL it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a NULL.

RETURN VALUES

Returns a CGeffect handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateEffectFromFile generates a new CGeffect object and adds it to the specified Cg context.

If an error occurs cgGetLastListing can be called to retrieve any warning or error messages from the compilation process.
EXAMPLES

CGcontext context = cgCreateContext();
CGeffect effect = cgCreateEffectFromFile(context, "filename.cgfx", NULL);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_FILE_READ_ERROR is generated if the given filename cannot be read.
CG_COMPILER_ERROR is generated if compilation fails.

HISTORY

cgCreateEffectFromFile was introduced in Cg 1.4.

SEE ALSO

cgCreateContext, cgCreateEffect, cgGetLastListing, cgGetTechniqueEffect, cgGetEffectName, cgSetEffectName,
cgCreateEffectAnnotation, cgDestroyEffect, cgGetFirstEffect

4.21 cgCreateEffectParameter

NAME

cgCreateEffectParameter - create a parameter in an effect

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgCreateEffectParameter( CGeffect effect,
                      const char * name,
                      CGtype type );

PARAMETERS

  effect
  The effect to which the new parameter will be added.

  name
  The name of the new parameter.

  type
  The type of the new parameter.

RETURN VALUES

Returns the handle to the new parameter.

DESCRIPTION

cgCreateEffectParameter adds a new parameter to the specified effect.
EXAMPLES

CGeffect effect = cgCreateEffect( ... );
CGparameter param = cgCreateEffectParameter( effect, "myFloatParam", CG_FLOAT );

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.
CG_INVALID_VALUE_TYPE_ERROR is generated if type is invalid.

HISTORY

cgCreateEffectParameter was introduced in Cg 1.5.

SEE ALSO

cgIsParameter, cgCreateEffectParameterArray, cgCreateEffectParameterMultiDimArray, cgCreateTechnique, cgCreatePass, cgConnectParameter

4.22 cgCreateEffectParameterArray

NAME

cgCreateEffectParameterArray - create an array parameter in an effect

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgCreateEffectParameterArray( CGeffect effect,
                                         const char * name,
                                         CGtype type,
                                         int length );

PARAMETERS

    effect
    The effect to which the new parameter will be added.

    name
    The name of the new parameter.

    type
    The type of the new parameter.

    length
    The size of the array.

RETURN VALUES

Returns the handle to the new array parameter on success.
Returns NULL if an error occurs.

DESCRIPTION

cgCreateEffectParameterArray adds a new array parameter to the specified effect.
EXAMPLES

CGeffect effect = cgCreateEffect( ... );
CGparameter array = cgCreateEffectParameterArray( effect, "myFloatArray", CG_FLOAT, 2 );

ERRORS

CG_INV A Li D_EFF ECT_HANDLE_ERROR is generated if effect is not a valid effect.
CG_INV A Li D_VALUE_TYPE_ERROR is generated if type is invalid.

HISTORY

cgCreateEffectParameterArray was introduced in Cg 1.5.

SEE ALSO

cgCreateEffectParameter, cgCreateEffectParameterMultiDimArray

4.23 cgCreateEffectParameterMultiDimArray

NAME

cgCreateEffectParameterMultiDimArray - create a multi-dimensional array in an effect

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgCreateEffectParameterMultiDimArray( CGeffect effect,
const char * name,
CGtype type,
int dim,
const int * lengths );

PARAMETERS

effect
The effect to which the new parameter will be added.
name
The name of the new parameter.
type
The type of the new parameter.
dim
The dimension of the array.
lengths
The sizes for each dimension of the array.
RETURN VALUES

Returns the handle of the new parameter on success.
Returns NULL if an error occurs.

DESCRIPTION

cgCreateEffectParameterMultiDimArray adds a new multidimensional array parameter to the specified effect.

EXAMPLES

CGEffect effect = cgCreateEffect( ... );
int lengths[] = {2,2};
CGparameter array = cgCreateEffectParameterMultiDimArray(effect,  
               "myFloatMultiArray", CG_FLOAT, 2, lengths);

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.
CG_INVALID_VALUE_TYPE_ERROR is generated if type is invalid.

HISTORY

cgCreateEffectParameterMultiDimArray was introduced in Cg 1.5.

SEE ALSO

cgCreateEffectParameter, cgCreateEffectParameterArray

4.24 cgCreateObj

NAME

cgCreateObj - create a cg object type from a shader string

SYNOPSIS

#include <Cg/cg.h>

CGobj cgCreateObj( CGcontext context,  
    CGenum program_type,  
    const char * source,  
    CGprofile profile,  
    const char ** args );

PARAMETERS

context
    The context to which the new object will be added.

program_type
    An enumerant describing the contents of the source string. The following enumerants are allowed:

    CG_SOURCE
        source contains Cg source code.
**CG_OBJECT**

*source* contains object code that resulted from the precompilation of some Cg source code.

A string containing either the programs source or object code. See *program_type* for more information.

*profile*

The profile enumerant for the program.

*args*

If *args* is not **NULL** it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a **NULL**.

**RETURN VALUES**

Returns a **CGobj** handle on success.

Returns **NULL** if an error occurs.

**DESCRIPTION**

cgCreateObj creates a new **CGobj** which is a source code object similar to a .obj or .o in C/C++ programming where various forms of data can be extracted. This can be used, for example, to create user defined data types from a Cg source string.

**EXAMPLES**

// Imagine a Cg source string that contains:

```c
const char src[] =
"typedef struct {
    float3 param1; \
    half4  param2; \\
} MyType;";
```

// To create a Cg obj:

```c
CGcontext ctx = cgCreateContext();
CGobj structObj = cgCreateObj(ctx, CG_SOURCE, src, CG_PROFILE_ARBVP1, NULL);
```

// Now we can get the CGtype:

```c
CGtype userDefinedMyType = cgGetNamedUserType(structObj, "MyType");
```

// We could also iterate through all the types in the CGobj printing their names like this:

```c
int numTypes = cgGetNumUserTypes(structObj);
for (int i=0; i<numTypes; i++) {
    cout << cgGetTypeString(cgGetUserType(structObj, i)) << endl;
}
```

**ERRORS**

**CG_INVALID_CONTEXT_HANDLE_ERROR** is generated if *context* is not a valid context.

**CG_INVALID_ENUMERANT_ERROR** is generated if *program_type* is not **CG_SOURCE** or **CG_OBJECT**.

**CG_UNKNOWN_PROFILE_ERROR** is generated if *profile* is not a supported profile.
**CG_COMPILER_ERROR** is generated if compilation fails.

**HISTORY**

cgCreateObj was introduced in Cg 2.0.

**SEE ALSO**

cgCreateObjFromFile, cgDestroyObj

### 4.25 cgCreateObjFromFile

**NAME**

cgCreateObjFromFile - create a cg object type from a shader file

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGobj cgCreateObjFromFile( CGcontext context,
                           CGenum program_type,
                           const char * source_file,
                           CGprofile profile,
                           const char ** args );
```

**PARAMETERS**

- **context**
  
The context to which the new object will be added.

- **program_type**
  
  An enumerant describing the contents of the **source** string. The following enumerants are allowed:

  - **CG_SOURCE**
    
    **source** contains Cg source code.

  - **CG_OBJECT**
    
    **source** contains object code that resulted from the precompilation of some Cg source code.

- **source_file**
  
  Name of a file containing source or object code. See **program_type** for more information.

- **profile**
  
  The profile enumerant for the program.

- **args**
  
  If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a **NULL**.

**RETURN VALUES**

Returns a **CGobj** handle on success.

Returns **NULL** if an error occurs.
DESCRIPTION

cgCreateObjFromFile creates a new CGobj which is a source code object similar to a .obj or .o in C/C++ programming where various forms of data can be extracted. This can be used, for example, to create user defined data types from a Cg source string.

EXAMPLES

// To create a Cg obj:

CGcontext ctx = cgCreateContext();
CGobj structObj = cgCreateObjFromFile(ctx, CG_SOURCE, source_file,
                                        CG_PROFILE_ARBVP1, NULL);

// Now we can get the CGtype:

CGtype userDefinedMyType = cgGetNamedUserType(structObj, "MyType");

// We could also iterate through all the types in the CGobj printing
// their names like this:

int numTypes = cgGetNumUserTypes(structObj);
for (int i=0; i<numTypes; i++) {
    cout << cgGetTypeString(cgGetUserType(structObj, i)) << endl;
}

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_INVALID_ENUMERANT_ERROR is generated if program_type is not CG_SOURCE or CG_OBJECT.
CG_UNKNOWN_PROFILE_ERROR is generated if profile is not a supported profile.
CG_COMPILER_ERROR is generated if compilation fails.

HISTORY

cgCreateObjFromFile was introduced in Cg 2.0.

SEE ALSO

cgCreateObj, cgDestroyObj

4.26 cgCreateParameter

NAME

cgCreateParameter - create a parameter

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgCreateParameter( CGcontext context,
                               CGtype type );
PARAMETERS

context
The context to which the new parameter will be added.

type
The type of the new parameter.

RETURN VALUES

Returns the handle to the new parameter.

DESCRIPTION

cgCreateParameter creates context level shared parameters. These parameters are primarily used by connecting them to one or more program parameters with cgConnectParameter.

EXAMPLES

CGcontext context = cgCreateContext();
CGparameter param = cgCreateParameter(context, CG_FLOAT);

ERRORS

CG_INVALID_VALUE_TYPE_ERROR is generated if type is invalid.
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgCreateParameter was introduced in Cg 1.2.

SEE ALSO

cgCreateParameterArray, cgCreateParameterMultiDimArray, cgCreateEffectParameter, cgDestroyParameter, cgConnectParameter

4.27 cgCreateParameterAnnotation

NAME

cgCreateParameterAnnotation - create an annotation in a parameter

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgCreateParameterAnnotation( CGparameter param, const char * name, CGtype type );

PARAMETERS

parm
The parameter to which the new annotation will be added.

name
The name of the new annotation.

type
The type of the new annotation.

**RETURN VALUES**

Returns the new **CGannotation** handle on success.

Returns **NULL** if an error occurs.

**DESCRIPTION**

cgCreateParameterAnnotation adds a new annotation to the specified parameter.

**EXAMPLES**

```c
CGannotation ann = cgCreateParameterAnnotation( param, "Apple", CG_FLOAT );
```

**ERRORS**

**CG_INVALID_PARAM_HANDLE_ERROR** is generated if **param** is not a valid parameter.

**CG_DUPLICATE_NAME_ERROR** is generated if **name** is already used by an annotation for this parameter.

**CG_INVALID_ENUMERANT_ERROR** is generated if **type** is not **CG_INT**, **CG_FLOAT**, **CG_BOOL**, or **CG_STRING**.

**HISTORY**

cgCreateParameterAnnotation was introduced in Cg 1.5.

**SEE ALSO**

cgGetNamedParameterAnnotation, cgGetFirstParameterAnnotation, cgGetNextAnnotation

### 4.28 cgCreateParameterArray

**NAME**

cgCreateParameterArray - creates a parameter array

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgCreateParameterArray( CGcontext context,
                                    CGtype type,
                                    int length );
```

**PARAMETERS**

context
The context to which the new parameter will be added.

type
The type of the new parameter.

length
The length of the array being created.

**RETURN VALUES**

Returns the handle to the new parameter array.

**DESCRIPTION**

`cgCreateParameterArray` creates context level shared parameter arrays. These parameters are primarily used by connecting them to one or more program parameter arrays with `cgConnectParameter`.

`cgCreateParameterArray` works similarly to `cgCreateParameter`, but creates an array of parameters rather than a single parameter.

**EXAMPLES**

```c
CGcontext context = cgCreateContext();
CGparameter param = cgCreateParameterArray(context, CG_FLOAT, 5);
```

**ERRORS**

`CG_INVALID_VALUE_TYPE_ERROR` is generated if `type` is invalid.

`CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.

**HISTORY**

`cgCreateParameterArray` was introduced in Cg 1.2.

**SEE ALSO**

`cgCreateParameter`, `cgCreateParameterMultiDimArray`, `cgDestroyParameter`

### 4.29 `cgCreateParameterMultiDimArray`

**NAME**

`cgCreateParameterMultiDimArray` - creates a multi-dimensional parameter array

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgCreateParameterMultiDimArray( CGcontext context,
                                          CGtype type,
                                          int dim,
                                          const int * lengths );
```

**PARAMETERS**

- **context**
  The context to which the new parameter will be added.
- **type**
  The type of the new parameter.
- **dim**
  The dimension of the multi-dimensional array.
lengths

An array of length values, one for each dimension of the array to be created.

RETURN VALUES

Returns the handle to the new parameter array.

DESCRIPTION

cgCreateParameterMultiDimArray creates context level shared multi-dimensional parameter arrays. These parameters are primarily used by connecting them to one or more program parameter arrays with cgConnectParameter.

cgCreateParameterMultiDimArray works similarly to cgCreateParameterArray. Instead of taking a single length parameter it takes an array of lengths, one per dimension. The dimension of the array is defined by the dim parameter.

EXAMPLES

/\* Creates a three dimensional float array similar to */
/\* the C declaration : float param[5][3][4]; */

int lengths[] = { 5, 3, 4 };
CGcontext context = cgCreateContext();
CGparameter param = cgCreateParameterMultiDimArray(context, CG_FLOAT, 3, lengths);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_VALUE_TYPE_ERROR is generated if type is invalid.

HISTORY

cgCreateParameterMultiDimArray was introduced in Cg 1.2.

SEE ALSO

cgCreateParameter, cgCreateParameterArray, cgDestroyParameter, cgConnectParameter

4.30 cgCreatePass

NAME

cgCreatePass - create a pass in a technique

SYNOPSIS

#include <Cg/cg.h>

CGpass cgCreatePass( CGtechnique tech,
                      const char * name );

PARAMETERS

tech

The technique to which the new pass will be added.

name
The name of the new pass.

RETURN VALUES

Returns the handle to the new pass on success.
Returns NULL if an error occurs.

DESCRIPTION

cgCreatePass adds a new pass to the specified technique.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgCreatePass was introduced in Cg 1.5.

SEE ALSO

cgCreateTechnique

4.31 cgCreatePassAnnotation

NAME

cgCreatePassAnnotation - create an annotation in a pass

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgCreatePassAnnotation( CGpass pass,
                                      const char * name,
                                      CGtype type );

PARAMETERS

    pass
        The pass to which the new annotation will be added.
    name
        The name of the new annotation.
    type
        The type of the new annotation.

RETURN VALUES

Returns the new CGannotation handle on success.
Returns NULL if an error occurs.
DESCRIPTION

cgCreatePassAnnotation adds a new annotation to a pass.

EXAMPLES

/* create a float annotation named "Apple" for CGpass pass */
CGannotation ann = cgCreatePassAnnotation( pass, "Apple", CG_FLOAT );

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.
CG_DUPLICATE_NAME_ERROR is generated if name is already used by an annotation for this pass.
CG_INVALID_ENUMERANT_ERROR is generated if type is not CG_INT, CG_FLOAT, CG_BOOL, or CG_STRING.

HISTORY

cgCreatePassAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetNamedPassAnnotation, cgGetFirstPassAnnotation, cgGetNextAnnotation

4.32 cgCreateProgram

NAME

cgCreateProgram - create a program object from a string

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCreateProgram( CGcontext context, 
                      CGenum program_type, 
                      const char * program, 
                      CGprofile profile, 
                      const char * entry, 
                      const char ** args );

PARAMETERS

context
    The context to which the new program will be added.

program_type
    An enumerant describing the contents of the program string. The following enumerants are allowed:

        CG_SOURCE
    program contains Cg source code.

        CG_OBJECT
    program contains object code that resulted from the precompilation of some Cg source code.
program
A string containing either the programs source or object code. See program_type for more information.
profile
The profile enumerant for the program.
entry
The entry point to the program in the Cg source. If NULL, the entry point defaults to “main”.
args
If args is not NULL it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a NULL.

RETURN VALUES
Returns a CGprogram handle on success.
Returns NULL if an error occurs.

DESCRIPTION
cgCreateProgram generates a new CGprogram object and adds it to the specified Cg context.

EXAMPLES
CGcontext context = cgCreateContext();
CGprogram program = cgCreateProgram(context,
    CG_SOURCE,
    mysourcestring,
    CG_PROFILE_ARBVP1,
    "myshader",
    NULL);

ERRORS
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_INVALID_ENUMERANT_ERROR is generated if program_type is not CG_SOURCE or CG_OBJECT.
CG_UNKNOWN_PROFILE_ERROR is generated if profile is not a supported profile.
CG_COMPILER_ERROR is generated if compilation fails.

HISTORY
cgCreateProgram was introduced in Cg 1.1.

SEE ALSO

cgCreateContext, cgCreateProgramFromFile, cgDestroyProgram, cgGetProgramString

4.33 cgCreateProgramAnnotation

NAME
cgCreateProgramAnnotation - create an annotation in a program
SYNOPSIS

#include <Cg/cg.h>

CGannotation cgCreateProgramAnnotation( CGprogram program,
                                      const char * name,
                                      CGtype type );

PARAMETERS

program
The program to which the new annotation will be added.

name
The name of the new annotation.

type
The type of the new annotation.

RETURN VALUES

Returns the new CGannotation handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateProgramAnnotation adds a new annotation to a program.

EXAMPLES

/* create a float annotation named "Apple" for CGprogram prog */
CGannotation ann = cgCreateProgramAnnotation( prog, "Apple", CG_FLOAT );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_DUPLICATE_NAME_ERROR is generated if name is already used by an annotation for this program.

CG_INVALID_ENUMERANT_ERROR is generated if type is not CG_INT, CG_FLOAT, CG_BOOL, or CG_STRING.

HISTORY

cgCreateProgramAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetNamedProgramAnnotation, cgGetFirstProgramAnnotation, cgGetNextAnnotation

4.34 cgCreateProgramFromEffect

NAME

cgCreateProgramFromEffect - create a program object from an effect
SYNOPSIS

```c
#include <Cg/cg.h>

CGprogram cgCreateProgramFromEffect( CGeffect effect,
                                      CGprofile profile,
                                      const char * entry,
                                      const char ** args );
```

PARAMETERS

- **effect**
  The effect containing the program source code from which to create the program.

- **profile**
  The profile enumerant for the program.

- **entry**
  The entry point to the program in the Cg source. If **NULL**, the entry point defaults to “main”.

- **args**
  If **args** is not **NULL** it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a **NULL**.

RETURN VALUES

Returns a **CGprogram** handle on success.

Returns **NULL** if an error occurs.

DESCRIPTION

cgCreateProgramFromEffect generates a new **CGprogram** object and adds it to the effect’s Cg context.

EXAMPLES

to-be-written

ERRORS

**CG_INVALID_EFFECT_HANDLE_ERROR** is generated if **effect** is not a valid effect.

**CG_UNKNOWN_PROFILE_ERROR** is generated if **profile** is not a supported profile.

**CG_COMPILER_ERROR** is generated if compilation fails.

HISTORY

cgCreateProgramFromEffect was introduced in Cg 1.4.

SEE ALSO

cgCreateProgram, cgCreateProgramFromFile

4.35 cgCreateProgramFromFile

NAME

cgCreateProgramFromFile - create a program object from a file
SYNOPSIS

#include <Cg/cg.h>

CGprogram cgCreateProgramFromFile( CGcontext context,
const char * program_file,
CGprofile profile,
const char * entry,
const char ** args );

PARAMETERS

context
The context to which the new program will be added.

program_type
An enumerant describing the contents of the program_file. The following enumerants are allowed:

CG_SOURCE
program_file contains Cg source code.

CG_OBJECT
program_file contains object code that resulted from the precompilation of some Cg source code.

program_file
Name of a file containing source or object code. See program_type for more information.

profile
The profile enumerant for the program.

entry
The entry point to the program in the Cg source. If NULL, the entry point defaults to “main”.

args
If args is not NULL it is assumed to be an array of null-terminated strings that will be passed directly to the compiler as arguments. The last value of the array must be a NULL.

RETURN VALUES

Returns a CGprogram handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateProgramFromFile generates a new CGprogram object and adds it to the specified Cg context.

EXAMPLES

CGcontext context = cgCreateContext();
CGprogram program = cgCreateProgramFromFile(context,
CG_SOURCE,
mysourcefilename,
CG_PROFILE_ARBVP1,
"myshader",
    NULL);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_INVALID_ENUMERANT_ERROR is generated if program_type is not CG_SOURCE or CG_OBJECT.
CG_UNKNOWN_PROFILE_ERROR is generated if profile is not a supported profile.
CG_COMPILER_ERROR is generated if compilation fails.

HISTORY

cgCreateProgramFromFile was introduced in Cg 1.1.

SEE ALSO

cgCreateContext, cgCreateProgram, cgCreateProgramFromEffect, cgGetProgramString

4.36 cgCreateSamplerState

NAME

cgCreateSamplerState - create a sampler state definition

SYNOPSIS

#include <Cg/cg.h>

CGstate cgCreateSamplerState( CGcontext context,
                             const char * name,
                             CGtype type );

PARAMETERS

class
    The context in which to define the new sampler state.

name
    The name of the new sampler state.

type
    The type of the new sampler state.

RETURN VALUES

Returns a handle to the newly created CGstate.
Returns NULL if there is an error.

DESCRIPTION

cgCreateSamplerState adds a new sampler state definition to the context. When an effect file is added to the context,
all state in sampler_state blocks must have already been defined via a call to cgCreateSamplerState or cgCreateArraySamplerState.
Applications will typically call cgSetStateCallbacks shortly after creating a new state with cgCreateSamplerState.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_PARAMETER_ERROR is generated if name is NULL or not a valid identifier, or if type is not a simple scalar, vector, or matrix-type. Array-typed state should be created with cgCreateArrayState.

HISTORY

cgCreateSamplerState was introduced in Cg 1.4.

SEE ALSO

cgCreateArraySamplerState, cgGetName, cgGetType, cgIsState, cgCreateSamplerStateAssignment, cgGLRegisterStates

4.37 cgCreateSamplerStateAssignment

NAME

cgCreateSamplerStateAssignment - create a sampler state assignment

SYNOPSIS

#include <Cg.cg.h>

CGstateassignment cgCreateSamplerStateAssignment( CGparameter param,
                                           CGstate state );

PARAMETERS

param
   The sampler parameter to which the new state assignment will be associated.
state
   The state for which to create the new state assignment.

RETURN VALUES

Returns the handle to the created sampler state assignment.

DESCRIPTION

cgCreateSamplerStateAssignment creates a new state assignment for the given state and sampler parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.
HISTORY

cgCreateSamplerStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgCreateTechnique, cgCreateStateAssignment, cgCreateSamplerState

4.38 cgCreateState

NAME
cgCreateState - create a state definition

SYNOPSIS

#include <Cg/cg.h>

CGstate cgCreateState( CGcontext context, const char * name, CGtype type );

PARAMETERS

closext

The context in which to define the new state.

ame

The name of the new state.

type

The type of the new state.

RETURN VALUES

Returns a handle to the newly created CGstate.

Returns NULL if there is an error.

DESCRIPTION

cgCreateState adds a new state definition to the context. When a CgFX file is added to the context, all state assignments in the file must have already been defined via a call to cgCreateState or cgCreateArrayState.

Applications will typically call cgSetStateCallbacks shortly after creating a new state with cgCreateState.

EXAMPLES

Example callback functions for a state to register:

CGbool foo_set( CGstateassignment sa )
{
    int nVals = 0;
    const CGbool *val = cgGetBoolStateAssignmentValues( sa, &nVals );
    printf( "%nFooState set called with value %d\n", *val );
    return CG_TRUE;
}
CGbool foo_reset( CGstateassignment sa )
{
    printf( "\nFooState reset called.\n" );
    return CG_TRUE;
}

CGbool foo_validate( CGstateassignment sa )
{
    printf( "\nFooState validate called.\n" );
    return CG_TRUE;
}

Registering the state:

// Create and register new state FooState
CGstate fooState = cgCreateState( myCgContext, "FooState", CG_BOOL );
cgSetStateCallbacks( fooState, foo_set, foo_reset, foo_validate );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_PARAMETER_ERROR is generated if name is NULL or not a valid identifier, or if type is not a simple scalar, vector, or matrix-type. Array-typed state should be created with cgCreateArrayState.

HISTORY

cgCreateState was introduced in Cg 1.4.

SEE ALSO

cgCreateArrayState, cgGetStateContext, cgGetStateName, cgGetStateType, cgIsState, cgSetStateCallbacks, cgGLRegisterStates, cgD3D9RegisterStates, cgCreateContext

4.39 cgCreateStateAssignment

NAME

cgCreateStateAssignment - create a state assignment

SYNOPSIS

#include <Cg/cg.h>

CGstateassignment cgCreateStateAssignment( CGpass pass,
                                          CGstate state );

PARAMETERS

    pass
    The pass in which to create the state assignment.

    state
    The state used to create the state assignment.
RETURN VALUES

Returns the handle to the created state assignment.
Returns NULL if an error occurs.

DESCRIPTION

cgCreateStateAssignment creates a state assignment for the specified pass. The new state assignment is appended to the pass’ existing list of state assignments. If the state is actually a state array, the created state assignment is created for array index zero. Use cgCreateStateAssignmentIndex to create state assignments for other indices of an array state.

EXAMPLES

/* Procedurally create state assignment equivalent to */
/* "BlendFunc = { SrcAlpha, OneMinusSrcAlpha };" */
CGstate blendFuncState = cgGetNamedState(context, "BlendFunc");
CGstateassignment blendFuncSA =
    cgCreateStateAssignment(pass, blendFuncState);
static const int blendFuncConfig[2] =
    { GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA };
cgSetIntArrayStateAssignment(blendFuncSA, blendFuncConfig);

/* Procedurally create state assignment equivalent to */
/* "BlendEnable = true;" */
CGstate blendEnableState =
    cgGetNamedState(context, "BlendEnable");
CGstateassignment blendEnableSA =
    cgCreateStateAssignment(pass, blendEnableState);
cgSetBoolStateAssignment(blendEnableSA, CG_TRUE);

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.
CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgCreateStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgCreateTechnique, cgCreateSamplerStateAssignment, cgCreateState, cgCreateStateAssignmentIndex

4.40 cgCreateStateAssignmentIndex

NAME

cgCreateStateAssignmentIndex - create a state assignment for a state array

SYNOPSIS

#include <Cg/cg.h>

CGstateassignment cgCreateStateAssignmentIndex( CGpass pass,
                                                  CGstate state,
                                                  int index );
PARAMETERS

pass
The pass in which to create the state assignment.

state
The state array used to create the state assignment.

index
The index for the state array.

RETURN VALUES

Returns the new state assignment handle.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateStateAssignmentIndex creates a state assignment for the specified pass. The new state assignment is appended to the pass’s existing list of state assignments. The state assignment is for the given index of for the specified array state.

EXAMPLES

This example shows how to create a state assignment for enabling light 5:

```c
/* Procedurally create state assignment equivalent to */
/* "LightEnable[5] = 1;" */
CGstate lightEnableState = cgGetNamedState(context, "LightEnable");
CGstateassignment light5sa = cgCreateStateAssignmentIndex(pass, lightEnableState , 5);
.cgSetBoolStateAssignment(light5sa, CG_TRUE);
```

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

If the index is negative or index is greater than or equal the number of elements for the state array, no error is generated but NULL is returned.

HISTORY

cgCreateStateAssignmentIndex was introduced in Cg 1.5.

SEE ALSO

cgGetStateAssignmentIndex, cgCreateTechnique, cgCreateSamplerStateAssignment, cgCreateState, cgCreateStateAssignment

4.41 cgCreateTechnique

NAME

cgCreateTechnique - create a technique in an effect
SYNOPSIS

#include <Cg/cg.h>

CGtechnique cgCreateTechnique( CGeffect effect,
    const char * name );

PARAMETERS

    effect
    The effect to which the new technique will be added.
    name
    The name for the new technique.

RETURN VALUES

Returns the handle to the new technique on success.
Returns NULL if an error occurs.

DESCRIPTION

cgCreateTechnique adds a new technique to the specified effect.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY

cgCreateTechnique was introduced in Cg 1.5.

SEE ALSO

cgIsTechnique, cgCreatePass, cgCreateEffectParameter, cgCreateEffectParameterArray, cgCreateEffectParameterMultiDimArray

4.42 cgCreateTechniqueAnnotation

NAME

cgCreateTechniqueAnnotation - create a technique annotation

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgCreateTechniqueAnnotation( CGtechnique tech,
    const char * name,
    CGtype type );
PARAMETERS

- **tech**
  The technique to which the new annotation will be added.

- **name**
  The name of the new annotation.

- **type**
  The type of the new annotation.

RETURN VALUES

Returns the new CGAnnotation handle on success.

Returns NULL if an error occurs.

DESCRIPTION

cgCreateTechniqueAnnotation adds a new annotation to the technique.

EXAMPLES

```c
/* create a float annotation named "Apple" for CGtechnique technique */
CGAnnotation ann = cgCreateTechniqueAnnotation( tech, "Apple", CG_FLOAT );
```

ERRORS

- **CG_INVALID_TECHNIQUE_HANDLE_ERROR** is generated if `tech` is not a valid technique.
- **CG_DUPLICATE_NAME_ERROR** is generated if `name` is already used by an annotation for this technique.
- **CG_INVALID_ENUMERANT_ERROR** is generated if `type` is not CG_INT, CG_FLOAT, CG_BOOL, or CG_STRING.

HISTORY

cgCreateTechniqueAnnotation was introduced in Cg 1.5.

SEE ALSO

`cgGetNamedTechniqueAnnotation`, `cgGetFirstTechniqueAnnotation`, `cgGetNextAnnotation`

## 4.43 cgDestroyBuffer

NAME

cgDestroyBuffer - delete a buffer

SYNOPSIS

```c
#include <Cg/cg.h>

void cgDestroyBuffer( CGbuffer buffer );
```
PARAMETERS

buffer
The buffer to delete.

RETURN VALUES
None.

DESCRIPTION

cgDestroyBuffer deletes a buffer. The buffer object is not actually destroyed until no more programs are bound to the buffer object and any pending use of the buffer has completed. However, the handle buffer no longer refers to the buffer object (although it may be subsequently allocated to a different created resource).

EXAMPLES

cgDestroyBuffer( myBuffer );

ERRORS

CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

HISTORY
cgDestroyBuffer was introduced in Cg 2.0.

SEE ALSO
cgCreateBuffer, cgGLCreateBuffer

4.44 cgDestroyContext

NAME
cgDestroyContext - destroy a context

SYNOPSIS

#include <Cg/cg.h>

void cgDestroyContext( CGContext context );

PARAMETERS

context
The context to be deleted.

RETURN VALUES
None.

DESCRIPTION

cgDestroyContext deletes a Cg context object and all the programs it contains.

EXAMPLES

to-be-written
ERRORS
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY
cgDestroyContext was introduced in Cg 1.1.

SEE ALSO
cgCreateContext

4.45 cgDestroyEffect

NAME
cgDestroyEffect - destroy an effect

SYNOPSIS
#include <Cg/cg.h>

void cgDestroyEffect( CGeffect effect );

PARAMETERS
  effect
  The effect object to delete.

RETURN VALUES
None.

DESCRIPTION
cgDestroyEffect removes the specified effect object and all its associated data. Any CGeffect handles that reference this effect will become invalid after the effect is deleted. Likewise, all techniques, passes, and parameters contained in the effect also become invalid after the effect is destroyed.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY
cgDestroyEffect was introduced in Cg 1.4.

SEE ALSO
cgCreateEffect, cgCreateEffectFromFile
4.46 cgDestroyObj

NAME
cgDestroyObj - destroy an obj

SYNOPSIS

#include <Cg/cg.h>

void cgDestroyObj( CGobj obj );

PARAMETERS

obj
The object to delete.

RETURN VALUES

None.

DESCRIPTION

cgDestroyObj removed the specified object and all its associated data.

EXAMPLES

CGcontext ctx = cgCreateContext();
CGobj structObj = cgCreateObj(ctx, CG_SOURCE, src, CG_PROFILE_ARBVP1, NULL);

// Use obj, then ...

cgDestroyObj( structObj );

ERRORS

CG_INVALID_OBJ_HANDLE_ERROR is generated if obj is not a valid object handle.

HISTORY

cgDestroyObj was introduced in Cg 2.0.

SEE ALSO

cgCreateObj, cgCreateObjFromFile

4.47 cgDestroyParameter

NAME
cgDestroyParameter - destroy a parameter

SYNOPSIS

#include <Cg/cg.h>

void cgDestroyParameter( CGparameter param );
PARAMETERS

param

The parameter to destroy.

RETURN VALUES

None.

DESCRIPTION

cgDestroyParameter destroys parameters created with cgCreateParameter, cgCreateParameterArray, or cgCreateParameterMultiDimArray.

Upon destruction, param will become invalid. Any connections (see cgConnectParameter) in which param is the destination parameter will be disconnected. An error will be thrown if param is a source parameter in any connections.

The parameter being destroyed may not be one of the children parameters of a struct or array parameter. In other words it must be a CGparameter returned by one of the cgCreateParameter family of entry points.

EXAMPLES

CGcontext context = cgCreateContext();
CGparameter floatParam = cgCreateParameter(context, CG_FLOAT);
CGparameter floatParamArray = cgCreateParameterArray(context, CG_FLOAT, 5);
/* ... */
cgDestroyParameter(floatParam);
cgDestroyParameter(floatParamArray);

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_NOT_ROOT_PARAMETER_ERROR is generated if the param isn’t the top-level parameter of a struct or array that was created.

CG_PARAMETER_IS_NOT_SHARED_ERROR is generated if param does not refer to a parameter created by one of the cgCreateParameter family of entry points.

CG_CANNOT_DESTROY_PARAMETER_ERROR is generated if param is a source parameter in a connection made by cgConnectParameter. cgDisconnectParameter should be used before calling cgDestroyParameter in such a case.

HISTORY

cgDestroyParameter was introduced in Cg 1.2.

SEE ALSO

cgCreateParameter, cgCreateParameterArray, cgCreateParameterMultiDimArray

4.48 cgDestroyProgram

NAME

cgDestroyProgram - destroy a program
SYNOPSIS

```c
#include <Cg/cg.h>

void cgDestroyProgram( CGprogram program );
```

PARAMETERS

- `program`
  The program object to delete.

RETURN VALUES

None.

DESCRIPTION

cgDestroyProgram removes the specified program object and all its associated data. Any `CGprogram` variables that reference this program will become invalid after the program is deleted. Likewise, any objects contained by this program (e.g. `CGparameter` objects) will also become invalid after the program is deleted.

EXAMPLES

to-be-written

ERRORS

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.

HISTORY

cgDestroyProgram was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram, cgCreateProgramFromFile

### 4.49 cgDisconnectParameter

NAME

cgDisconnectParameter - disconnects two parameters

SYNOPSIS

```c
#include <Cg/cg.h>

void cgDisconnectParameter( CGparameter param );
```

PARAMETERS

- `param`
  The destination parameter in the connection that will be disconnected.

RETURN VALUES

None.
DESCRIPTION

cgDisconnectParameter disconnects an existing connection made with cgConnectParameter between two parameters. Since a given parameter can only be connected to one source parameter, only the destination parameter is required as an argument to cgDisconnectParameter.

If the type of param is an interface and the struct connected to it implements the interface, any sub-parameters created by the connection will also be destroyed. See cgConnectParameter for more information.

EXAMPLES

CGparameter timeParam = cgGetNamedParameter(program, "time");
CGparameter sharedTime = cgCreateParameter(context,
    cgGetParameterType(timeParam));

cgConnectParameter(sharedTime, timeParam);

/* ... */

cgDisconnectParameter(timeParam);

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgDisconnectParameter was introduced in Cg 1.2.

SEE ALSO

cgGetConnectedParameter, cgGetConnectedToParameter, cgConnectParameter

4.50 cgEvaluateProgram

NAME

cgEvaluateProgram - evaluates a Cg program on the CPU

SYNOPSIS

#include <Cg/cg.h>

void cgEvaluateProgram( CGprogram program,
    float * buf,
    int ncomps,
    int nx,
    int ny,
    int nz );

PARAMETERS

program

The program to be evaluated.

buf

Buffer in which to store the results of program evaluation.
ncombs
Number of components to store for each returned program value.

nx
Number of points at which to evaluate the program in the x direction.

ny
Number of points at which to evaluate the program in the y direction.

nz
Number of points at which to evaluate the program in the z direction.

RETURN VALUES
None.

DESCRIPTION

cgEvaluateProgram evaluates a Cg program at a set of regularly spaced points in one, two, or three dimensions. The program must have been compiled with the CG_PROFILE_GENERIC profile. The value returned from the program via the COLOR semantic is stored in the given buffer for each evaluation point, and any varying parameters to the program with POSITION semantic take on the (x,y,z) position over the range zero to one at which the program is evaluated at each point. The PSIZE semantic can be used to find the spacing between evaluating points.

The total size of buf should be equal to ncomps*nx*ny*nz.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_INVALID_PROFILE_ERROR is generated if program’s profile is not CG_PROFILE_GENERIC.

CG_INVALID_PARAMETER_ERROR is generated if buf is NULL, any of nx, ny, or nz is less than zero, or ncomps is not 0, 1, 2, or 3.

HISTORY

cgEvaluateProgram was introduced in Cg 1.4.

SEE ALSO

cgCreateProgram, cgCreateProgramFromFile, cgCreateProgramFromEffect, cgGetProgramProfile

4.51 cgGetAnnotationName

NAME

cgGetAnnotationName - get an annotation’s name

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetAnnotationName( CGannotation ann );
PARAMETERS

ann

The annotation from which to get the name.

RETURN VALUES

Returns the null-terminated name string for the annotation.
Returns NULL if ann is invalid.

DESCRIPTION

cgGetAnnotationName allows the application to retrieve the name of a annotation. This name can be used later to retrieve the annotation using cgGetNamedPassAnnotation, cgGetNamedParameterAnnotation, cgGetNamedTechniqueAnnotation, or cgGetNamedProgramAnnotation.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.

HISTORY

cgGetAnnotationName was introduced in Cg 1.4.

SEE ALSO

cgGetNamedPassAnnotation, cgGetNamedParameterAnnotation, cgGetNamedTechniqueAnnotation, cgGetNamedProgramAnnotation

4.52 cgGetAnnotationType

NAME

cgGetAnnotationType - get an annotation’s type

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetAnnotationType( CGannotation ann );

PARAMETERS

ann

The annotation from which to get the type.

RETURN VALUES

Returns the type enumerant of ann.
Returns CG_UNKNOWN_TYPE if an error occurs.
DESCRIPTION

cgGetAnnotationType allows the application to retrieve the type of an annotation in a Cg effect.

cgGetAnnotationType will return CG_STRUCT if the annotation is a struct and CG_ARRAY if the annotation is an array. Otherwise it will return the data type associated with the annotation.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.

HISTORY

cgGetAnnotationType was introduced in Cg 1.4.

SEE ALSO

cgGetType, cgGetTypeString

4.53 cgGetArrayDimension

NAME

cgGetArrayDimension - get the dimension of an array parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetArrayDimension( CGparameter param );

PARAMETERS

param

The array parameter handle.

RETURN VALUES

Returns the dimension of param if param references an array.

Returns 0 otherwise.

DESCRIPTION

cgGetArrayDimension returns the dimension of the array specified by param. cgGetArrayDimension is used when inspecting an array parameter in a program.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
HISTORY

cgGetArrayDimension was introduced in Cg 1.1.

SEE ALSO

cgGetArraySize, cgCreateParameterArray, cgCreateParameterMultiDimArray

4.54 cgGetArrayParameter

NAME

cgGetArrayParameter - get a parameter from an array

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetArrayParameter( CGparameter param,  
int index );

PARAMETERS

  param
  The array parameter handle.

  index
  The index into the array.

RETURN VALUES

Returns the parameter at the specified index of param if param references an array, and the index is valid.

Returns NULL otherwise.

DESCRIPTION

cgGetArrayParameter returns the parameter of array param specified by index. cgGetArrayParameter is used when inspecting elements of an array parameter in a program.

EXAMPLES

CGparameter array = ...; /* some array parameter */
int array_size = cgGetArraySize( array );
for(i=0; i < array_size; ++i)
{  
  CGparameter element = cgGetArrayParameter(array, i);
  /* Do stuff with element */
}

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if index is outside the bounds of param.
HISTORY

cgGetArrayParameter was introduced in Cg 1.1.

SEE ALSO

cgGetArrayDimension, cgGetArraySize, cgGetParameterType

4.55 cgGetArraySize

NAME

cgGetArraySize - get the size of one dimension of an array parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetArraySize( CGparameter param,
                        int dimension );

PARAMETERS

param

The array parameter handle.

dimension

The array dimension whose size will be returned.

RETURN VALUES

Returns the size of param if param is an array.

Returns 0 if param is not an array, or an error occurs.

DESCRIPTION

cgGetArraySize returns the size of the given dimension of the array specified by param. cgGetArraySize is used when inspecting an array parameter in a program.

EXAMPLES

/* Compute the number of elements in an array, in the */
/* style of cgGetArrayTotalSize(param) */
int elements = cgGetArraySize(param, 0);
if (elements>0) {
    int dim = cgGetArrayDimension(param);
    for (int i = 1; i < dim; i++) {
        elements *= cgGetArraySize(param, i);
    }
}

ERRORS

CG_INVALID_DIMENSION_ERROR is generated if dimension is less than 0.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
HISTORY

cgGetArraySize was introduced in Cg 1.1.

SEE ALSO

cgGetArrayTotalSize, cgGetArrayDimension, cgGetArrayParameter, cgMatrixSize, cgGetTypeSizes

4.56 cgGetArrayTotalSize

NAME

cgGetArrayTotalSize - get the total size of an array parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetArrayTotalSize(CGparameter param);

PARAMETERS

param

The array parameter handle.

RETURN VALUES

Returns the total size of param if param is an array.

Returns 0 if param is not an array, or if an error occurs.

DESCRIPTION

cgGetArrayTotalSize returns the total number of elements of the array specified by param. The total number of elements is equal to the product of the size of each dimension of the array.

EXAMPLES

Given a handle to a parameter declared as:

    float2x3 array[6][1][4];

cgGetArrayTotalSize will return 24.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetArrayTotalSize was introduced in Cg 1.4.

SEE ALSO

cgGetArraySize, cgGetArrayDimension, cgGetArrayParameter
4.57 cgGetArrayType

NAME
cgGetArrayType - get the type of an array parameter

SYNOPSIS
#include <Cg/cg.h>
CGtype cgGetArrayType( CGparameter param );

PARAMETERS
    param
The array parameter handle.

RETURN VALUES
Returns the type of the innermost array.
Returns CG_UNKNOWN_TYPE if an error occurs.

DESCRIPTION
cgGetArrayType returns the type of the members of an array. If the given array is multi-dimensional, it will return the type of the members of the innermost array.

EXAMPLES
CGcontext context = cgCreateContext();
CGparameter array = cgCreateParameterArray(context, CG_FLOAT, 5);

CGtype arrayType = cgGetArrayType(array); /* This will return CG_FLOAT */

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

HISTORY
cgGetArrayType was introduced in Cg 1.2.

SEE ALSO
cgGetArraySize, cgGetArrayDimension

4.58 cgGetAutoCompile

NAME
cgGetAutoCompile - get the auto-compile enumerant for a context
SYNOPSIS

```c
#include <Cg/cg.h>

CGenum cgGetAutoCompile( CGcontext context );
```

PARAMETERS

- `context`:
The context.

RETURN VALUES

Returns the auto-compile enumerant for `context`.

Returns `CG_UNKNOWN` if `context` is not a valid context.

DESCRIPTION

`cgGetAutoCompile` returns the auto-compile enumerant for `context`. See `cgSetAutoCompile` for more information.

EXAMPLES

`to-be-written`

ERRORS

`CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.

HISTORY

`cgGetAutoCompile` was introduced in Cg 1.4.

SEE ALSO

`cgSetAutoCompile`

### 4.59 cgGetBehavior

NAME

`cgGetBehavior` - get the behavior enumerant from a behavior name

SYNOPSIS

```c
#include <Cg/cg.h>

CGbehavior cgGetBehavior( const char * behavior_string );
```

PARAMETERS

- `behavior_string`:
  A string containing the case-sensitive behavior name.
RETURN VALUES
Returns the behavior enumerant associated with behavior_string.
Returns CG_BEHAVIOR_UNKNOWN if behavior_string is NULL or if no CGbehavior is associated with the given string.

DESCRIPTION
cgGetBehavior returns the enumerant assigned to a behavior name.

EXAMPLES
CGbehavior b = cgGetBehavior("latest");
/* b == CG_BEHAVIOR_LATEST */

ERRORS
CG_INVALID_PARAMETER_ERROR is generated if behavior_string is NULL.

HISTORY
cgGetBehavior was introduced in Cg 3.0.

SEE ALSO
cgGetBehaviorString, cgGetContextBehavior, cgSetContextBehavior

4.60 cgGetBehaviorString

NAME
cgGetBehaviorString - get the behavior name associated with a behavior enumerant

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetBehaviorString( CGbehavior behavior );

PARAMETERS
behavior
    The behavior enumerant.

RETURN VALUES
Returns the behavior string associated with behavior.
Returns NULL if behavior is not a valid CGbehavior.

DESCRIPTION
cgGetBehaviorString returns the behavior name associated with a given behavior enumerant.
EXAMPLES

static void dumpCgContextInfo(CGcontext context)
{
    const char* p = cgGetBehaviorString(cgGetContextBehavior(context));
    if ( p ) {
        printf(" Behavior: %s\n", p);
    }
    /* ... */
}

ERRORS

None.

HISTORY

cgGetBehaviorString was introduced in Cg 3.0.

SEE ALSO

cgGetBehavior, cgGetContextBehavior, cgSetContextBehavior

4.61 cgGetBoolAnnotationValues

NAME

cgGetBoolAnnotationValues - get the values from a boolean-valued annotation

SYNOPSIS

#include <Cg/cg.h>

const CGbool * cgGetBoolAnnotationValues( CGannotation ann,
                                        int * nvalues );

PARAMETERS

    ann
        The annotation.

    nvalues
        Pointer to integer where the number of returned values will be stored.

RETURN VALUES

Returns a pointer to an array of CGbool values. The number of values in the array is returned via the nvalues parameter.

Returns NULL if no values are available. nvalues will be 0.

DESCRIPTION

cgGetBoolAnnotationValues allows the application to retrieve the value(s) of a boolean typed annotation.

EXAMPLES

to-be-written
ERRORS
CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.
CG_INVALID_PARAMETER_ERROR is generated if nvalues is NULL.

HISTORY
cgGetBoolAnnotationValues was introduced in Cg 1.5.

SEE ALSO
cgGetAnnotationType, cgGetFloatAnnotationValues, cgGetIntAnnotationValues, cgGetStringAnnotationValue

4.62 cgGetBooleanAnnotationValues

NAME
cgGetBooleanAnnotationValues - deprecated

DESCRIPTION
cgGetBooleanAnnotationValues is deprecated. Use cgGetBoolAnnotationValues instead.

SEE ALSO
cgGetBoolAnnotationValues

4.63 cgGetBoolStateAssignmentValues

NAME
cgGetBoolStateAssignmentValues - get the values from a bool-valued state assignment

SYNOPSIS
#include <Cg/cg.h>

const CGbool * cgGetBoolStateAssignmentValues( CGstateassignment sa,
                                               int * nvalues);

PARAMETERS
    sa
    The state assignment.
    nvalues
    Pointer to integer where the number of returned values will be stored.

RETURN VALUES
Returns a pointer to an array of CGbool values. The number of values in the array is returned via the nvalues parameter.
Returns NULL if an error occurs or if no values are available. nvalues will be 0 in the latter case.
DESCRIPTION

`cgGetBoolStateAssignmentValues` allows the application to retrieve the value(s) of a boolean typed state assignment.

EXAMPLES

* to-be-written

ERRORS

- `CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.
- `CG_INVALID_PARAMETER_ERROR` is generated if `nvalues` is `NULL`.
- `CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR` is generated if `sa` is not a state assignment of a bool type.

HISTORY

`cgGetBoolStateAssignmentValues` was introduced in Cg 1.4.

SEE ALSO

- `cgGetStateAssignmentState`, `cgGetStateType`, `cgGetFloatStateAssignmentValues`, `cgGetIntStateAssignmentValues`, `cgGetStringStateAssignmentValue`, `cgGetProgramStateAssignmentValue`, `cgGetSamplerStateAssignmentValue`, `cgGetTextureStateAssignmentValue`

4.64 cgGetBufferSize

NAME

`cgGetBufferSize` - get the size of a buffer

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetBufferSize( CGbuffer buffer );
```

PARAMETERS

- `buffer`:
  The buffer for which the size will be retrieved.

RETURN VALUES

Returns the size in bytes of `buffer`.

Returns `-1` if an error occurs.

DESCRIPTION

`cgGetBufferSize` returns the size in bytes of `buffer`.

EXAMPLES

```c
int size = cgGetBufferSize( myBuffer );
```
ERRORS

`CG_INVALID_BUFFER_HANDLE_ERROR` is generated if `buffer` is not a valid buffer.

HISTORY

cgGetBufferSize was introduced in Cg 2.0.

SEE ALSO

cgCreateBuffer, cgGLCreateBuffer, cgSetBufferData, cgSetBufferSubData

### 4.65 cgGetCompilerIncludeCallback

NAME

cgGetCompilerIncludeCallback - get the include callback function

SYNOPSIS

```c
#include <Cg/cg.h>

typedef void (*CGIncludeCallbackFunc)( CGcontext context, const char *filename );
CGIncludeCallbackFunc cgGetCompilerIncludeCallback( CGcontext context );
```

PARAMETERS

context

The context of the desired include callback function.

RETURN VALUES

Returns the current include callback function.

Returns `NULL` if no callback function is set.

DESCRIPTION

cgGetCompilerIncludeCallback returns the current callback function used for handing include statements.

EXAMPLES

```c
CGIncludeCallbackFunc includeCB = cgGetCompilerIncludeCallback(context);
```

ERRORS

`CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.

HISTORY

cgGetCompilerIncludeCallback was introduced in Cg 2.1.

SEE ALSO

cgSetCompilerIncludeCallback, cgSetCompilerIncludeString, cgSetCompilerIncludeFile
4.66 cgGetConnectedParameter

NAME

cgGetConnectedParameter - gets the connected source parameter

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetConnectedParameter( CGparameter param );

PARAMETERS

param
The destination parameter.

RETURN VALUES

Returns the connected source parameter if param is connected to one.
Returns NULL otherwise.

DESCRIPTION

Returns the source parameter to which param is connected.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetConnectedParameter was introduced in Cg 1.2.

SEE ALSO

cgConnectParameter, cgDisconnectParameter, cgGetConnectedToParameter

4.67 cgGetConnectedStateAssignmentParameter

NAME

cgGetConnectedStateAssignmentParameter - get effect parameter which determines a state assignment’s value

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetConnectedStateAssignmentParameter( CGstateassignment sa );
PARAMETERS

sa

A state assignment whose value is determined using an effect parameter.

RETURN VALUES

Returns the effect parameter used by sa.
Returns 0 if sa is not using a parameter for its value, if the state assignment is set to an expression, or if an error occurs.

DESCRIPTION

cgGetConnectedStateAssignmentParameter returns the effect parameter from which a given state assignment’s value is determined.

EXAMPLES

/* in Effect.cgfx file */

int MyMinFilter;
sampler2D Samp = sampler_state {
    MinFilter = MyMinFilter;
};

/* in .c/.cpp file */

CGparameter sampParam = cgGetNamedEffectParameter( myEffect, "Samp" );
CGstateassignment sa = cgGetNamedSamplerStateAssignment( sampParam, "MinFilter" );
CGparameter connected = cgGetConnectedStateAssignmentParameter( sa );

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetConnectedStateAssignmentParameter was introduced in Cg 2.0.

SEE ALSO

cgGetNamedEffectParameter, cgGetNamedSamplerStateAssignment

4.68 cgGetConnectedToParameter

NAME

cgGetConnectedToParameter - gets a connected destination parameter

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetConnectedToParameter( CGparameter param,
                                int index );
PARAMETERS

    param
    The source parameter.
    index
    Since there may be multiple destination (to) parameters connected to `param`, `index` is need to specify which one is returned. `index` must be within the range of 0 to N - 1 where N is the number of connected destination parameters.

RETURN VALUES

Returns one of the destination parameters connected to `param`. Returns NULL if an error occurs.

DESCRIPTION

Returns one of the destination parameters connected to `param`. `cgGetNumConnectedToParameters` should be used to determine the number of destination parameters connected to `param`.

EXAMPLES

```cpp
int nParams = cgGetNumConnectedToParameters( sourceParam );

for ( int i=0; i < nParams; ++i )
{
    CGparameter toParam = cgGetConnectedToParameter( sourceParam, i );
    /* Do stuff with toParam ... */
}
```

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

`CG_OUT_OF_ARRAY_BOUNDS_ERROR` is generated if `index` is less than 0 or greater than or equal to the number of parameters connected to `param`.

HISTORY

`cgGetConnectedToParameter` was introduced in Cg 1.2.

SEE ALSO

`cgConnectParameter`, `cgGetNumConnectedToParameters`

4.69 cgGetContextBehavior

NAME

`cgGetContextBehavior` - get the behavior enumerant for a context

SYNOPSIS

```cpp
#include <Cg/cg.h>

CGbehavior cgGetContextBehavior( CGcontext context );
```
PARAMETERS

context

The context for which the behavior enumerant will be returned.

RETURN VALUES

Returns the behavior enumerant for context.

Returns CG_BEHAVIOR_UNKNOWN if an error occurs.

DESCRIPTION

cgGetContextBehavior allows the application to retrieve the behavior enumerant for a context. The valid enumerants and their meanings are described in cgSetContextBehavior.

EXAMPLES

/* create a context and get the default context behavior enum */

CGcontext context = cgCreateContext();
CGbehavior cb = cgGetContextBehavior(context);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetContextBehavior was introduced in Cg 3.0.

SEE ALSO

cgCreateContext, cgSetContextBehavior, cgGetBehavior, cgGetBehaviorString

4.70 cgGetDependentAnnotationParameter

NAME

cgGetDependentAnnotationParameter - get one of the parameters that an annotation’s value depends on

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetDependentAnnotationParameter( CGannotation ann,
                                                int index );

PARAMETERS

ann

The annotation handle.

index

The index of the parameter to return.
RETURN VALUES

Returns a handle to the selected dependent annotation on success.
Returns NULL if an error occurs.

DESCRIPTION

Annotations in CgFX files may include references to one or more effect parameters on the right hand side of the annotation that are used for computing the annotation’s value. **cgGetDependentAnnotationParameter** returns one of these parameters, as indicated by the given index. **cgGetNumDependentAnnotationParameters** can be used to determine the total number of such parameters.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter’s value.

EXAMPLES

to-be-written

ERRORS

**CG_INVALID_ANNOTATION_HANDLE_ERROR** is generated if **ann** is not a valid annotation.

**CG_OUT_OF_ARRAY_BOUNDS_ERROR** is generated if **index** is less than zero or greater than or equal to the number of dependent parameters, as returned by **cgGetNumDependentAnnotationParameters**.

HISTORY

cgGetDependentAnnotationParameter was introduced in Cg 1.4.

SEE ALSO

cgGetDependentStateAssignmentParameter, cgGetNumDependentAnnotationParameters

4.71  cgGetDependentProgramArrayStateAssignmentParameter

NAME

cgGetDependentProgramArrayStateAssignmentParameter - get one of the parameters that a state assignment’s value depends on

SYNOPSIS

```c
#include <Cg/cg.h>

CGparameter cgGetDependentProgramArrayStateAssignmentParameter( CGstateassignment sa, int index);
```

PARAMETERS

- **sa**
  The state assignment handle.

- **index**
  The index of the parameter to return.
**RETURN VALUES**

Returns a handle to the selected dependent parameter on success.

Returns NULL if sa is not a program state assignment or an error occurs.

**DESCRIPTION**

State assignments in CgFX files may include references to an array indexed by an effect parameter (or expression) on the right hand side of the state assignment that is used for computing the state assignment’s value. Usually this array holds the compile statements of shader programs and by changing the index of the shader array, it’s possible to switch to a different program or profile on-the-fly.

Each compile statement in the array can depend on one or more effect parameters which are passed to the program in its parameter list. It is sometimes necessary for the application to query what those parameters are so values can be properly set to them.

cgGetDependentProgramArrayStateAssignmentParameter returns one of these parameters, as indicated by the given index.

**EXAMPLES**

In CgFX file:

```cpp
vertexshader Torus[4] =
{
    compile vp40 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile vp30 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile arbvp1 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile vp20 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) )
};

pixelshader SpecSurf[4] =
{
    compile fp40 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube ),
    compile fp30 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube ),
    compile arbfp1 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube ),
    compile fp20 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube )
};

int select = 0;
```
technique bumpdemo
{
    pass
    {
        VertexProgram = (Torus[select]);
        FragmentProgram = (SpecSurf[select]);
    }
}

In application:

```c
int numParameters = cgGetNumDependentProgramArrayStateAssignmentParameters(stateAssignment);
for(int i = 0; i < numParameters; ++i) {
    CGparameter param = cgGetDependentProgramArrayStateAssignmentParameter(stateAssignment, i);
    /* Set value for 'param' */
}
```

In the above example, assuming select = 0 and stateAssignment is for VertexProgram, the list of parameters returned from `cgGetDependentProgramArrayStateAssignmentParameter` would be: LightPosition, EyePosition, ModelViewProj, OuterRadius, InnerRadius.

If stateAssignment was for FragmentProgram, then the list of parameters returned from `cgGetDependentProgramArrayStateAssignmentParameter` would be: Ambient, DiffuseMaterial, LightColor, SpecularMaterial, LightColor, normalMap, normalizeCube, normalizeCube.

**ERRORS**

`CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.

`CG_OUT_OF_ARRAY_BOUNDS_ERROR` is generated if `index` is less than zero or greater than or equal to the number of dependent parameters, as returned by `cgGetNumDependentProgramArrayStateAssignmentParameters`.

**HISTORY**

cgGetDependentProgramArrayStateAssignmentParameter was introduced in Cg 3.0.

**SEE ALSO**

cgGetNumDependentProgramArrayStateAssignmentParameters

### 4.72 cgGetDependentStateAssignmentParameter

**NAME**

cgGetDependentStateAssignmentParameter - get one of the parameters that a state assignment’s value depends on

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgGetDependentStateAssignmentParameter( CGstateassignment sa, int index );
```

**PARAMETERS**

- `sa`  
  The state assignment handle.
index

The index of the parameter to return.

**RETURN VALUES**

Returns a handle to the selected dependent parameter on success.

Returs `NULL` if an error occurs.

**DESCRIPTION**

State assignments in CgFX files may include references to one or more effect parameters on the right hand side of the state assignment that are used for computing the state assignment’s value. `cgGetDependentStateAssignmentParameter` returns one of these parameters, as indicated by the given index. `cgGetNumDependentStateAssignmentParameters` can be used to determine the total number of such parameters.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter’s value.

**EXAMPLES**

*to-be-written*

**ERRORS**

`CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.

`CG_OUT_OF_ARRAY_BOUNDS_ERROR` is generated if `index` is less than zero or greater than or equal to the number of dependent parameters, as returned by `cgGetNumDependentStateAssignmentParameters`.

**HISTORY**

`cgGetDependentStateAssignmentParameter` was introduced in Cg 1.4.

**SEE ALSO**

`cgGetDependentAnnotationParameter`, `cgGetNumDependentStateAssignmentParameters`

### 4.73 cgGetDomain

**NAME**

`cgGetDomain` - get the domain enumerant from a domain name

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGdomain cgGetDomain( const char * domain_string );
```

**PARAMETERS**

- **domain_string**
  
  A string containing the case-sensitive domain name.
RETURN VALUES

Returns the domain enumerant of domain_string.
Returns CG_UNKNOWN if the given domain does not exist.

DESCRIPTION

cgGetDomain returns the enumerant assigned to a domain name.

EXAMPLES

CGdomain ARBVP1domain = cgGetDomain("arbvp1");

if(cgGetProgramDomain(myprog) == ARBVP1Domain)
{
    /* Do stuff */
}

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if domain_string is NULL.

HISTORY

cgGetDomain was introduced in Cg 2.2.

SEE ALSO

cgGetDomainString, cgGetProgramDomain

4.74 cgGetDomainString

NAME

 cgGetDomainString - get the domain name associated with a domain enumerant

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetDomainString( CGdomain domain );

PARAMETERS

domain

The domain enumerant.

RETURN VALUES

Returns the domain string of the enumerant domain.
Returns NULL if domain is not a valid domain.

DESCRIPTION

cgGetDomainString returns the domain name associated with a domain enumerant.
EXAMPLES

static void dumpCgProgramInfo(CGprogram program)
{
    const char * p = cgGetDomainString(cgGetProgramDomain(program));
    if ( p ) {
        printf(" Domain: %s\n", cgGetDomainString(cgGetProgramDomain(program)));
    }
    /* ... */
}

ERRORS

None.

HISTORY

cgGetDomainString was introduced in Cg 2.2.

SEE ALSO

cgGetDomain, cgGetProgramDomain

4.75 cgGetEffectContext

NAME

cgGetEffectContext - get a effect’s context

SYNOPSIS

#include <Cg/cg.h>

CGcontext cgGetEffectContext( CGeffect effect );

PARAMETERS

effect

    The effect.

RETURN VALUES

Returns the context to which effect belongs.
Returns NULL if an error occurs.

DESCRIPTION

cgGetEffectContext allows the application to retrieve a handle to the context to which a given effect belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.
HISTORY

cgGetEffectContext was introduced in Cg 1.4.

SEE ALSO

cgCreateEffect, cgCreateEffectFromFile, cgCreateContext

4.76 cgGetEffectName

NAME

cgGetEffectName - get an effect’s name

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetEffectName( CGeffect effect );

PARAMETERS

effect

The effect from which the name will be retrieved.

RETURN VALUES

Returns the name from the specified effect.

Returns NULL if the effect doesn’t have a valid name or an error occurs.

DESCRIPTION

cgGetEffectName returns the name from the specified effect.

EXAMPLES

char *effectSource = ...;
CGcontext context = cgCreateContext();
CGeffect effect = cgCreateEffect(context, effectSource, NULL);

const char * myEffectName = "myEffectName";
CGbool okay = cgSetEffectName(effect, myEffectName);
if (!okay) {
    /* handle error */
}

const char * testName = cgGetEffectName(effect);

if (strcmp(testName, myEffectName)) {
    /* shouldn’t be here */
}

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.
HISTORY
cgGetEffectName was introduced in Cg 1.5.

SEE ALSO
cgSetEffectName

4.77 cgGetEffectParameterBuffer

NAME
cgGetEffectParameterBuffer - get the Cg buffer associated to the passed effect parameter.

SYNOPSIS
#include <Cg/cg.h>

CGbuffer cgGetEffectParameterBuffer(CGparameter param);

PARAMETERS

param
The effect parameter associated with a Cg buffer (using the BUFFER semantic) set by cgSetEffectParameterBuffer.

RETURN VALUES
Returns the CGbuffer object set by cgSetEffectParameterBuffer.

Returns NULL if param is invalid or does not have a CGbuffer set by cgSetEffectParameterBuffer.

DESCRIPTION
cgGetEffectParameterBuffer returns the CGbuffer object set by cgSetEffectParameterBuffer.

EXAMPLES
In effect:

struct Material{
  float4 ambient;
  float4 diffuse;
  float4 specular;
  float4 shine;
} cbuffer0_Material : BUFFER[0];

In C/C++:

CGbuffer myCgBuffer = cgCreateBuffer(...);

cgSetEffectParameterBuffer(cgGetNamedEffectParameter(myCgEffect, "cbuffer0_Material"), myCgBuffer);

// ...

CGbuffer buffer = cgGetEffectParameterBuffer(cgGetNamedEffectParameter(myCgEffect, "cbuffer0_Material"));

// Now buffer == myCgBuffer
ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetEffectParameterBuffer was introduced in Cg 3.0.

SEE ALSO

cgSetEffectParameterBuffer

### 4.78 cgGetEffectParameterBySemantic

**NAME**

cgGetEffectParameterBySemantic - get the a parameter in an effect via its semantic

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgGetEffectParameterBySemantic( CGeffect effect,
                                          const char * semantic );
```

**PARAMETERS**

- `effect`
  The effect from which to retrieve the parameter.
- `semantic`
  The name of the semantic.

**RETURN VALUES**

Returns the `CGparameter` object in `effect` that has the given semantic.

Returns NULL if `effect` is invalid or does not have any parameters with the given semantic.

**DESCRIPTION**

`cgGetEffectParameterBySemantic` returns the parameter in an effect which is associated with the given semantic. It multiple parameters in the effect have this semantic, an arbitrary one of them will be returned.

**EXAMPLES**

`to-be-written`

**ERRORS**

CG_INVALID_EFFECT_HANDLE_ERROR is generated if `effect` is not a valid effect.

CG_INVALID_PARAMETER_ERROR is generated if `semantic` is NULL or the empty string.

**HISTORY**

cgGetEffectParameterBySemantic was introduced in Cg 1.4.
SEE ALSO
cgGetNamedEffectParameter

4.79 cgGetEnum

NAME
cgGetEnum - get the enumerant assigned with the given string name

SYNOPSIS
#include <Cg/cg.h>

CGenum cgGetEnum( const char * enum_string );

PARAMETERS

enum_string
A string containing the case-sensitive enum name.

RETURN VALUES

Returns the enumerant for enum_string.

Returns CG_UNKNOWN if no such enumerant exists

DESCRIPTION
cgGetEnum returns the enumerant assigned to an enum name.

EXAMPLES

CGenum VaryingEnum = cgGetEnum("CG_VARYING");

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if enum_string is NULL.

HISTORY
cgGetEnum was introduced in Cg 1.2.

SEE ALSO
cgGetEnumString

4.80 cgGetEnumString

NAME
cgGetEnumString - get the name string associated with an enumerant

4.79. cgGetEnum
SYNOPSIS

#include <Cg/cg.h>

const char * cgGetEnumString( CGenum enum );

PARAMETERS

enum

The enumerant.

RETURN VALUES

Returns the string representation of the enumerant enum.

Returns NULL if enum is not a valid Cg enumerant.

DESCRIPTION

cgGetEnumString returns the name string associated with an enumerant. It’s primary use to print debugging information.

EXAMPLES

/* This prints "CG_UNIFORM" to stdout */
const char *EnumString = cgGetEnumString(CG_UNIFORM);
printf("%s\n", EnumString);

ERRORS

None.

HISTORY

cgGetEnumString was introduced in Cg 1.2.

SEE ALSO

cgGetEnum

4.81 cgGetError

NAME

cgGetError - get error condition

SYNOPSIS

#include <Cg/cg.h>

CGerror cgGetError( void );

PARAMETERS

None.
RETURN VALUES

Returns the last error condition that has occurred.
Returns CG_NO_ERROR if no error has occurred.

DESCRIPTION

cgGetError returns the last error condition that has occurred. The error condition is reset after cgGetError is called.

EXAMPLES

CGerror err = cgGetError();

ERRORS

None.

HISTORY

cgGetError was introduced in Cg 1.1.

SEE ALSO

cgSetErrorCallback, cgSetErrorHandler

4.82 cgGetErrorCallback

NAME

cgGetErrorCallback - get the error callback function

SYNOPSIS

#include <Cg/cg.h>

typedef void (*CGerrorCallbackFunc)( void);

CGerrorCallbackFunc cgGetErrorCallback( void);

PARAMETERS

None.

RETURN VALUES

Returns the currently set error callback function.
Returns NULL if no callback function has been set.

DESCRIPTION

cgGetErrorCallback returns the current error callback function.

EXAMPLES

CGerrorCallbackFunc errorCB = cgGetErrorCallback();
ERRORS
None.

HISTORY

cgGetErrorCallback was introduced in Cg 1.1.

SEE ALSO

cgSetErrorCallback

4.83 cgGetErrorHandler

NAME
cgGetErrorHandler - get the error handler callback function

SYNOPSIS

#include <Cg/cg.h>

typedef void (*)(CGerrorHandlerFunc)( CGcontext context,
               CGerror error,
               void * appdata );

CGerrorHandlerFunc cgGetErrorHandler( void ** appdataptr );

PARAMETERS

appdataptr

A pointer for an application provided data pointer.

RETURN VALUES

Returns the current error handler callback function.
Returns NULL if no callback function is set.
If appdataptr is not NULL then the current appdata pointer will be copied into the location pointed to by appdataptr.

DESCRIPTION

cgGetErrorHandler returns the current error handler callback function and application provided data pointer.

EXAMPLES

void * appdata = NULL;
CGerrorHandlerFunc errorHandler = cgGetErrorHandler( &appdata );

ERRORS

None.

HISTORY

cgGetErrorHandler was introduced in Cg 1.4.
SEE ALSO

cgSetErrorHandler

4.84 cgGetErrorString

NAME
cgGetErrorString - get a human readable error string

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetErrorString( CGerror error );

PARAMETERS

   error
   The error condition.

RETURN VALUES

Returns a human readable error string for the given error condition.

DESCRIPTION

cgGetErrorString returns a human readable error string for the given error condition.

EXAMPLES

const char * pCompilerError = cgGetErrorString( CG_COMPILER_ERROR );

ERRORS

None.

HISTORY

cgGetErrorString was introduced in Cg 1.1.

SEE ALSO

cgGetError

cgGetFirstDependentParameter

4.85 cgGetFirstDependentParameter

NAME

cgGetFirstDependentParameter - get the first dependent parameter from a parameter

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetFirstDependentParameter( CGparameter param );

4.84. cgGetErrorString
PARAMETERS

param

The parameter.

RETURN VALUES

Returns a handle to the first member parameter.

Returns NULL if param is not a struct or if some other error occurs.

DESCRIPTION

cgGetFirstDependentParameter returns the first member dependent parameter associated with a given parameter.

The rest of the members may be retrieved from the first member by iterating with cgGetNextParameter.

Dependent parameters are parameters that have the same name as a given parameter but different resources. They only exist in profiles that have multiple resources associated with one parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetFirstDependentParameter was introduced in Cg 1.1.

SEE ALSO

cgGetNextParameter, cgGetFirstParameter

4.86 cgGetFirstEffect

NAME

cgGetFirstEffect - get the first effect in a context

SYNOPSIS

#include <Cg/cg.h>

CGeffect cgGetFirstEffect( CGcontext context );

PARAMETERS

context

The context from which to retrieve the first effect.

RETURN VALUES

Returns the first CGeffect object in context.

Returns NULL if context contains no effects.
DESCRIPTION

cgGetFirstEffect is used to begin iteration over all of the effects contained by a context. See \textit{cgGetNextEffect} for more information.

EXAMPLES

/* one or more effects have previously been loaded into context */
CGeffect effect = cgGetFirstEffect( context );

ERRORS

\textbf{CG_INVALID_CONTEXT_HANDLE_ERROR} is generated if context is not a valid context.

HISTORY

cgGetFirstEffect was introduced in Cg 1.4.

SEE ALSO

cgGetNextEffect, cgCreateEffect, cgCreateEffectFromFile, cgDestroyEffect, cgIsEffect, cgGetFirstProgram

4.87 \textbf{cgGetFirstEffectAnnotation}

NAME

cgGetFirstEffectAnnotation - get the first annotation in an effect

SYNOPSIS

\begin{verbatim}
#include <Cg/cg.h>

CGannotation cgGetFirstEffectAnnotation( CGeffect effect );
\end{verbatim}

PARAMETERS

effect

The effect from which to retrieve the first annotation.

RETURN VALUES

Returns the first annotation in an effect.

Returns NULL if the effect has no annotations.

DESCRIPTION

The first annotation associated with an effect can be retrieved using \textit{cgGetFirstEffectAnnotation}. The rest of the effect’s annotations can be discovered by iterating through them using \textit{cgGetNextAnnotation}.

EXAMPLES

CGannotation ann = cgGetFirstEffectAnnotation( effect );
while( ann )
{
   /* do something with ann */
   ann = cgGetNextAnnotation( ann );
}

4.87. cgGetFirstEffectAnnotation 221
ERRORS

**CG_INVALID_EFFECT_HANDLE_ERROR** is generated if `effect` is not a valid effect.

HISTORY

cgGetFirstEffectAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetNamedEffectAnnotation, cgGetNextAnnotation

### 4.88 cgGetFirstEffectParameter

**NAME**

cgGetFirstEffectParameter - get the first parameter in an effect

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgGetFirstEffectParameter( CGeffect effect );
```

**PARAMETERS**

- `effect`
  
The effect from which to retrieve the first parameter.

**RETURN VALUES**

Returns the first `CGparameter` object in `effect`.

Returns `NULL` if `effect` is invalid or if `effect` does not have any parameters.

**DESCRIPTION**

The first top-level parameter in an effect can be retrieved using `cgGetFirstEffectParameter`. The rest of the effect’s parameters can be discovered by iterating through them using `cggetNextParameter`.

**EXAMPLES**

```c
CGparameter param = cgGetFirstEffectParameter( effect );
while ( param )
{
    /* do something with param */
    param = cgGetNextParameter( param );
}
```

**ERRORS**

**CG_INVALID_EFFECT_HANDLE_ERROR** is generated if `effect` is not a valid effect.

HISTORY

cgGetFirstEffectParameter was introduced in Cg 1.4.
SEE ALSO

cgGetNextParameter, cgGetNamedEffectParameter

4.89 cgGetFirstError

NAME
cgGetFirstError - get the first error condition

SYNOPSIS

#include <Cg/cg.h>

CGerror cgGetFirstError( void );

PARAMETERS

None.

RETURN VALUES

Returns the first error condition that has occurred since cgGetFirstError was last called.

Returns CG_NO_ERROR if no error has occurred.

DESCRIPTION

cgGetFirstError returns the first error condition that has occurred since cgGetFirstError was previously called.

EXAMPLES

CGerror firstError = cgGetFirstError();

ERRORS

None.

HISTORY

cgGetFirstError was introduced in Cg 1.4.

SEE ALSO

cSetErrorHandler, cgGetError

4.90 cgGetFirstLeafEffectParameter

NAME

cgGetFirstLeafEffectParameter - get the first leaf parameter in an effect

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetFirstLeafEffectParameter( CGeffect effect );
PARAMETERS
   effect
   The effect from which to retrieve the first leaf parameter.

RETURN VALUES
Returns the first leaf CGparameter object in effect.
Returns NULL if effect is invalid or if effect does not have any parameters.

DESCRIPTION

cgGetFirstLeafEffectParameter returns the first leaf parameter in an effect. The combination of cgGetFirstLeafEffectParameter and cgGetNextLeafParameter allows the iteration through all of the parameters of basic data types (not structs or arrays) without recursion. See cgGetNextLeafParameter for more information.

EXAMPLES

CGparameter leaf = cgGetFirstLeafEffectParameter( effect );
while (leaf)
{
    /* Do stuff with leaf */
    leaf = cgGetNextLeafParameter( leaf );
}

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY

cgGetFirstLeafEffectParameter was introduced in Cg 1.4.

SEE ALSO

cgGetNextLeafParameter, cgGetFirstLeafParameter

4.91 cgGetFirstLeafParameter

NAME
cgGetFirstLeafParameter - get the first leaf parameter in a program

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetFirstLeafParameter( CGprogram program,
                                    CGenum name_space );

PARAMETERS

   program
   The program from which to retrieve the first leaf parameter.
   name_space
Specifies the parameter namespace through which to iterate. Currently `CG_PROGRAM` and `CG_GLOBAL` are supported.

RETURN VALUES

Returns the first leaf `CGParameter` object in `program`.

Returns `NULL` if `program` is invalid or if `program` does not have any parameters.

DESCRIPTION

`cgGetFirstLeafParameter` returns the first leaf parameter in a program. The combination of `cgGetFirstLeafParameter` and `cgGetNextLeafParameter` allow the iteration through all of the parameters of basic data types (not structs or arrays) without recursion. See `cgGetNextLeafParameter` for more information.

EXAMPLES

```c
CGparameter leaf = cgGetFirstLeafParameter( program );
while ( leaf )
{
  /* Do stuff with leaf */
  leaf = cgGetNextLeafParameter( leaf );
}
```

ERRORS

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.

`CG_INVALID_ENUMERANT_ERROR` is generated if `name_space` is not `CG_PROGRAM` or `CG_GLOBAL`.

HISTORY

`cgGetFirstLeafParameter` was introduced in Cg 1.1.

SEE ALSO

`cgGetNextLeafParameter`

### 4.92 cgGetFirstParameter

#### NAME

`cgGetFirstParameter` - get the first parameter in a program

#### SYNOPSIS

```c
#include <Cg/cg.h>

CGparameter cgGetFirstParameter( CGprogram program,
              CGenum name_space );
```

#### PARAMETERS

- `program`
  - The program from which to retrieve the first parameter.

- `name_space`
  -
Specifies the parameter namespace through which to iterate. Currently **CG_PROGRAM** and **CG_GLOBAL** are supported.

**RETURN VALUES**

Returns the first **CGParameter** object in **program**.

Returns zero if **program** is invalid or if **program** does not have any parameters.

Also returns zero if **program** is a combined program. To access the parameters of a combined program, use **cgGetProgramDomainProgram** to get each domain program and then call **cgGetFirstParameter** on each domain program.

**DESCRIPTION**

**cgGetFirstParameter** returns the first top-level parameter in a program. **cgGetFirstParameter** is used for recursing through all parameters in a program. See **cgGetNextParameter** for more information on parameter traversal.

**EXAMPLES**

```c
CGparameter param = cgGetFirstParameter( program, CG_GLOBAL );
while ( param )
{
    /* Do stuff with leaf */
    param = cgGetNextParameter( param );
}
```

**ERRORS**

**CG_INVALID_PROGRAMHANDLE_ERROR** is generated if **program** is not a valid program handle.

**CG_INVALID_ENUMERANT_ERROR** is generated if **name_space** is not **CG_PROGRAM** or **CG_GLOBAL**.

**HISTORY**

**cgGetFirstParameter** was introduced in Cg 1.1.

**SEE ALSO**

**cgGetNextParameter**, **cgGetProgramDomainProgram**, **cgGetFirstDependentParameter**, **cgGetFirstEffectParameter**, **cgGetFirstParameterAnnotation**

4.93 **cgGetFirstParameterAnnotation**

**NAME**

**cgGetFirstParameterAnnotation** - get the first annotation of a parameter

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGannotation cgGetFirstParameterAnnotation( CGparameter param );
```

**PARAMETERS**

param

The parameter from which to retrieve the annotation.
RETURN VALUES

Returns the first annotation for the given parameter.

Returns NULL if the parameter has no annotations or an error occurs.

DESCRIPTION

The annotations associated with a parameter can be retrieved with cgGetFirstParameterAnnotation. Use cgGetNextAnnotation to iterate through the remainder of the parameter’s annotations.

EXAMPLES

CGannotation ann = cgGetFirstParameterAnnotation( param );
while ( ann )
{
    /* do something with ann */
    ann = cgGetNextAnnotation( ann );
}

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetFirstParameterAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetNamedParameterAnnotation, cgGetNextAnnotation

4.94 cgGetFirstPass

NAME

cgGetFirstPass - get the first pass in a technique

SYNOPSIS

#include <Cg/cg.h>

CGpass cgGetFirstPass( CGtechnique tech );

PARAMETERS

    tech

The technique from which to retrieve the first pass.

RETURN VALUES

Returns the first CGpass object in tech.

Returns NULL if tech contains no passes.

DESCRIPTION

cgGetFirstPass is used to begin iteration over all of the passes contained within a technique. See cgGetNextPass for more information.
EXAMPLES

CGpass pass = cgGetFirstPass( tech );
while ( pass )
{
    /* Do stuff with pass */
    leaf = cgGetNextPass( pass );
}

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetFirstPass was introduced in Cg 1.4.

SEE ALSO

cgGetNextPass, cgGetNamedPass, cgIsPass, cgGetFirstPassAnnotation

4.95 cgGetFirstPassAnnotation

NAME

cgGetFirstPassAnnotation - get the first annotation of a pass

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetFirstPassAnnotation( CGpass pass );

PARAMETERS

pass

The pass from which to retrieve the annotation.

RETURN VALUES

Returns the first annotation from the given pass.

Returns NULL if the pass has no annotations or an error occurs.

DESCRIPTION

The annotations associated with a pass can be retrieved using cgGetFirstPassAnnotation. The remainder of the pass’s annotations can be discovered by iterating through the parameters, calling cgGetNextAnnotation to get to the next one.

EXAMPLES

CGannotation ann = cgGetFirstPassAnnotation( pass );
while( ann )
{
    /* do something with ann */
    ann = cgGetNextAnnotation( ann );
}
ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY

cgGetFirstPassAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetNamedPassAnnotation, cgGetNextAnnotation

4.96 cgGetFirstProgram

NAME

cgGetFirstProgram - get the first program in a context

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgGetFirstProgram( CGcontext context );

PARAMETERS

context

The context from which to retrieve the first program.

RETURN VALUES

Returns the first CGprogram object in context.

Returns NULL if context contains no programs or an error occurs.

DESCRIPTION

cgGetFirstProgram is used to begin iteration over all of the programs contained within a context. See cgGetNextProgram for more information.

EXAMPLES

CGprogram program = cgGetFirstProgram( context );
while ( program )
{
    /* do something with program */
    program = cgGetNextProgram( program );
}

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetFirstProgram was introduced in Cg 1.1.
SEE ALSO

cgGetNextProgram, cgCreateProgram, cgDestroyProgram, cgIsProgram, cgGetFirstEffect

4.97 cgGetFirstProgramAnnotation

NAME
cgGetFirstProgramAnnotation - get the first annotation of a program

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetFirstProgramAnnotation( CGprogram program );

PARAMETERS

program

The program from which to retrieve the annotation.

RETURN VALUES

Returns the first annotation from the given program.

Returns NULL if the program has no annotations.

DESCRIPTION

The annotations associated with a program can be retrieved using cgGetFirstProgramAnnotation. The remainder of the program’s annotations can be discovered by iterating through the parameters, calling cgGetNextAnnotation to get to the next one.

EXAMPLES

CGannotation ann = cgGetFirstProgramAnnotation( program );
while( ann )
{
    /* do something with ann */
    ann = cgGetNextAnnotation( ann );
}

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetFirstProgramAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetNamedProgramAnnotation, cgGetNextAnnotation
4.98  cgGetFirstSamplerState

NAME

cgGetFirstSamplerState  -  get the first sampler state definition in a context

SYNOPSIS

#include <Cg/cg.h>

CGstate cgGetFirstSamplerState( CGcontext context );

PARAMETERS

context

The context from which to retrieve the first sampler state definition.

RETURN VALUES

Returns the first CGstate object in context.

Returns NULL if context contains no programs or an error occurs.

DESCRIPTION

cgGetFirstSamplerState is used to begin iteration over all of the sampler state definitions contained within a context. See cgGetNextState for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetFirstSamplerState was introduced in Cg 1.4.

SEE ALSO

cgGetNextState, cgGetNamedSamplerState

4.99  cgGetFirstSamplerStateAssignment

NAME

 cgGetFirstSamplerStateAssignment  -  get the first state assignment in a sampler_state block

SYNOPSIS

#include <Cg/cg.h>

CGstateassignment cgGetFirstSamplerStateAssignment( CGparameter param );

4.98. cgGetFirstSamplerState
PARAMETERS

    param
    The sampler parameter from which to retrieve the first state assignment.

RETURN VALUES

Returns the first CGstateassignment object assigned to param.
Returns NULL if param has no sampler_state block or an error occurs.

DESCRIPTION

cgGetFirstSamplerStateAssignment is used to begin iteration over all of the state assignments contained within a
sampler_state block assigned to a parameter in an effect file. See cgGetNextStateAssignment for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetFirstSamplerStateAssignment was introduced in Cg 1.4.

SEE ALSO

cgGetNextStateAssignment, cgIsStateAssignment

4.100 cgGetFirstState

NAME

cgGetFirstState - get the first state definition in a context

SYNOPSIS

#include <Cg/cg.h>

CGstate cgGetFirstState( CGcontext context );

PARAMETERS

    context
    The context from which to retrieve the first state definition.

RETURN VALUES

Returns the first CGstate object in context.
Returns NULL if context contains no state definitions or an error occurs.

DESCRIPTION

cgGetFirstState is used to begin iteration over all of the state definitions contained within a context. See cgGet-
NextState for more information.
EXAMPLES

*to-be-written*

ERRORS

**CG_INVALID_CONTEXT_HANDLE_ERROR** is generated if *context* is not a valid context.

HISTORY

cgGetFirstState was introduced in Cg 1.4.

SEE ALSO

*cgGetNextState, cgGetNamedState, cgIsState*

### 4.101 cgGetFirstStateAssignment

**NAME**

*cgGetFirstStateAssignment* - get the first state assignment in a pass

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGstateassignment cgGetFirstStateAssignment( CGpass pass );
```

**PARAMETERS**

*pass*

The pass from which to retrieve the first state assignment.

**RETURN VALUES**

Returns the first *CGstateassignment* object in *pass*.

Returns **NULL** if *pass* contains no state assignments or an error occurs.

**DESCRIPTION**

cgGetFirstStateAssignment is used to begin iteration over all of the state assignment contained within a pass. See *cgGetNextStateAssignment* for more information.

**EXAMPLES**

*to-be-written*

**ERRORS**

**CG_INVALID_PASS_HANDLE_ERROR** is generated if *pass* is not a valid pass.

**HISTORY**

cgGetFirstStateAssignment was introduced in Cg 1.4.

**SEE ALSO**

cgGetNextStateAssignment, cgIsStateAssignment
4.102 cgGetFirstStructParameter

NAME
cgGetFirstStructParameter - get the first child parameter from a struct parameter

SYNOPSIS
#include <Cg/cg.h>
CGparameter cgGetFirstStructParameter( CGparameter param );

PARAMETERS
- param
  Specifies the struct parameter. This parameter must be of type CG_STRUCT (returned by cgGetParameterType).

RETURN VALUES
- Returns a handle to the first member parameter.
- Returns NULL if param is not a struct or if some other error occurs.

DESCRIPTION
cgGetFirstStructParameter returns the first member parameter of a struct parameter. The rest of the members may be retrieved from the first member by iterating with cgGetNextParameter.

EXAMPLES
to-be-written

ERRORS
- CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
- CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a struct parameter.

HISTORY
cgGetFirstStructParameter was introduced in Cg 1.1.

SEE ALSO
cgGetNextParameter, cgGetFirstParameter, cgGetFirstUniformBufferParameter

4.103 cgGetFirstTechnique

NAME
cgGetFirstTechnique - get the first technique in an effect

SYNOPSIS
#include <Cg/cg.h>

CGtechnique cgGetFirstTechnique( CGeffect effect );
PARAMETERS

effect

The effect from which to retrieve the first technique.

RETURN VALUES

Returns the first CGtechnique object in effect.

Returns NULL if effect contains no techniques or an error occurs.

DESCRIPTION

.cgGetFirstTechnique is used to begin iteration over all of the techniques contained within a effect. See .cgGetNextTechnique for more information.

EXAMPLES

Iterating through the techniques in an effect:

```c
cGeffect effect = cgCreateEffectFromFile(context, cgfx_filename, NULL);
cGtechnique technique = cgGetFirstTechnique(effect);
while (technique) {
    // Do something with each technique
    technique = cgGetNextTechnique(technique);
}
```

ERRORS

.CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY

cgGetFirstTechnique was introduced in Cg 1.4.

SEE ALSO

cgGetFirstTechniqueAnnotation, cgGetNamedTechniqueAnnotation, cgGetNextTechnique, cgGetNamedTechnique, cgValidateTechnique, cgGetPassTechnique, cgGetTechniqueEffect, cgGetTechniqueName, cgIsTechnique

4.104 cgGetFirstTechniqueAnnotation

NAME

cgGetFirstTechniqueAnnotation - get the first annotation of a technique

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstTechniqueAnnotation( CGtechnique tech );
```

PARAMETERS

   tech

   The technique from which to retrieve the annotation.
RETURN VALUES

Returns the first annotation in the given technique.

Returns NULL if the technique has no annotations or an error occurs.

DESCRIPTION

The annotations associated with a technique can be retrieved using cgGetFirstTechniqueAnnotation. The remainder of the technique’s annotations can be discovered by iterating through the parameters, calling cgGetNextAnnotation to get to the next one.

EXAMPLES

CGannotation ann = cgGetFirstTechniqueAnnotation( technique );
while( ann )
{
    /* do something with ann */
    ann = cgGetNextAnnotation( ann );
}

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetFirstTechniqueAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetNamedTechniqueAnnotation, cgGetNextAnnotation

4.105 cgGetFirstUniformBufferParameter

NAME

cgGetFirstUniformBufferParameter - get the first child parameter from a uniform buffer parameter

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetFirstUniformBufferParameter( CGparameter param );

PARAMETERS

    param

    Specifies the uniform buffer parameter. This parameter must be of type CG_UNIFORMBUFFER (returned by cgGetParameterType).

RETURN VALUES

Returns a handle to the first member parameter.

Returns NULL if param is not a uniform buffer or if some other error occurs.
DESCRIPTION

cgGetFirstUniformBufferParameter returns the first member parameter of a uniform buffer parameter. The rest of the members may be retrieved from the first member by iterating with cgGetNextParameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a uniform buffer parameter.

HISTORY

cgGetFirstUniformBufferParameter was introduced in Cg 3.1.

SEE ALSO

cgGetNextParameter, cgGetFirstParameter, cgGetFirstStructParameter

4.106 cgGetFloatAnnotationValues

NAME

.cgGetFloatAnnotationValues - get a float-valued annotation’s values

SYNOPSIS

#include <Cg/cg.h>

const float * cgGetFloatAnnotationValues( CGannotation ann,
                                          int * nvalues );

PARAMETERS

ann

The annotation from which the values will be retrieved.

nvalues

Pointer to integer where the number of returned values will be stored.

RETURN VALUES

Returns a pointer to an array of float values. The number of values in the array is returned via the nvalues parameter.

Returns NULL if no values are available. nvalues will be 0.

DESCRIPTION

cgGetFloatAnnotationValues allows the application to retrieve the value(s) of a floating-point typed annotation.

EXAMPLES

to-be-written
**ERRORS**

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if `ann` is not a valid annotation.

CG_INVALID_PARAMETER_ERROR is generated if `nvalues` is NULL.

**HISTORY**

cgGetFloatAnnotationValues was introduced in Cg 1.4.

**SEE ALSO**

cgGetAnnotationType, cgGetIntAnnotationValues, cgGetStringAnnotationValue, cgGetBoolAnnotationValues

### 4.107 cgGetFloatStateAssignmentValues

**NAME**

cgGetFloatStateAssignmentValues - get a float-valued state assignment’s values

**SYNOPSIS**

```c
#include <Cg/cg.h>

const float * cgGetFloatStateAssignmentValues( CGstateassignment sa,
                                             int * nvalues );
```

**PARAMETERS**

- `sa`
  - The state assignment from which the values will be retrieved.
- `nvalues`
  - Pointer to integer where the number of returned values will be stored.

**RETURN VALUES**

Returns a pointer to an array of `float` values. The number of values in the array is returned via the `nvalues` parameter. Returns `NULL` if an error occurs or if no values are available. `nvalues` will be 0 in the latter case.

**DESCRIPTION**

cgGetFloatStateAssignmentValues allows the application to retrieve the value(s) of a floating-point typed state assignment.

**EXAMPLES**

to-be-written

**ERRORS**

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if `sa` is not a valid state assignment.

CG_INVALID_PARAMETER_ERROR is generated if `nvalues` is NULL.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if `sa` is not a state assignment of a float type.
4.108  cgGetIntAnnotationValues

NAME

cgGetIntAnnotationValues - get an integer-valued annotation’s values

SYNOPSIS

```
#include <Cg/cg.h>

const int * cgGetIntAnnotationValues( CGannotation ann,
                                        int * nvalues );
```

PARAMETERS

- `ann`  
The annotation from which the values will be retrieved.
- `nvalues`  
  Pointer to integer where the number of returned values will be stored.

RETURN VALUES

Returns a pointer to an array of `int` values. The number of values in the array is returned via the `nvalues` parameter.

Returns `NULL` if no values are available. `nvalues` will be `0`.

DESCRIPTION

`cgGetIntAnnotationValues` allows the application to retrieve the value(s) of an int typed annotation.

EXAMPLES

to-be-written

ERRORS

- `CG_INVALID_ANNOTATION_HANDLE_ERROR` is generated if `ann` is not a valid annotation.
- `CG_INVALID_PARAMETER_ERROR` is generated if `nvalues` is `NULL`.

HISTORY

`cgGetIntAnnotationValues` was introduced in Cg 1.4.

SEE ALSO

`cgGetAnnotationType`, `cgGetFloatAnnotationValues`, `cgGetStringAnnotationValue`, `cgGetBoolAnnotationValues`
4.109 cgGetIntStateAssignmentValues

NAME
cgGetIntStateAssignmentValues - get an int-valued state assignment’s values

SYNOPSIS

```c
#include <Cg/cg.h>

const int * cgGetIntStateAssignmentValues( CGstateassignment sa,
                                          int * nvalues );
```

PARAMETERS

- `sa`: The state assignment from which the values will be retrieved.
- `nvalues`: Pointer to integer where the number of values returned will be stored.

RETURN VALUES

Returns a pointer to an array of `int` values. The number of values in the array is returned via the `nvalues` parameter. Returns `NULL` if an error occurs or if no values are available. `nvalues` will be 0 in the latter case.

DESCRIPTION

cgGetIntStateAssignmentValues allows the application to retrieve the value(s) of an integer typed state assignment.

EXAMPLES

to-be-written

ERRORS

- `CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.
- `CG_INVALID_PARAMETER_ERROR` is generated if `nvalues` is `NULL`.
- `CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR` is generated if `sa` is not a state assignment of an integer type.

HISTORY

cgGetIntStateAssignmentValues was introduced in Cg 1.4.

SEE ALSO

cgGetStateAssignmentState, cgGetStateType, cgGetFloatStateAssignmentValues, cgGetBoolStateAssignmentValues, cgGetStringStateAssignmentValue, cgGetProgramStateAssignmentValue, cgGetSamplerStateAssignmentValue, cgGetTextureStateAssignmentValue
4.110 cgGetLastErrorString

NAME
cgGetLastErrorString - get the current error condition

SYNOPSIS
#include <Cg/cg.h>
const char * cgGetLastErrorString( CGerror * error );

PARAMETERS
 error
     A pointer to a CGerror variable for returning the last error code.

RETURN VALUES
Returns the last error string.
Returns NULL if there was no error.
If error is not NULL, the last error code will be returned in the location specified by error. This is the same value
that would be returned by cgGetError.

DESCRIPTION
cgGetLastErrorString returns the current error condition and error condition string. It's similar to calling cgGetErrorString with the result of cgGetError. However in certain cases the error string may contain more information about
the specific error that last occurred than what cgGetErrorString would return.

EXAMPLES
CGerror error;
const char * errorString = cgGetLastErrorString( &error );

ERRORS
None.

HISTORY
cgGetLastErrorString was introduced in Cg 1.2.

SEE ALSO
cgGetError, cgGetErrorString

4.111 cgGetLastListing

NAME
cgGetLastListing - get the current listing text
SYNOPSIS

#include <Cg/cg.h>

const char *cgGetLastListing( CGcontext context );

PARAMETERS

context

The context handle.

RETURN VALUES

Returns a null-terminated string containing the current listing text.

Returns NULL if no listing text is available, or the listing text string is empty.

In all cases, the pointer returned by cgGetLastListing is only guaranteed to be valid until the next Cg entry point not related to error reporting is called. For example, calls to cgCreateProgram, cgCompileProgram, cgCreateEffect, or cgValidateTechnique will invalidate any previously-returned listing pointer.

DESCRIPTION

Each Cg context maintains a null-terminated string containing warning and error messages generated by the Cg compiler, state managers and the like. cgGetLastListing allows applications and custom state managers to query the listing text.

cgGetLastListing returns the current listing string for the given CGcontext. When a Cg runtime error occurs, applications can use the listing text from the appropriate context to provide the user with detailed information about the error.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetLastListing was introduced in Cg 1.1.

SEE ALSO

cgSetLastListing, cgCreateContext, cgSetErrorHandler

4.112 cgGetLockingPolicy

NAME

cgGetLockingPolicy - get locking policy

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetLockingPolicy( void );
PARAMETERS
None.

RETURN VALUES
Returns an enumerant indicating the current locking policy.

DESCRIPTION
cgGetLockingPolicy returns an enumerant indicating the current locking policy for the library. See cgSetLockingPolicy for more information.

EXAMPLES
CGenum currentLockingPolicy = cgGetLockingPolicy();

ERRORS
None.

HISTORY
cgGetLockingPolicy was introduced in Cg 2.0.

SEE ALSO
cgSetLockingPolicy

4.113 cgGetMatrixParameter

NAME
cgGetMatrixParameter - gets the values from a matrix parameter

SYNOPSIS
#include <Cg/cg.h>

/* TYPE is int, float, or double */

void cgGetMatrixParameter{ifd}{rc}( CGparameter param,
                              TYPE * matrix );

PARAMETERS

  param
    The parameter from which the values will be returned.

  matrix
    An array of values into which the parameter’s value will be written. The array must have size equal to the
    number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES
None.
DESCRIPTION

The cgGetMatrixParameter functions retrieve the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take int, float or double values signified by the i, f or d in the function name.

There are versions of each function that specify the order in which matrix values should be written to the array. Row-major copying is indicated by r, while column-major is indicated by c.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

The cgGetMatrixParameter functions were introduced in Cg 1.4.

SEE ALSO

cgGetMatrixParameterRows, cgGetMatrixParameterColumns, cgGetMatrixParameterOrder, cgGetMatrixParameterValues

4.114 cgGetMatrixParameterdc

NAME

cgGetMatrixParameterdc - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cg.h>

void cgGetMatrixParameterdc(CGparameter param,
                             double * matrix);

PARAMETERS

    param
    The parameter from which the values will be returned.

    matrix
    An array of doubles into which the matrix values will be written. The array must have size equal to the number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGetMatrixParameterdc retrieves the values of the given matrix parameter using column-major ordering.
EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetMatrixParameterdc was introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.115 cgGetMatrixParameterdr

NAME

cgGetMatrixParameterdr - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cg.h>

void cgGetMatrixParameterdr( CGparameter param,
                              double * matrix );

PARAMETERS

param

The parameter from which the values will be returned.

matrix

An array of doubles into which the matrix values will be written. The array must have size equal to the
number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGetMatrixParameterdr retrieves the values of the given matrix parameter using row-major ordering.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
HISTORY

CgGetMatrixParameterDr was introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.116 CgGetMatrixParameterFc

NAME

cgGetMatrixParameterFc - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cg.h>

void cgGetMatrixParameterFc( CGparameter param,
                              float * matrix );

PARAMETERS

param

The parameter from which the values will be returned.

matrix

An array of floats into which the matrix values will be written. The array must have size equal to the number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGetMatrixParameterFc retrieves the values of the given matrix parameter using column-major ordering.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetMatrixParameterFc was introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues
4.117  cgGetMatrixParameterfr

NAME

cgGetMatrixParameterfr - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cg.h>

void cgGetMatrixParameterfr( CGparameter param,
                             float * matrix );

PARAMETERS

  param
     The parameter from which the values will be returned.

  matrix
     An array of floats into which the matrix values will be written. The array must have size equal to the
     number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGetMatrixParameterfr retrieves the values of the given matrix parameter using row-major ordering.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetMatrixParameterfr was introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.118  cgGetMatrixParameteric

NAME

cgGetMatrixParameteric - get the values from a matrix parameter
SYNOPSIS

```
#include <Cg/cg.h>

void cgGetMatrixParameteric( CGparameter param,
                               int * matrix );
```

PARAMETERS

- **param**
  The parameter from which the values will be returned.

- **matrix**
  An array of ints into which the matrix values will be written. The array must have size equal to the number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

`cgGetMatrixParameteric` retrieves the values of the given matrix parameter using column-major ordering.

EXAMPLES

`to-be-written`

ERRORS

- **CG_NOT_MATRIX_PARAM_ERROR** is generated if `param` is not a matrix parameter.
- **CG_INVALID_PARAM_HANDLE_ERROR** is generated if `param` is not a valid parameter.

HISTORY

`cgGetMatrixParameteric` was introduced in Cg 1.4.

SEE ALSO

`cgGetParameterRows`, `cgGetParameterColumns`, `cgGetMatrixParameter`, `cgGetParameterValues`

### 4.119 cgGetMatrixParameterir

NAME

`cgGetMatrixParameterir` - get the values from a matrix parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgGetMatrixParameterir( CGparameter param,
                             int * matrix );
```
PARAMETERS

param
The parameter from which the values will be returned.

matrix
An array of ints into which the matrix values will be written. The array must have size equal to the number of rows in the matrix times the number of columns in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGetMatrixParameterIR retrieves the values of the given matrix parameter using row-major ordering.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetMatrixParameterIR was introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.120 cgGetMatrixParameterOrder

NAME

cgGetMatrixParameterOrder - get the row or column order of a matrix parameter

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetMatrixParameterOrder( CGparameter param );

PARAMETERS

param
The parameter.

RETURN VALUES

Returns CG_ROW_MAJOR for a row-major matrix parameter.
Returns CG_COLUMN_MAJOR for a column-major matrix parameter.
Returns CG_UNKNOWN for a parameter that is not a matrix.
DESCRIPTION

cgGetMatrixParameterOrder returns the row or column order of a matrix parameter.

The Cg compiler supports #pragma pack_matrix(row_major) or #pragma pack_matrix(column_major) for specifying the order of matrix parameters. Row-major order is the Cg default.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter or param is not a matrix parameter.

HISTORY

cgGetMatrixParameterOrder was introduced in Cg 2.2.

SEE ALSO

cgGetMatrixParameter, cgGetMatrixSize

4.121 cgGetMatrixSize

NAME

cgGetMatrixSize - get the size of one dimension of an array parameter

SYNOPSIS

#include <Cg/cg.h>

void cgGetMatrixSize( CGtype type,
    int * nrows,
    int * ncols );

PARAMETERS

type

The type enumerant.

nrows

A pointer to the location where the number of rows that type has will be written.

ncols

A pointer to the location where the number of columns that type has will be written.

RETURN VALUES

None.
**DESCRIPTION**

`cgGetMatrixSize` writes the number of rows and columns contained by the specified matrix type into `nrows` and `ncols` locations respectively. If `type` is not a matrix enumerant type, 0 is written as both the rows and columns size.

Contrast this routine with `cgGetTypeSizes` where the number of rows and columns will be set to 1 row and 1 column for both scalar and non-numeric types but for vector types, the number of rows and columns will be set to 1 row and $N$ columns where $N$ is the number of components in the vector.

**EXAMPLES**

`to-be-written`

**ERRORS**

None.

**HISTORY**

`cgGetMatrixSize` was introduced in Cg 1.5.

**SEE ALSO**

`cgGetMatrixParameterOrder`, `cgGetArrayTotalSize`, `cgGetArrayDimension`, `cgGetArrayParameter`, `cgGetTypeSizes`

4.122  **cgGetNamedEffect**

**NAME**

`cgGetNamedEffect` - get an effect from a context by name

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGeffect cgGetNamedEffect( CGcontext context, 
                         const char * name );
```

**PARAMETERS**

- `context`
  - The context from which to retrieve the effect.
- `name`
  - The name of the effect to retrieve.

**RETURN VALUES**

Returns the named effect if found.

Returns NULL if `context` has no effect corresponding to `name` or if an error occurs.

**DESCRIPTION**

The effects in a context can be retrieved directly by name using `cgGetNamedEffect`. The effect names can be discovered by iterating through the context’s effects (see `cgGetFirstEffect` and `cgGetNextEffect`) and calling `cgGetEffectName` for each.
EXAMPLES

/* get "simpleEffect" from context */
CGeffect effect = cgGetNamedEffect( context, "simpleEffect" );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetNamedEffect was introduced in Cg 1.5.

SEE ALSO

cgGetEffectName, cgSetEffectName, cgGetFirstEffect, cgGetNextEffect

4.123 cgGetNamedEffectAnnotation

NAME

cgGetNamedEffectAnnotation - get an effect annotation by name

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNamedEffectAnnotation( CGeffect effect,
                                          const char * name );

PARAMETERS

effect
   The effect from which to retrieve the annotation.

name
   The name of the annotation to retrieve.

RETURN VALUES

Returns the named annotation.

Returns NULL if the effect has no annotation corresponding to name.

DESCRIPTION

The annotations associated with an effect can be retrieved directly by name using cgGetNamedEffectAnnotation. The names of a effect’s annotations can be discovered by iterating through the annotations (see cgGetFirstEffectAnnotation and cgGetNextAnnotation), calling cgGetAnnotationName for each one in turn.

EXAMPLES

/* fetch annotation "Apple" from CGeffect effect */
CGannotation ann = cgGetNamedEffectAnnotation( effect, "Apple" );
ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

CG_INVALID_POINTER_ERROR is generated if name is NULL.

HISTORY

cgGetNamedEffectAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetFirstEffectAnnotation, cgGetNextAnnotation, cgGetAnnotationName

### 4.124 cgGetNamedEffectParameter

**NAME**

cgGetNamedEffectParameter - get an effect parameter by name

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGparameter cgGetNamedEffectParameter( CGeffect effect,
                                        const char * name );
```

**PARAMETERS**

- **effect**
  - The effect from which to retrieve the parameter.

- **name**
  - The name of the parameter to retrieve.

**RETURN VALUES**

Returns the named parameter from the effect.

Returns NULL if the effect has no parameter corresponding to name.

**DESCRIPTION**

The parameters of a effect can be retrieved directly by name using cgGetNamedEffectParameter. The names of the parameters in a effect can be discovered by iterating through the effect's parameters (see cgGetFirstEffectParameter and cgGetNextParameter), calling cgGetParameterName for each one in turn.

The given name may be of the form “foo.bar[2]”, which retrieves the second element of the array “bar” in a structure named “foo”.

**EXAMPLES**

to-be-written

**ERRORS**

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.
HISTORY

cgGetNamedEffectParameter was introduced in Cg 1.4.

SEE ALSO

cgGetFirstEffectParameter, cgGetNextParameter, cgGetParameterName, cgGetNamedParameter

4.125 cgGetNamedEffectUniformBuffer

NAME

cgGetNamedEffectUniformBuffer - get a uniform buffer parameter from an effect by block name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedEffectUniformBuffer( CGeffect effect,
                                          const char *blockName );

PARAMETERS

effect
   The effect from which to retrieve the uniform buffer parameter.

blockName
   The block name of the uniform buffer parameter to retrieve.

RETURN VALUES

Returns the uniform buffer parameter with the matching block name from the effect.

Returns NULL if the effect has no parameter corresponding to blockName or an error occurs.

DESCRIPTION

The uniform buffer parameters of an effect can be retrieved directly by block name using cgGetNamedEffectUniformBuffer. The block names of the uniform buffer parameters in an effect can be discovered by iterating through the effect’s parameters with cgGetFirstParameter and cgGetNextParameter, calling cgGetUniformBufferBlockName for each uniform buffer parameter in turn.

EXAMPLES

If the file buf.fx contains this shader:

    uniform myBuf {
        float4 var;
    } a : BUFFER;

    float4 vertex() : POSITION
    {
        return float4(a.var.r, a.var.q, a.var.b, 1.0);
    }

and if effect refers to the CGeffect created from buf.fx, then calling:
CGparameter p = cgGetNamedEffectUniformBuffer(effect, "myBuf");

results in p containing the CGparameter associated with variable a.

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

CG_INVALID_POINTER_ERROR is generated if blockName is NULL.

HISTORY

cgGetNamedEffectUniformBuffer was introduced in Cg 3.1.

SEE ALSO

cgIsParameter, cgGetFirstParameter, cgGetNextParameter, cgGetUniformBufferBlockName

4.126 cgGetNamedParameter

NAME

cgGetNamedParameter - get a program parameter by name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedParameter( CGprogram program,
                                       const char * name );

PARAMETERS

program
   The program from which to retrieve the parameter.

name
   The name of the parameter to retrieve.

RETURN VALUES

Returns the named parameter from the program.

Returns NULL if the program has no parameter corresponding to name.

DESCRIPTION

The parameters of a program can be retrieved directly by name using cgGetNamedParameter. The names of the parameters in a program can be discovered by iterating through the program’s parameters (see cgGetNextParameter), calling cgGetParameterName for each one in turn.

The parameter name does not have to be complete name for a leaf node parameter. For example, if you have Cg program with the following parameters:

struct FooStruct
{
    float4 A;
    float4 B;


```c
struct BarStruct {
    FooStruct Foo[2];
};

void main(BarStruct Bar[3]) {
    /* ... */
}
```

The following leaf-node parameters will be generated:

- `Bar[0].Foo[0].A`
- `Bar[0].Foo[0].B`
- `Bar[0].Foo[1].A`
- `Bar[0].Foo[1].B`
- `Bar[1].Foo[0].A`
- `Bar[1].Foo[0].B`
- `Bar[1].Foo[1].A`
- `Bar[1].Foo[1].B`
- `Bar[2].Foo[0].A`
- `Bar[2].Foo[0].B`
- `Bar[2].Foo[1].A`
- `Bar[2].Foo[1].B`

A handle to any of the non-leaf arrays or structs can be directly obtained by using the appropriate name. The following are a few examples of names valid names that may be used with `cgGetNamedParameter` given the above Cg program:

- "Bar"
- "Bar[1]"
- "Bar[1].Foo"
- "Bar[1].Foo[0]"
- "Bar[1].Foo[0].B"

**EXAMPLES**

to-be-written

**ERRORS**

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.

**HISTORY**

`cgGetNamedParameter` was introduced in Cg 1.1.

**SEE ALSO**

`cgIsParameter`, `cgGetFirstParameter`, `cgGetNextParameter`, `cgGetArrayParameter`, `cgGetParameterName`
4.127 cgGetNamedParameterAnnotation

NAME

cgGetNamedParameterAnnotation - get a parameter annotation by name

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNamedParameterAnnotation( CGparameter param, const char * name );

PARAMETERS

param

The parameter from which to retrieve the annotation.

name

The name of the annotation to retrieve.

RETURN VALUES

Returns the named annotation.

Returns NULL if the parameter has no annotation corresponding to name.

DESCRIPTION

The annotations associated with a parameter can be retrieved directly by name using cgGetNamedParameterAnnotation. The names of a parameter's annotations can be discovered by iterating through the annotations (see cgGetFirstParameterAnnotation and cgGetNextAnnotation), calling cgGetAnnotationName for each one in turn.

EXAMPLES

/* fetch annotation "Apple" from CGparameter param */
CGannotation ann = cgGetNamedParameterAnnotation( param, "Apple" );

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetNamedParameterAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetFirstParameterAnnotation, cgGetNextAnnotation, cgGetAnnotationName

4.128 cgGetNamedPass

NAME

cgGetNamedPass - get a technique pass by name
SYNOPSIS

#include <Cg/cg.h>

CGpass cgGetNamedPass( CGtechnique tech,
                       const char * name );

PARAMETERS

  tech
   The technique from which to retrieve the pass.

  name
   The name of the pass to retrieve.

RETURN VALUES

Returns the named pass from the technique.

Returns NULL if the technique has no pass corresponding to name.

DESCRIPTION

The passes of a technique can be retrieved directly by name using cgGetNamedPass. The names of the passes in a technique can be discovered by iterating through the technique’s passes (see cgGetFirstPass and cgGetNextPass), calling cgGetPassName for each one in turn.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetNamedPass was introduced in Cg 1.4.

SEE ALSO

cgGetFirstPass, cgGetNextPass, cgGetPassName

4.129 cgGetNamedPassAnnotation

NAME

cgGetNamedPassAnnotation - get a pass annotation by name

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNamedPassAnnotation( CGpass pass,
                                       const char * name );
PARAMETERS

pass
The pass from which to retrieve the annotation.

name
The name of the annotation to retrieve.

RETURN VALUES

Returns the named annotation.

Returns NULL if the pass has no annotation corresponding to name.

DESCRIPTION

The annotations associated with a pass can be retrieved directly by name using cgGetNamedPassAnnotation. The names of a pass’s annotations can be discovered by iterating through the annotations (see cgGetFirstPassAnnotation and cgGetNextAnnotation), calling cgGetAnnotationName for each one in turn.

EXAMPLES

/* fetch annotation "Apple" from CGpass pass */
CGannotation ann = cgGetNamedPassAnnotation( pass, "Apple" );

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY

cgGetNamedPassAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetFirstPassAnnotation, cgGetNextAnnotation, cgGetAnnotationName

4.130  cgGetNamedProgramAnnotation

NAME

cgGetNamedProgramAnnotation - get a program annotation by name

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNamedProgramAnnotation( CGprogram program,
const char * name );

PARAMETERS

program
The program from which to retrieve the annotation.

name
The name of the annotation to retrieve.
RETURN VALUES

Returns the named annotation.

Returns NULL if the program has no annotation corresponding to name.

DESCRIPTION

The annotations associated with a program can be retrieved directly by name using cgGetNamedProgramAnnotation. The names of a program’s annotations can be discovered by iterating through the annotations (see cgGetFirstProgramAnnotation and cgGetNextAnnotation), calling cgGetAnnotationName for each one in turn.

EXAMPLES

/* fetch annotation "Apple" from CGprogram program */
CGannotation ann = cgGetNamedProgramAnnotation( program, "Apple" );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetNamedProgramAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetFirstProgramAnnotation, cgGetNextAnnotation, cgGetAnnotationName

4.131 cgGetNamedProgramParameter

NAME

cgGetNamedProgramParameter - get a program parameter by name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedProgramParameter( CGprogram program,
                                      CGenum name_space,
                                      const char * name );

PARAMETERS

  program
  The program from which to retrieve the parameter.

  name_space
  Specifies the namespace of the parameter to iterate through. Currently CG_PROGRAM and CG_GLOBAL are supported.

  name
  Specifies the name of the parameter to retrieve.
RETURN VALUES

Returns the named parameter from the program.
Returns NULL if the program has no parameter corresponding to name.

DESCRIPTION

cgGetNamedProgramParameter is essentially identical to cgGetNamedParameter except it limits the search of the parameter to the name space specified by name_space.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetNamedProgramParameter was introduced in Cg 1.2.

SEE ALSO

cgGetNamedParameter

4.132 cgGetNamedProgramUniformBuffer

NAME

cgGetNamedProgramUniformBuffer - get a program uniform buffer parameter by block name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedProgramUniformBuffer( CGprogram program,
                                          const char *blockName );

PARAMETERS

  program
    The program from which to retrieve the uniform buffer parameter.

  blockName
    The block name of the uniform buffer parameter to retrieve.

RETURN VALUES

Returns the uniform buffer parameter with the matching block name from the program.
Returns NULL if the program has no parameter corresponding to blockName or an error occurs.
DESCRIPTION

The uniform buffer parameters of a program can be retrieved directly by block name using `cgGetNamedProgramUniformBuffer`. The block names of the uniform buffer parameters in a program can be discovered by iterating through the program’s parameters with `cgGetFirstParameter` and `cgGetNextParameter`, calling `cgGetUniformBufferBlockName` for each uniform buffer parameter in turn.

EXAMPLES

If the file `buf.cg` contains this shader:

```c
uniform myBuf {
    float4 var;
} a : BUFFER;

float4 vertex() : POSITION {
    return float4(a.var.r, a.var.g, a.var.b, 1.0);
}
```

and if `program` refers to the `CGprogram` created from `buf.cg`, then calling:

```c
CGparameter p = cgGetNamedProgramUniformBuffer(program, "myBuf");
```

results in `p` containing the `CGparameter` associated with variable `a`.

ERRORS

- `CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.
- `CG_INVALID_POINTER_ERROR` is generated if `blockName` is NULL.

HISTORY

`cgGetNamedProgramUniformBuffer` was introduced in Cg 3.1.

SEE ALSO

`cgIsParameter`, `cgGetFirstParameter`, `cgGetNextParameter`, `cgGetUniformBufferBlockName`

4.133 cgGetNamedSamplerState

NAME

cgGetNamedSamplerState - get a sampler state by name

SYNOPSIS

```c
#include <Cg/cg.h>

CGstate cgGetNamedSamplerState( CGcontext context,
                                const char * name );
```

PARAMETERS

- context
  
The context from which to retrieve the named sampler state.
- name
The name of the state to retrieve.

**RETURN VALUES**

Returns the named sampler state.

Returns **NULL** if **context** is invalid or if **context** has no sampler states corresponding to **name**.

**DESCRIPTION**

The sampler states associated with a context, as specified with a **sampler_state** block in an effect file, can be retrieved directly by name using **cgGetNamedSamplerState**.

**EXAMPLES**

*to-be-written*

**ERRORS**

**CG_INVALID_CONTEXT_HANDLE_ERROR** is generated if **context** is not a valid context.

**CG_INVALID_PARAMETER_ERROR** is generated if **name** is **NULL**.

**HISTORY**

**cgGetNamedSamplerState** was introduced in Cg 1.4.

**SEE ALSO**

*cgCreateArraySamplerState, cgCreateSamplerState, cgGetFirstSamplerState, cgSetSamplerState*

### 4.134 cgGetNamedSamplerStateAssignment

**NAME**

**cgGetNamedSamplerStateAssignment** - get a sampler state assignment by name

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGStateAssignment cgGetNamedSamplerStateAssignment( CGparameter param,
                                                        const char * name );
```

**PARAMETERS**

- **param**
  
  The sampler parameter from which to retrieve the sampler state assignment.

- **name**
  
  The name of the state assignment to retrieve.

**RETURN VALUES**

Returns the named sampler state assignment.

Returns **NULL** if the pass has no sampler state assignment corresponding to **name**.
DESCRIPTION

The sampler state assignments associated with a `sampler` parameter, as specified with a `sampler_state` block in an effect file, can be retrieved directly by name using `cgGetNamedSamplerStateAssignment`. The names of the sampler state assignments can be discovered by iterating through the sampler’s state assignments (see `cgGetFirstSamplerStateAssignment` and `cgGetNextStateAssignment`), calling `cgGetSamplerStateAssignmentState` then `cgGetStateName` for each one in turn.

EXAMPLES

to-be-written

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

HISTORY

cgGetNamedSamplerStateAssignment was introduced in Cg 1.4.

SEE ALSO

cgIsStateAssignment, cgGetFirstSamplerStateAssignment, cgGetNextStateAssignment, cgGetStateName

4.135 cgGetNamedState

NAME

cgGetNamedState - get a context state by name

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgGetNamedState( CGcontext context, 
                          const char * name );
```

PARAMETERS

- `context`
  - The context from which to retrieve the state.
- `name`
  - The name of the state to retrieve.

RETURN VALUES

Returns the named state from the context.
Returns `NULL` if the context has no state corresponding to `name`.

DESCRIPTION

The states of a context can be retrieved directly by name using `cgGetNamedState`. The names of the states in a context can be discovered by iterating through the context’s states (see `cgGetFirstState` and `cgGetNextState`), calling `cgGetStateName` for each one in turn.
EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if name is NULL.

HISTORY
cgGetNamedState was introduced in Cg 1.4.

SEE ALSO
cgCreateState, cgGetFirstState, cgGetNextState, cgGetStateEnumerantName, cgGetStateEnumerantValue, cgGetStateName, cgGetStateType, cgIsState

4.136 cgGetNamedStateAssignment

NAME
cgGetNamedStateAssignment - get a pass state assignment by name

SYNOPSIS

#include <Cg/cg.h>

CGstateassignment cgGetNamedStateAssignment( CGpass pass,
    const char * name );

PARAMETERS

pass
The pass from which to retrieve the state assignment.

name
The name of the state assignment to retrieve.

RETURN VALUES

Returns the named state assignment from the pass.

Returns NULL if the pass has no state assignment corresponding to name.

DESCRIPTION

The state assignments of a pass can be retrieved directly by name using cgGetNamedStateAssignment. The names of the state assignments in a pass can be discovered by iterating through the pass’s state assignments (see cgGetFirstStateAssignment and cgGetNextStateAssignment), calling cgGetStateAssignmentState then cgGetStateName for each one in turn.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.
HISTORY

cgGetNamedStateAssignment was introduced in Cg 1.4.

SEE ALSO

cgIsStateAssignment, cgGetFirstStateAssignment, cgGetNextStateAssignment, cgGetStateAssignmentState, cgGetStateName

4.137 cgGetNamedStructParameter

NAME

cgGetNamedStructParameter - get member of a struct parameter by name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedStructParameter( CGparameter param,
                                           const char * name );

PARAMETERS

param

The struct parameter from which to retrieve the member parameter.

name

The name of the member parameter to retrieve.

RETURN VALUES

Returns the member parameter from the given struct.

Returns NULL if the struct has no member parameter corresponding to name.

DESCRIPTION

The member parameters of a struct parameter may be retrieved directly by name using cgGetNamedStructParameter. The names of the parameters in a struct may be discovered by iterating through the struct’s member parameters (see cgGetFirstStructParameter and cgGetNextParameter), and calling cgGetParameterName for each one in turn.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a struct parameter.

HISTORY

cgGetNamedStructParameter was introduced in Cg 1.2.

SEE ALSO

cgGetFirstStructParameter, cgGetNextParameter, cgGetParameterName
4.138  cgGetNamedSubParameter

NAME

cgGetNamedSubParameter - gets a “shallow” or “deep” member from an aggregate parameter (i.e. array, struct, uniform buffer, etc.)

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedSubParameter( CGparameter param, 
const char * name );

PARAMETERS

param
Aggregate parameter from which to retrieve a member.

name
Name of the desired member inside the aggregate parameter param.

RETURN VALUES

Returns the named member of param.

Returns NULL if param has no parameter corresponding to name or an error occurs.

DESCRIPTION

cgGetNamedSubParameter is a generalized parameter finding function that will retrieve parameters, including deep parameters, of an aggregate parameter type such as a array, struct, or uniform buffer.

EXAMPLES

CGparameter parent = cgGetNamedParameter( program, "someParameter" );
CGparameter deepChild = cgGetNamedSubParameter( parent, "foo.list[3].item" );

/* Note: 'deepChild' is the same parameter returned by:
   cgGetNamedParameter( program, "someParameter.foo.list[3].item" ); */

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetNamedSubParameter was introduced in Cg 1.5.

SEE ALSO

cgGetNamedParameter, cgGetNamedStructParameter, cgGetNamedUniformBufferParameter, cgGetArrayParameter
4.139 cgGetNamedTechnique

NAME

cgGetNamedTechnique - get an effect’s technique by name

SYNOPSIS

#include <Cg/cg.h>

CGtechnique cgGetNamedTechnique( CGeffect effect,
            const char * name );

PARAMETERS

effect
    The effect from which to retrieve the technique.

name
    The name of the technique to retrieve.

RETURN VALUES

Returns the named technique from the effect.

Returns NULL if the effect has no technique corresponding to name.

DESCRIPTION

The techniques of an effect can be retrieved directly by name using cgGetNamedTechnique. The names of the techniques in a effect can be discovered by iterating through the effect’s techniques (see cgGetFirstTechnique and cgGetNextTechnique), calling cgGetTechniqueName for each one in turn.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY

cgGetNamedTechnique was introduced in Cg 1.4.

SEE ALSO

.cgGetFirstTechnique, cgGetNextTechnique, cgGetTechniqueName

4.140 cgGetNamedTechniqueAnnotation

NAME

.cgGetNamedTechniqueAnnotation - get a technique annotation by name
SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNamedTechniqueAnnotation( CGtechnique tech,
                                          const char * name );

PARAMETERS

    tech
        The technique from which to retrieve the annotation.
    name
        The name of the annotation to retrieve.

RETURN VALUES

Returns the named annotation.
Returns NULL if the technique has no annotation corresponding to name.

DESCRIPTION

The annotations associated with a technique can be retrieved directly by name using cgGetNamedTechniqueAnnotation. The names of a technique’s annotations can be discovered by iterating through the annotations (see cgGetFirstTechniqueAnnotation and cgGetNextAnnotation), calling cgGetAnnotationName for each one in turn.

EXAMPLES

/* fetch annotation "Apple" from CGtechnique technique */
CGannotation ann = cgGetNamedTechniqueAnnotation( technique, "Apple" );

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetNamedTechniqueAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetFirstTechniqueAnnotation, cgGetNextAnnotation, cgGetAnnotationName

4.141 cgGetNamedUniformBufferParameter

NAME

cgGetNamedUniformBufferParameter - get member of a uniform buffer parameter by name

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetNamedUniformBufferParameter( CGparameter param,
                                              const char * name );
PARAMETERS

param
The uniform buffer from which to retrieve a member parameter.

name
The name of the desired member parameter inside uniform buffer param.

RETURN VALUES

Returns the member parameter from the given uniform buffer.
Returns NULL if the uniform buffer has no member parameter corresponding to name.

DESCRIPTION

The member parameters of a uniform buffer parameter may be retrieved directly by name using cgGetNamedUniformBufferParameter.

The names of the parameters in a uniform buffer may be discovered by iterating through the uniform buffer’s member parameters (see cgGetFirstUniformBufferParameter and cgGetNextParameter), and calling cgGetParameterName for each one in turn.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a uniform buffer parameter.
CG_INVALID_POINTER_ERROR is generated if name is NULL.

HISTORY

cgGetNamedUniformBufferParameter was introduced in Cg 3.1.

SEE ALSO

cgGetNamedStructParameter, cgGetNamedSubParameter, cgGetFirstUniformBufferParameter, cgGetNextParameter, cgGetParameterName

4.142 cgGetNamedUserType

NAME

cgGetNamedUserType - get enumerant associated with type name

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetNamedUserType( CGhandle handle, const char * name );
PARAMETERS

handle
The CGprogram or CGeffect in which the type is defined.

name
A string containing the case-sensitive type name.

RETURN VALUES

Returns the type enumerant associated with name.

Returns CG_UNKNOWN_TYPE if no such type exists.

DESCRIPTION

cgGetNamedUserType returns the enumerant associated with the named type defined in the construct associated with handle, which may be a CGprogram or CGeffect.

For a given type name, the enumerant returned by this entry point is guaranteed to be identical if called with either an CGeffect handle, or a CGprogram that is defined within that effect.

If two programs in the same context define a type using identical names and definitions, the associated enumerants are also guaranteed to be identical.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if handle is not a valid program or effect.

HISTORY

cgGetNamedUserType was introduced in Cg 1.2.

SEE ALSO

.cgGetUserType, cgGetType

4.143 cgGetNextAnnotation

NAME

cgGetNextAnnotation - iterate through annotations

SYNOPSIS

#include <Cg/cg.h>

CGannotation cgGetNextAnnotation( CGannotation ann );

PARAMETERS

ann
The current annotation.
RETURN VALUES

Returns the next annotation in the sequence of annotations associated with the annotated object.

Returns NULL when ann is the last annotation.

DESCRIPTION

The annotations associated with a parameter, pass, technique, or program can be iterated over by using cgGetNextAnnotation.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each annotation will be visited exactly once.

EXAMPLES

CGAnnotation ann = cgGetFirstParameterAnnotation( param );
while ( ann )
{
    /* do something with ann */
    ann = cgGetNextAnnotation( ann );
}

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.

HISTORY

cgGetNextAnnotation was introduced in Cg 1.4.

SEE ALSO

cgGetFirstParameterAnnotation, cgGetFirstPassAnnotation, cgGetFirstTechniqueAnnotation, cgGetFirstProgramAnnotation, cgGetNamedParameterAnnotation, cgGetNamedPassAnnotation, cgGetNamedTechniqueAnnotation, cgGetNamedProgramAnnotation, cgIsAnnotation

4.144 cgGetNextEffect

NAME

cgGetNextEffect - iterate through effects in a context

SYNOPSIS

#include <Cg/cg.h>

CGEffect cgGetNextEffect( CGEffect effect );

PARAMETERS

effect

The current effect.

RETURN VALUES

Returns the next effect in the context’s internal sequence of effects.

Returns NULL when effect is the last effect in the context.
DESCRIPTION
The effects within a context can be iterated over with \texttt{cgGetNextEffect}.
Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each effect will be visited exactly once. No guarantees can be made if effects are created or deleted during iteration.

EXAMPLES
\begin{verbatim}
CGeffect effect = cgGetFirstEffect( context );
while( effect )
{ /* do something with effect */
    effect = cgGetNextEffect( effect );
}
\end{verbatim}

ERRORS
\texttt{CG_INVALID_EFFECT_HANDLE_ERROR} is generated if \texttt{effect} is not a valid effect.

HISTORY
cgGetNextEffect was introduced in Cg 1.4.

SEE ALSO
cgGetFirstEffect

4.145 \texttt{cgGetNextLeafParameter}

NAME
cgGetNextLeafParameter - get the next leaf parameter in a program or effect

SYNOPSIS
\begin{verbatim}
#include <Cg/cg.h>

CGparameter cgGetNextLeafParameter( CGparameter param );
\end{verbatim}

PARAMETERS
\begin{itemize}
  \item \texttt{param}
    \begin{itemize}
      \item The current leaf parameter.
    \end{itemize}
\end{itemize}

RETURN VALUES
Returns the next leaf \texttt{CGparameter} object.
Returns \texttt{NULL} if \texttt{param} is invalid or if the program or effect from which the iteration started does not have any more leaf parameters.

DESCRIPTION
cgGetNextLeafParameter returns the next leaf parameter (not struct or array parameters) following a given leaf parameter.
In a similar manner, the leaf parameters in an effect can be iterated over starting with a call to \texttt{cgGetFirstLeafEffect-Parameter}.
EXAMPLES

```c
CGparameter leaf = cgGetFirstLeafParameter( program );
while(leaf)
{
    /* Do stuff with leaf */
    leaf = cgGetNextLeafParameter( leaf );
}
```

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

HISTORY

`cgGetNextLeafParameter` was introduced in Cg 1.1.

SEE ALSO

`cgGetFirstLeafParameter`, `cgGetFirstLeafEffectParameter`

4.146 cgGetNextParameter

NAME

`cgGetNextParameter` - iterate through a program’s or effect’s parameters

SYNOPSIS

```c
#include <Cg/cg.h>

CGparameter cgGetNextParameter( CGparameter current );
```

PARAMETERS

- `current`
  
The current parameter.

RETURN VALUES

Returns the next parameter in the program or effect’s internal sequence of parameters.

Returns `NULL` when `current` is the last parameter in the program or effect.

DESCRIPTION

The parameters of a program or effect can be iterated over using `cgGetNextParameter` with `cgGetFirstParameter`, or `cgGetArrayParameter`.

Similarly, the parameters in an effect can be iterated over starting with a call to `cgGetFirstEffectParameter`.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each parameter will be visited exactly once.

EXAMPLES
void RecurseParams( CGparameter param )
{
    if(!param)
        return;
    do
    {
        switch(cgGetParameterType(param))
        {
            case CG_STRUCT :
                RecurseParams(cgGetFirstStructParameter(param));
                break;
            case CG_ARRAY :
                {
                    int ArraySize = cgGetArraySize(param, 0);
                    int i;
                    for(i=0; i < ArraySize; ++i)
                        RecurseParams(cgGetArrayParameter(param, i));
                }
                break;
            default:
                /* Do stuff to param */
        }
    } while((param = cgGetNextParameter(param)) != 0);
}

void RecurseParamsInProgram( CGprogram program )
{
    RecurseParams( cgGetFirstParameter( program ) );
}

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetNextParameter was introduced in Cg 1.1.

SEE ALSO

cgGetFirstParameter, cgGetFirstEffectParameter, cgGetFirstStructParameter, cgGetArrayParameter, cgGetParameterType

4.147 cgGetNextPass

NAME

cgGetNextPass - iterate through the passes in a technique

SYNOPSIS
#include <Cg/cg.h>

CGpass cgGetNextPass( CGpass pass );

PARAMETERS
    pass
        The current pass.

RETURN VALUES
Returns the next pass in the technique’s internal sequence of passes.
Returns NULL when pass is the last pass in the technique.

DESCRIPTION
The passes within a technique can be iterated over using cgGetNextPass.
Passes are returned in the order defined in the technique.

EXAMPLES
CGpass pass = cgGetFirstPass( technique );
while( pass )
{
    /* do something with pass */
    pass = cgGetNextPass( pass );
}

ERRORS
CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY
cgGetNextPass was introduced in Cg 1.4.

SEE ALSO
cgGetFirstPass, cgGetNamedPass, cgIsPass

4.148 cgGetNextProgram

NAME
cgGetNextProgram - iterate through programs in a context

SYNOPSIS
#include <Cg/cg.h>

CGprogram cgGetNextProgram( CGprogram program );

PARAMETERS
    program
        The current program.
**RETURN VALUES**

Returns the next program in the context’s internal sequence of programs.

Returns **NULL** when **program** is the last program in the context.

**DESCRIPTION**

The programs within a context can be iterated over by using **cgGetNextProgram**.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each program will be visited exactly once. No guarantees can be made if programs are generated or deleted during iteration.

**EXAMPLES**

```c
CGprogram program = cgGetFirstProgram( context );
while( program )
{
    /* do something with program */
    program = cgGetNextProgram( program );
}
```

**ERRORS**

**CG_INVALID_PROGRAM_HANDLE_ERROR** is generated if **program** is not a valid program handle.

**HISTORY**

cgGetNextProgram was introduced in Cg 1.1.

**SEE ALSO**

`cgGetFirstProgram`, `cgCreateProgram`, `cgDestroyProgram`, `cgIsProgram`

### 4.149 cgGetNextState

**NAME**

cgGetNextState - iterate through states in a context

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGstate cgGetNextState( CGstate state );
```

**PARAMETERS**

- **state**
  
The current state.

**RETURN VALUES**

Returns the next state in the context’s internal sequence of states.

Returns **NULL** when **state** is the last state in the context.
DESCRIPTION

The states within a context can be iterated over using cgGetNextState.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each state will be visited exactly once. No guarantees can be made if states are created or deleted during iteration.

EXAMPLES

CGstate state = cgGetFirstState( context );
while( state ) {
    /* do something with state */
    state = cgGetNextState( state );
}

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgGetNextState was introduced in Cg 1.4.

SEE ALSO

cgGetFirstState, cgGetNamedState, cgCreateState, cgIsState

4.150 cgGetNextStateAssignment

NAME

cgGetNextStateAssignment - iterate through state assignments in a pass

SYNOPSIS

#include <Cg/cg.h>

CGstateassignment cgGetNextStateAssignment( CGstateassignment sa );

PARAMETERS

sa

The current state assignment.

RETURN VALUES

Returns the next state assignment in the pass’ internal sequence of state assignments.

Returns NULL when prog is the last state assignment in the pass.

DESCRIPTION

The state assignments within a pass can be iterated over by using cgGetNextStateAssignment.

State assignments are returned in the same order specified in the pass in the effect.
EXAMPLES

CGstateassignment sa = cgGetFirstStateAssignment( pass );
while( sa )
{
    /* do something with sa */
    sa = cgGetNextStateAssignment( sa );
}

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetNextStateAssignment was introduced in Cg 1.4.

SEE ALSO

cgGetFirstStateAssignment, cgGetNamedStateAssignment, cgIsStateAssignment

4.151 cgGetNextTechnique

NAME

cgGetNextTechnique - iterate through techniques in a effect

SYNOPSIS

#include <Cg/cg.h>

CGtechnique cgGetNextTechnique( CGtechnique tech );

PARAMETERS

    tech

        The current technique.

RETURN VALUES

Returns the next technique in the effect’s internal sequence of techniques.

Returns NULL when tech is the last technique in the effect.

DESCRIPTION

The techniques within a effect can be iterated over using cgGetNextTechnique.

Note that no specific order of traversal is defined by this mechanism. The only guarantee is that each technique will be visited exactly once.

EXAMPLES

CGtechnique tech = cgGetFirstTechnique( effect );
while( tech )
{
    /* do something with tech */
    tech = cgGetNextTechnique( tech );
}
4.152 cgGetNumConnectedToParameters

NAME

cgGetNumConnectedToParameters - gets the number of connected destination parameters

SYNOPSIS

#include <Cg/cg.h>

int cgGetNumConnectedToParameters( CGparameter param );

PARAMETERS

param
    The source parameter.

RETURN VALUES

Returns the number of destination parameters connected to param.

Returns 0 if an error occurs.

DESCRIPTION

cgGetNumConnectedToParameters returns the number of destination parameters connected to the source parameter param. It's primarily used with cgGetConnectedToParameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetNumConnectedToParameters was introduced in Cg 1.2.

SEE ALSO

cgConnectParameter, cgGetConnectedParameter, cgGetConnectedToParameter
4.153  cgGetNumDependentAnnotationParameters

NAME

cgGetNumDependentAnnotationParameters - get the number of effect parameters on which an annotation depends

SYNOPSIS

#include <Cg/cg.h>

int cgGetNumDependentAnnotationParameters( CGannotation ann );

PARAMETERS

ann

The annotation handle.

RETURN VALUES

Returns the number of parameters on which ann depends.

DESCRIPTION

Annotations in CgFX files may include references to one or more effect parameters on the right hand side of the annotation that are used for computing the annotation’s value. cgGetNumDependentAnnotationParameters returns the total number of such parameters. cgGetDependentAnnotationParameter can then be used to iterate over these parameters.

This information can be useful for applications that wish to cache the values of annotations so that they can determine which annotations may change as the result of changing a particular parameter’s value.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.

HISTORY

cgGetNumDependentAnnotationParameters was introduced in Cg 1.4.

SEE ALSO

cgGetDependentAnnotationParameter, cgGetNumDependentStateAssignmentParameters

4.154  cgGetNumDependentProgramArrayStateAssignmentParameters

NAME

cgGetNumDependentProgramArrayStateAssignmentParameters - get the number of parameters on which a state assignment’s value depends
SYNOPSIS

#include <Cg/cg.h>

int cgGetNumDependentProgramArrayStateAssignmentParameters( CGstateassignment sa );

PARAMETERS

sa

The state assignment handle.

RETURN VALUES

Returns the number of parameters on which sa depends.

Returns 0 if sa is not a program state assignment or an error occurs.

DESCRIPTION

State assignments in CgFX files may include references to an array indexed by an effect parameter (or expression) on the right hand side of the state assignment that is used for computing the state assignment’s value. Usually this array holds the compile statements of shader programs and by changing the index of the shader array, it’s possible to switch to a different program or profile on-the-fly.

Each compile statement in the array can depend on one or more effect parameters which are passed to the program in its parameter list. It is sometimes necessary for the application to query what those parameters are so values can be properly set to them.

Before you can query these parameters, you must know how many there are. cgGetNumDependentProgramArrayStateAssignmentParameters will return this number.

EXAMPLES

In CgFX file:

vertexshader Torus[4] =
{
    compile vp40 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile vp30 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile arbvp1 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) ),
    compile vp20 C8E6v_torus( LightPosition, EyePosition, ModelViewProj,
        float2( OuterRadius, InnerRadius ) )
};

pixelshader SpecSurf[4] =
{
    compile fp40 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube ),
    compile fp30 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1),
        float4(SpecularMaterial * LightColor, 1),
        normalMap, normalizeCube, normalizeCube ),
compile arbfp1 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1), float4(SpecularMaterial * LightColor, 1), normalMap, normalizeCube, normalizeCube ),

compile fp20 C8E4f_specSurf( Ambient, float4(DiffuseMaterial * LightColor, 1), float4(SpecularMaterial * LightColor, 1), normalMap, normalizeCube, normalizeCube )

int select = 0;

technique bumpdemo
{
    pass
    {
        VertexProgram = (Torus[select]);
        FragmentProgram = (SpecSurf[select]);
    }
}

In application:

int numParameters = cgGetNumDependentProgramArrayStateAssignmentParameters(stateAssignment);
for(int i = 0; i < numParameters; ++i) {
    CGparameter param = cgGetDependentProgramArrayStateAssignmentParameter(stateAssignment, i);

    /* Set value for 'param' */
}

For this example when select = 0 cgGetNumDependentProgramArrayStateAssignmentParameters will return 5 for VertexProgram and 8 for FragmentProgram.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetNumDependentProgramArrayStateAssignmentParameters was introduced in Cg 3.0.

SEE ALSO

cgGetDependentProgramArrayStateAssignmentParameter

4.155 cgGetNumDependentStateAssignmentParameters

NAME

cgGetNumDependentStateAssignmentParameters - get the number of effect parameters on which a state assignment depends

SYNOPSIS

#include <Cg/cg.h>

int cgGetNumDependentStateAssignmentParameters( CGstateassignment sa );
PARAMETERS

sa

The state assignment handle.

RETURN VALUES

Returns the number of parameters on which sa depends.

DESCRIPTION

State assignments in CgFX passes may include references to one or more effect parameters on the right hand side of the state assignment that are used for computing the state assignment’s value. cgGetNumDependentStateAssignmentParameters returns the total number of such parameters. cgGetDependentStateAssignmentParameter can then be used to iterate over these parameters.

This information can be useful for applications that wish to cache the values of state assignments for customized state management so that they can determine which state assignments may change as the result of changing a parameter’s value.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetNumDependentStateAssignmentParameters was introduced in Cg 1.4.

SEE ALSO

cgGetDependentStateAssignmentParameter, cgGetFirstStateAssignment, cgGetNamedStateAssignment, cgGetNumDependentAnnotationParameters

4.156 cgGetNumParentTypes

NAME

cgGetNumParentTypes - gets the number of parent types of a given type

SYNOPSIS

#include <Cg/cg.h>

int cgGetNumParentTypes( CGtype type );

PARAMETERS

type

The child type.

RETURN VALUES

Returns the number of parent types.

Returns 0 if there are no parents.
DESCRIPTION

cgGetNumParentTypes returns the number of parents from which type inherits.

A parent type is one from which the given type inherits, or an interface type that the given type implements.

Note that the current Cg language specification implies that a type may only have a single parent type – an interface implemented by the given type.

EXAMPLES

Given the type definitions:

```c
interface myiface {
    float4 eval(void);
};

struct mystruct : myiface {
    float4 value;
    float4 eval(void) { return value; }
};
```

mystruct has a single parent type, myiface.

ERRORS

None.

HISTORY

cgGetNumParentTypes was introduced in Cg 1.2.

SEE ALSO

cgGetParentType

4.157 cgGetNumProgramDomains

NAME

cgGetNumProgramDomains - get the number of domains in a combined program

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetNumProgramDomains( CGprogram program );
```

PARAMETERS

- **program**
  The combined program object to be queried.

RETURN VALUES

Returns the number of domains in the combined program program.

Returns 0 if an error occurs.
DESCRIPTION

`cgGetNumProgramDomains` returns the number of domains in a combined program. For example, if the combined program contained a vertex program and a fragment program, `cgGetNumProgramDomains` will return 2.

`cgGetNumProgramDomains` will always return 1 for a non-combined program.

EXAMPLES

```c
CGprogram combined = cgCombinePrograms2( prog1, prog2 );
int numDomains = cgGetNumProgramDomains( combined );
/* numDomains == 2 */
```

ERRORS

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.

HISTORY

`cgGetNumProgramDomains` was introduced in Cg 1.5.

SEE ALSO

`cgGetProfileDomain`, `cgGetProgramDomainProfile`, `cgGetProgramDomainProgram`

4.158 cgGetNumStateEnumerants

NAME

`cgGetNumStateEnumerants` - gets the number of enumerants associated with a state

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetNumStateEnumerants( CGstate state );
```

PARAMETERS

state

The state from which to retrieve the number of associated enumerants.

RETURN VALUES

Returns the number of enumerants associated with `state`.

Returns 0 if an error occurs.

DESCRIPTION

`cgGetNumStateEnumerants` returns the number of enumerants associated with a given `CGstate`. Enumerants can be added to a `CGstate` using `cgAddStateEnumerant`.

EXAMPLES

```c
int value;
char* pName;

int n Enums = cgGetNumStateEnumerants(state);
```
for (ii=0; ii<nEnums; ++ii) {
    pName = cgGetStateEnumerant(state, ii, &value);
    printf("%i: %s %i\n", ii+1, pName, value);
}

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgGetNumStateEnumerants was introduced in Cg 2.2.

SEE ALSO

cgAddStateEnumerant, cgGetStateEnumerant, cgGetStateEnumerantName, cgGetStateEnumerantValue

4.159 cgGetNumSupportedProfiles

NAME

cgGetNumSupportedProfiles - get the number of supported profiles

SYNOPSIS

#include <Cg/cg.h>

int cgGetNumSupportedProfiles( void );

PARAMETERS

None.

RETURN VALUES

Returns the number of profiles supported by this version of Cg.

DESCRIPTION

cgGetNumSupportedProfiles provides the number of profiles which are supported by this version of the Cg library. Note that a profile may be recognized by Cg but not supported by the platform on which the application is currently running. A graphics API specific routine such as cgGLIsProfileSupported must still be used to determine if the current GPU and driver combination supports a given profile.

EXAMPLES

CGprofile profile;
int nProfiles;
int ii;

nProfiles = cgGetNumSupportedProfiles();
printf("NumSupportedProfiles: %i\n", nProfiles);

for (ii=0; ii<nProfiles; ++ii) {
    profile = cgGetSupportedProfile(ii);
    printf("SupportedProfile %i: %s %i\n", ii, cgGetProfileString(profile),

cgGetProfile(cgGetProfileString(profile));
}

ERRORS
None.

HISTORY
cgGetNumSupportedProfiles was introduced in Cg 2.2.

SEE ALSO
cgGetSupportedProfile, cgIsProfileSupported, cgGetProfileProperty, cgGLIsProfileSupported, cgD3D9IsProfileSupported, cgD3D10IsProfileSupported, cgGetProfileString, cgGetProfile

4.160 cgGetNumUserTypes

NAME
cgGetNumUserTypes - get number of user-defined types in a program or effect

SYNOPSIS
#include <Cg/cg.h>

int cgGetNumUserTypes( CGhandle handle );

PARAMETERS
handle

The CGprogram or CGeffect in which the types are defined.

RETURN VALUES
Returns the number of user defined types.

DESCRIPTION
cgGetNumUserTypes returns the number of user-defined types in a given CGprogram or CGeffect.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if handle is not a valid program or effect handle.

HISTORY
cgGetNumUserTypes was introduced in Cg 1.2.

SEE ALSO
cgGetType, cgGetNamedUserType
4.161 cgGetParameterBaseResource

NAME

cgGetParameterBaseResource - get a parameter’s base resource

SYNOPSIS

#include <Cg/cg.h>

CGresource cgGetParameterBaseResource( CGparameter param );

PARAMETERS

param

The parameter.

RETURN VALUES

Returns the base resource of param.

Returns CG_UNDEFINED if no base resource exists for the given parameter.

DESCRIPTION

cgGetParameterBaseResource allows the application to retrieve the base resource for a parameter in a Cg program. The base resource is the first resource in a set of sequential resources. For example, if a given parameter has a resource of CG_ATTR7, it’s base resource would be CG_ATTR0. Only parameters with resources whose name ends with a number will have a base resource. For all other parameters the undefined resource CG_UNDEFINED will be returned.

The numerical portion of the resource may be retrieved with cgGetParameterResourceIndex. For example, if the resource for a given parameter is CG_ATTR7, cgGetParameterResourceIndex will return 7.

EXAMPLES

/* log info about parameter param for debugging */

printf("Resource: %s:%d (base %s)\n",
   cgGetResourceString(cgGetParameterResource(param)),
   cgGetParameterResourceIndex(param),
   cgGetResourceString(cgGetParameterBaseResource(param)));

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a leaf node.

HISTORY

cgGetParameterBaseResource was introduced in Cg 1.1.

SEE ALSO

cgGetParameterResource, cgGetParameterResourceIndex, cgGetResourceString
4.162 cgGetParameterBaseType

NAME
cgGetParameterBaseType - get a program parameter’s base type

SYNOPSIS
#include <Cg/cg.h>

CGtype cgGetParameterBaseType( CGparameter param );

PARAMETERS

param
The parameter.

RETURN VALUES

Returns the base type enumerant of param.
Returns CG_UNKNOWN_TYPE if an error occurs.

DESCRIPTION
cgGetParameterBaseType allows the application to retrieve the base type of a parameter.

If param is of a numeric type (scalar, vector, or matrix), the scalar enumerant corresponding to param’s type will be returned. For example, if param is of type CG_FLOAT4x3, cgGetParameterBaseType will return CG_FLOAT.

If param is an array, the base type of the array elements will be returned.

If param is a structure, its type-specific enumerant will be returned, as per cgGetParameterNamedType.
Otherwise, param’s type enumerant will be returned.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterBaseType was introduced in Cg 1.4.

SEE ALSO
cgGetParameterType, cgGetParameterNamedType, cgGetType, cgGetTypeString, cgGetParameterClass

4.163 cgGetParameterBufferIndex

NAME
cgGetParameterBufferIndex - get buffer index by parameter
SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterBufferIndex( CGparameter param );
```

PARAMETERS

- **param**
  The parameter for which the associated buffer index will be retrieved.

RETURN VALUES

Returns the index for the buffer to which `param` belongs.
Returns -1 if `param` does not belong to a buffer or an error occurs.

DESCRIPTION

cgGetParameterBufferIndex returns the index for the buffer to which a parameter belongs. If `param` does not belong to a buffer, then -1 is returned.

If the program to which `param` belongs is in an uncompiled state, it will be compiled by `cgGetParameterBufferIndex`. See `cgSetAutoCompile` for more information about program compile states.

EXAMPLES

```c
int index = cgGetParameterBufferIndex( myParam );
```

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

HISTORY

cgGetParameterBufferIndex was introduced in Cg 2.0.

SEE ALSO

cgSetProgramBuffer, cgGetParameterBufferOffset, cgGetParameterResourceSize, cgSetAutoCompile

### 4.164 cgGetParameterBufferOffset

NAME

cgGetParameterBufferOffset - get a parameter’s buffer offset

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterBufferOffset( CGparameter param );
```

PARAMETERS

- **param**
  The parameter for which the buffer offset will be retrieved.
RETURN VALUES

Returns the buffer offset for `param`. Returns -1 if `param` does not belong to a buffer or an error occurs.

DESCRIPTION
cgGetParameterBufferOffset returns the buffer offset associated with a parameter. If `param` does not belong to a buffer, then -1 is returned.

If the program to which `param` belongs is in an uncompiled state, it will be compiled by cgGetParameterBufferOffset. See cgSetAutoCompile for more information about program compile states.

EXAMPLES

```c
int offset = cgGetParameterBufferOffset( myParam );
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if `param` is not a valid parameter.

HISTORY
cgGetParameterBufferOffset was introduced in Cg 2.0.

SEE ALSO
cgSetProgramBuffer, cgGetParameterBufferIndex, cgGetParameterResourceSize, cgSetAutoCompile

4.165 cgGetParameterClass

NAME
cgGetParameterClass - get a parameter’s class

SYNOPSIS

```c
#include <Cg/cg.h>

CGparameterclass cgGetParameterClass( CGparameter param );
```

PARAMETERS
cgGetParameterClass( CGparameter param );

- `param`: The parameter.

RETURN VALUES

Returns the parameter class enumerant of `param`. Returns CG_PARAMETERCLASS_UNKNOWN if an error occurs.

DESCRIPTION
cgGetParameterClass allows the application to retrieve the class of a parameter. The returned CGparameterclass value enumerates the high-level parameter classes:
CG_PARAMETERCLASS_SCALAR
The parameter is of a scalar type, such as CG_INT, or CG_FLOAT.

CG_PARAMETERCLASS_VECTOR
The parameter is of a vector type, such as CG_INT1, or CG_FLOAT4.

CG_PARAMETERCLASS_MATRIX
The parameter is of a matrix type, such as CG_INT1x1, or CG_FLOAT4x4.

CG_PARAMETERCLASS_STRUCT
The parameter is a struct or interface.

CG_PARAMETERCLASS_ARRAY
The parameter is an array.

CG_PARAMETERCLASS_SAMPLER
The parameter is a sampler.

CG_PARAMETERCLASS_OBJECT
The parameter is a texture, string, or program.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterClass was introduced in Cg 1.4.

SEE ALSO

cgGetParameterClassString, cgGetParameterClassEnum, cgGetParameterType, cgGetType, cgGetTypeString

4.166  cgGetParameterClassEnum

NAME

cgGetParameterClassEnum - get the enumerant associated with a parameter class name

SYNOPSIS

#include <Cg/cg.h>

CGparameterclass cgGetParameterClassEnum( const char * pString );

PARAMETERS

pString

A string containing the case-sensitive parameter class name.
RETURN VALUES
Returns the parameter class enumerant associated with pString.
Returns CG_PARAMETERCLASS_UNKNOWN if the given parameter class does not exist.

DESCRIPTION
cgGetParameterClassEnum returns the enumerant associated with a parameter class name.

EXAMPLES
CGparameterclass structParameterClass = cgGetParameterClassEnum("struct");

if(cgGetParameterClassEnum(myparam) == structParameterClass)
{
    /* Do struct stuff */
}

ERRORS
CG_INVALID_PARAMETER_ERROR is generated if pString is NULL.

HISTORY
cgGetParameterClassEnum was introduced in Cg 2.2.

SEE ALSO
cgGetParameterClassString, cgGetParameterClass, cgGetTypeClass

4.167 cgGetParameterClassString

NAME
cgGetParameterClassString - get the name associated with a parameter class enumerant

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetParameterClassString( CGparameterclass parameterclass );

PARAMETERS
parameterclass
The parameter class enumerant.

RETURN VALUES
Returns the name associated with parameterclass.
Returns “unknown” if parameterclass is not a valid parameter class.

DESCRIPTION
cgGetParameterClassString returns the name associated with a parameter class enumerant.
EXAMPLES

```c
static void dumpCgParameterInfo(CGparameter parameter)
{
    /* ... */
    const char* p = cgGetParameterClassString(cgGetParameterType(parameter));
    if ( p ) {
        printf(" ParameterClass: %s\n", p);
    }
    /* ... */
}
```

ERRORS

None.

HISTORY

cgGetParameterClassString was introduced in Cg 2.2.

SEE ALSO

cgGetParameterClassEnum, cgGetParameterClass, cgGetTypeClass

4.168  cgGetParameterColumns

NAME

cgGetParameterColumns - get number of parameter columns

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterColumns( CGparameter param );
```

PARAMETERS

- `param`

  The parameter.

RETURN VALUES

Returns the number of columns associated with the type if `param` is a numeric type or an array of numeric types.

Returns 0 otherwise.

DESCRIPTION

cgGetParameterColumns return the number of columns associated with the given parameter’s type.

If `param` is an array, the number of columns associated with each element of the array is returned.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterColumns was introduced in Cg 1.4.

SEE ALSO

cgGetParameterType, cgGetParameterRows

4.169 cgGetParameterContext

NAME

cgGetParameterContext - get a parameter’s parent context

SYNOPSIS

#include <Cg/cg.h>

CGcontext cgGetParameterContext( CGparameter param );

PARAMETERS

param

The parameter.

RETURN VALUES

Returns a CGcontext handle to the parent context.

Returns NULL if an error occurs.

DESCRIPTION

cgGetParameterContext allows the application to retrieve a handle to the context to which a given parameter belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterContext was introduced in Cg 1.2.

SEE ALSO

cgCreateParameter, cgGetParameterProgram
4.170 cgGetParameterDefaultValue

NAME

cgGetParameterDefaultValue - get the default values of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

/* TYPE is int, float, or double */

int cgGetParameterDefaultValue{ifd}{rc}( CGparameter param,
    int nelements,
    TYPE * v );

PARAMETERS

param
    The program parameter whose default values will be retrieved.

nelements
    The number of elements in array v.

v
    Destination buffer to which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to v.

DESCRIPTION

The cgGetParameterDefaultValue functions allow the application to get the default values from any numeric parameter or parameter array. The default values are returned in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

There are versions of each function that return int, float or double values signified by i, f or d in the function name.

There are versions of each function that will cause any matrices referenced by param to be copied in either row-major or column-major order, as signified by the r or c in the function name.

For example, cgGetParameterDefaultValueic retrieves the default values of the given parameter using the supplied array of integer data, and copies matrix data in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of default values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

    int nrows = cgGetParameterRows(param);
    int ncols = cgGetParameterColumns(param);
    int asize = cgGetArrayTotalSize(param);
    int ntotal = nrows*ncols;
    if (asize > 0) ntotal *= asize;
EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

The cgGetParameterDefaultValue functions were introduced in Cg 2.1.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue, cgSetParameterValue,
cgGetParameterDefaultValuec, cgGetParameterDefaultValuedr, cgGetParameterDefaultValuefc, cgGetParameterDefaultValueld,
cgGetParameterDefaultValueldr, cgGetParameterDefaultValueic, cgGetParameterDefaultValuier

4.171 cgGetParameterDefaultValuedc

NAME

cgGetParameterDefaultValuedc - get the default values of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterDefaultValuedc( CGparameter param,
        int nelements,
        double * v );

PARAMETERS

param

The parameter whose default values will be retrieved.
nelements

The number of elements in array v.
v

Destination buffer into which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to v.
DESCRIPTION

cgGetParameterDefaultValuedc allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as doubles in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of default values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterDefaultValuedc was introduced in Cg 2.1.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.172 cgGetParameterDefaultValuedr

NAME

cgGetParameterDefaultValuedr - get the default values of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterDefaultValuedr( CGparameter param,
                                       int nelements,
                                       double * v );
```

4.172. cgGetParameterDefaultValuedr  299
PARAMETERS

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param</td>
<td>The parameter whose default values will be retrieved.</td>
</tr>
<tr>
<td>nelements</td>
<td>The number of elements in array ( v ).</td>
</tr>
<tr>
<td>v</td>
<td>Destination buffer into which the parameter default values will be written.</td>
</tr>
</tbody>
</table>

RETURN VALUES

Returns the total number of default values written to \( v \).

DESCRIPTION

cgGetParameterDefaultValuedr allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as doubles in \( v \).

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by \( \text{param} \) will be copied in row-major order.

The size of \( v \) is passed as \text{nelements}. If \( v \) is smaller than the total number of default values in the given source parameter, \text{CG_NOT_ENOUGH_DATA_ERROR} is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

\text{CG_INVALID_PARAM_HANDLE_ERROR} is generated if \( \text{param} \) is not a valid parameter.

\text{CG_INVALID_POINTER_ERROR} is generated if \( v \) is NULL.

\text{CG_NOT_ENOUGH_DATA_ERROR} is generated if \text{nelements} is less than the total size of \( \text{param} \).

\text{CG_NON_NUMERIC_PARAMETER_ERROR} is generated if \( \text{param} \) is of a non-numeric type.

HISTORY

cgGetParameterDefaultValuedr was introduced in Cg 2.1.

SEE ALSO

\text{cgGetParameterValue}, \text{cgSetParameterValue}, \text{cgGetParameterRows}, \text{cgGetParameterColumns}, \text{cgGetArrayTotalSize}
4.173 cgGetParameterDefaultValuefc

NAME

cgGetParameterDefaultValuefc - get the default values of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterDefaultValuefc( CGparameter param,
                                   int nelements,
                                   float * v );
```

PARAMETERS

- **param**
  The parameter whose default values will be retrieved.
- **nelements**
  The number of elements in array *v*.
- **v**
  Destination buffer into which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to *v*.

DESCRIPTION

cgGetParameterDefaultValuefc allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as floats in *v*.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by *param* will be copied in column-major order.

The size of *v* is passed as *nelements*. If *v* is smaller than the total number of default values in the given source parameter, **CG_NOT_ENOUGH_DATA_ERROR** is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

**CG_INVALID_PARAM_HANDLE_ERROR** is generated if *param* is not a valid parameter.

**CG_INVALID_POINTER_ERROR** is generated if *v* is NULL.
CG NOT ENOUGH DATA ERROR is generated if nelements is less than the total size of param.

CG NON NUMERIC PARAMETER ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterDefaultValuefr was introduced in Cg 2.1.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.174 cgGetParameterDefaultValuefr

NAME

cgGetParameterDefaultValuefr - get the default values of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterDefaultValuefr( CGparameter param, 
                                 int nelements, 
                                 float * v );

PARAMETERS

param
    The parameter whose default values will be retrieved.

nelements
    The number of elements in array v.

v
    Destination buffer into which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to v.

DESCRIPTION

cgGetParameterDefaultValuefr allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as floats in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in row-major order.

The size of v is passed as nelements. If v is smaller than the total number of default values in the given source parameter, CG NOT ENOUGH DATA ERROR is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterDefaultValueic was introduced in Cg 2.1.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.175 cgGetParameterDefaultValuеic

NAME

cgGetParameterDefaultValuеic - get the default values of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterDefaultValuеic( CGparameter param,
                                 int nelements,
                                 int * v);

PARAMETERS

param
   The parameter whose default values will be retrieved.
nelements
   The number of elements in array v.
v
   Destination buffer into which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to v.
DESCRIPTION

cgetGetParameterDefaultValueic allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as ints in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of default values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cGetParameterDefaultValueic was introduced in Cg 2.1.

SEE ALSO

cGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.176 cgGetParameterDefaultValueir

NAME

cgGetParameterDefaultValueir - get the default values of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterDefaultValueir( CGparameter param,
                                 int nelements,
                                 int * v );
```
PARAMETERS

param
The parameter whose default values will be retrieved.
nelements
The number of elements in array \( v \).
v
Destination buffer into which the parameter default values will be written.

RETURN VALUES

Returns the total number of default values written to \( v \).

DESCRIPTION

cgGetParameterDefaultValueir allows the application to get the default values from any numeric parameter or parameter array. The default values are returned as ints in \( v \).

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by \( \text{param} \) will be copied in row-major order.

The size of \( v \) is passed as \( \text{nelements} \). If \( v \) is smaller than the total number of default values in the given source parameter, \text{CG\_NOT\_ENOUGH\_DATA\_ERROR} is generated.

The total number of default values in a parameter, ntotal, may be computed as follow:

\[
\text{int } \text{nrows} = \text{cgGetParameterRows(param)}; \\
\text{int } \text{ncols} = \text{cgGetParameterColumns(param)}; \\
\text{int } \text{asize} = \text{cgGetArrayTotalSize(param)}; \\
\text{int } \text{ntotal} = \text{nrows}\times\text{ncols}; \\
\text{if } (\text{asize} > 0) \text{ntotal }*=\text{asize};
\]

EXAMPLES

\text{to-be-written}

ERRORS

\text{CG\_INVALID\_PARAM\_HANDLE\_ERROR} is generated if \( \text{param} \) is not a valid parameter.

\text{CG\_INVALID\_POINTER\_ERROR} is generated if \( v \) is NULL.

\text{CG\_NOT\_ENOUGH\_DATA\_ERROR} is generated if \( \text{nelements} \) is less than the total size of \( \text{param} \).

\text{CG\_NON\_NUMERIC\_PARAMETER\_ERROR} is generated if \( \text{param} \) is of a non-numeric type.

HISTORY

cgGetParameterDefaultValueir was introduced in Cg 2.1.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize
4.177 cgGetParameterDirection

NAME
cgGetParameterDirection - get a program parameter’s direction

SYNOPSIS

#include <Cg/cg.h>

cGenum cgGetParameterDirection( cGparameter param );

PARAMETERS

num
The program parameter.

RETURN VALUES

Returns the direction of param.

Returns CG_ERROR if an error occurs.

DESCRIPTION

cgGetParameterDirection allows the application to distinguish program input parameters from program output parameters. This information is necessary for the application to properly supply the program inputs and use the program outputs.

cgGetParameterDirection will return one of the following enumerants :

   CG_IN
   Specifies an input parameter.

   CG_OUT
   Specifies an output parameter.

   CG_INOUT
   Specifies a parameter that is both input and output.

   CG_ERROR
   If an error occurs.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterDirection was introduced in Cg 1.1.

SEE ALSO
cgGetNamedParameter, cgGetNextParameter, cgGetParameterName, cgGetParameterType, cgGetParameterVariability, cgSetParameterVariability
4.178 cgGetParameterEffect

NAME
cgGetParameterEffect - get a parameter’s parent program

SYNOPSIS
#include <Cg/cg.h>

CGeffect cgGetParameterEffect( CGparameter param );

PARAMETERS

param
The parameter.

RETURN VALUES

Returns a CGeffect handle to the parent effect.
Returns NULL if the parameter is not a child of an effect or if an error occurs.

DESCRIPTION

cgGetParameterEffect allows the application to retrieve a handle to the effect to which a given parameter belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterEffect was introduced in Cg 1.5.

SEE ALSO

cgCreateEffect, cgGetParameterProgram, cgCreateParameter

4.179 cgGetParameterIndex

NAME

cgGetParameterIndex - get an array member parameter’s index

SYNOPSIS
#include <Cg/cg.h>

int cgGetParameterIndex( CGparameter param );
PARAMETERS

param

The parameter.

RETURN VALUES

Returns the index associated with an array member parameter.
Returns -1 if the parameter is not in an array.

DESCRIPTION

cgGetParameterIndex returns the integer index of an array parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

HISTORY

cgGetParameterIndex was introduced in Cg 1.2.

SEE ALSO

cgGetArrayParameter

4.180 cgGetParameterName

NAME

cgGetParameterName - get a program parameter’s name

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetParameterName( CGparameter param );

PARAMETERS

param

The program parameter.

RETURN VALUES

Returns the null-terminated name string for the parameter.
Returns NULL if param is invalid.

DESCRIPTION

cgGetParameterName allows the application to retrieve the name of a parameter in a Cg program. This name can be used later to retrieve the parameter from the program using cgGetNamedParameter.
EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterName was introduced in Cg 1.1.

SEE ALSO
cgGetNamedParameter, cgGetNextParameter, cgGetParameterType, cgGetParameterVariability, cgGetParameterDirection, cgSetParameterVariability

4.181 cgGetParameterNamedType

NAME
cgGetParameterNamedType - get a program parameter's type

SYNOPSIS
#include <Cg/cg.h>

CGtype cgGetParameterNamedType( CGparameter param );

PARAMETERS
param
The parameter.

RETURN VALUES
Returns the type of param.

DESCRIPTION
cgGetParameterNamedType returns the type of param similarly to cgGetParameterType. However, if the type is a user defined struct it will return the unique enumerant associated with the user defined type instead of CG_STRUCT.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterNamedType was introduced in Cg 1.2.

SEE ALSO
cgGetParameterType, cgGetParameterBaseType
4.182 cgGetParameterOrdinalNumber

NAME

cgGetParameterOrdinalNumber - get a program parameter’s ordinal number

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterOrdinalNumber( CGparameter param );

PARAMETERS

    param
    The program parameter.

RETURN VALUES

Returns the ordinal number associated with a parameter. If the parameter is a constant (cgGetParameterVariability
returns CG_CONSTANT) then 0 is returned and no error is generated.

When cgGetParameterOrdinalNumber is passed an array, the ordinal number of the first array element is returned. When passed a struct, the ordinal number of first struct data member is returned.

DESCRIPTION

cgGetParameterOrdinalNumber returns an integer that represents the order in which the parameter was declared
within the Cg program.

Ordinal numbering begins at zero, starting with a program’s first local leaf parameter. The subsequent local leaf
parameters are enumerated in turn, followed by the program’s global leaf parameters.

EXAMPLES

The following Cg program:

struct MyStruct { float a; sampler2D b; }
float globalvar1;
float globalvar2
float4 main(float2 position : POSITION,
    float4 color : COLOR,
    uniform MyStruct mystruct,
    float2 texCoord : TEXCOORD0) : COLOR
{
    /* etc ... */
}

Would result in the following parameter ordinal numbering:

position -> 0
color -> 1
mystruct.a -> 2
mystruct.b -> 3
texCoord -> 4
globalvar1 -> 5
globalvar2 -> 6
ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterOrdinalNumber was introduced in Cg 1.1.

SEE ALSO
cgGetParameterVariability

4.183 cgGetParameterProgram

NAME
cgGetParameterProgram - get a parameter’s parent program

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgGetParameterProgram( CGparameter param );

PARAMETERS

param
The parameter.

RETURN VALUES

Returns a CGprogram handle to the parent program.

Returns NULL if the parameter is not a child of a program or an error occurs.

DESCRIPTION

cgGetParameterProgram allows the application to retrieve a handle to the program to which a given parameter belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterProgram was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram, cgGetParameterEffect
4.184 cgGetParameterResource

NAME

cgGetParameterResource - get a program parameter’s resource

SYNOPSIS

#include <Cg/cg.h>

CGresource cgGetParameterResource( CGparameter param );

PARAMETERS

  param

  The program parameter.

RETURN VALUES

Returns the resource of param.

DESCRIPTION

cgGetParameterResource allows the application to retrieve the resource for a parameter in a Cg program. This resource is necessary for the application to be able to supply the program’s inputs and use the program’s outputs.

The resource enumerant is a profile-specific hardware resource.

EXAMPLES

/* log info about parameter param for debugging */

printf("Resource: %s:%d (base %s)\n",
  cgGetResourceString(cgGetParameterResource(param)),
  cgGetParameterResourceIndex(param),
  cgGetResourceString(cgGetParameterBaseResource(param)));

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a leaf node.

HISTORY

cgGetParameterResource was introduced in Cg 1.1.

SEE ALSO

cgGetParameterResourceIndex, cgGetParameterBaseResource, cgGetResourceString

4.185 cgGetParameterResourceIndex

NAME

cgGetParameterResourceIndex - get a program parameter’s resource index
SYNOPSIS

#include <Cg/cg.h>

unsigned long cgGetParameterResourceIndex( CGparameter param );

PARAMETERS

param

The program parameter.

RETURN VALUES

Returns the resource index of \texttt{param}.

Returns 0 if an error occurs.

DESCRIPTION

cgGetParameterResourceIndex allows the application to retrieve the resource index for a parameter in a Cg program. This index value is only used with resources that are linearly addressable.

EXAMPLES

/* log info about parameter \texttt{param} for debugging */

printf("Resource: %s:%d (base %s)\n",
   cgGetResourceString(cgGetParameterResource(param)),
   cgGetParameterResourceIndex(param),
   cgGetResourceString(cgGetParameterBaseResource(param)));

ERRORS

\texttt{CG_INVALID_PARAM_HANDLE_ERROR} is generated if \texttt{param} is not a valid parameter.

\texttt{CG_INVALID_PARAMETER_ERROR} is generated if \texttt{param} is not a leaf node.

HISTORY

cgGetParameterResourceIndex was introduced in Cg 1.1.

SEE ALSO


4.186 cgGetParameterResourceName

NAME

\texttt{cgGetParameterResourceName} - get a program parameter’s resource name

SYNOPSIS

#include <Cg/cg.h>

\texttt{const char * cgGetParameterResourceName( CGparameter param );}
PARAMETERS

    param
    The program parameter.

RETURN VALUES

Returns the resource name of param.
Returns NULL if an error occurs.

DESCRIPTION

cgGetParameterResourceName allows the application to retrieve the resource name for a parameter in a Cg program. For translated profiles this name will most likely be different from the string returned by cgGetResourceString.

EXAMPLES

/* log info about parameter param for debugging */

printf("Resource: %s:%d (base %s) [name:%s]\n",
    cgGetResourceString(cgGetParameterResource(param)),
    cgGetParameterResourceIndex(param),
    cgGetResourceString(cgGetParameterBaseResource(param)),
    cgGetParameterResourceName(param));

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is not a leaf node.

HISTORY

cgGetParameterResourceName was introduced in Cg 2.1.

SEE ALSO


4.187 cgGetParameterResourceSize

NAME

cgGetParameterResourceSize - get size of resource associated with a parameter

SYNOPSIS

#include <Cg/cg.h>

long cgGetParameterResourceSize( CGparameter param );

PARAMETERS

    param
    The parameter for which the associated resource size will be retrieved.
RETURN VALUES

Returns the size on the GPU of the resource associated with `param`.
Returns -1 if an error occurs.

DESCRIPTION

`cgGetParameterResourceSize` returns the size in bytes of the resource corresponding to a parameter if the parameter belongs to a Cg buffer resource.

The size of sampler parameters is zero because they have no actual data storage.

The size of an array parameter is the size of an array element parameter times the length of the array.

The size of a structure parameter is the sum of the size of all the members of the structure plus zero or more bytes of profile-dependent padding.

EXAMPLES

_to-be-written_

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

HISTORY

`cgGetParameterResourceSize` was introduced in Cg 2.0.

SEE ALSO

`cgGetParameterBufferOffset`, `cgGetParameterBufferIndex`

4.188 `cgGetParameterResourceType`

NAME

`cgGetParameterResourceType` - get a parameter’s resource type

SYNOPSIS

```c
#include <Cg/cg.h>

CGtype cgGetParameterResourceType( CGparameter param );
```

PARAMETERS

- `param`
  The parameter for which the resource type will be returned.

RETURN VALUES

Returns the resource type of `param`.

Returns `CG_UNKNOWN_TYPE` if the parameter does not belong to a program, if the program is not compiled, or if an error occurs.

DESCRIPTION

`cgGetParameterResourceType` allows the application to retrieve the resource type for a parameter in a Cg program.
EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterResourceType was introduced in Cg 2.0.

SEE ALSO


4.189 cgGetParameterRows

NAME

cgGetParameterRows - get number of parameter rows

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterRows( CGparameter param );

PARAMETERS

param
The parameter.

RETURN VALUES

Returns the number of rows associated with the type if param is a numeric type or an array of numeric types.

Returns 0 otherwise.

DESCRIPTION

cgGetParameterRows return the number of rows associated with the given parameter’s type.

If param is an array, the number of rows associated with each element of the array is returned.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterRows was introduced in Cg 1.4.
SEE ALSO
cgGetParameterType, cgGetParameterColumns

4.190 cgGetParameterSemantic

NAME
cgGetParameterSemantic - get a parameter’s semantic

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetParameterSemantic( CGparameter param );

PARAMETERS
param
The parameter.

RETURN VALUES
Returns the null-terminated semantic string for the parameter.
Returns NULL if an error occurs.

DESCRIPTION
cgGetParameterSemantic allows the application to retrieve the semantic of a parameter in a Cg program. If a uniform
parameter does not have a user-assigned semantic, an empty string will be returned. If a varying parameter does not
have a user-assigned semantic, the semantic string corresponding to the compiler-assigned resource for that varying
will be returned.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGetParameterSemantic was introduced in Cg 1.1.

SEE ALSO
cgSetParameterSemantic, cgGetSemanticCasePolicy, cgSetSemanticCasePolicy, cgGetParameterResource, cgGetPa-
terResourceIndex, cgGetParameterName, cgGetParameterType

4.191 cgGetParameterSettingMode

NAME
cgGetParameterSettingMode - get the parameter setting mode for a context
SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetParameterSettingMode( CGcontext context );

PARAMETERS

context

The context from which the parameter setting mode will be retrieved.

RETURN VALUES

Returns the parameter setting mode enumerant for context.

Returns CG_UNKNOWN if an error occurs.

DESCRIPTION

cgGetParameterSettingMode returns the current parameter setting mode enumerant for context. See cgSetParameterSettingMode for more information.

EXAMPLES

/* assumes cgGetProgramContext(program) == context */

if (cgGetParameterSettingMode(context) == CG_DEFERRED_PARAMETER_SETTING) {
    cgUpdateProgramParameters(program);
}

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGetParameterSettingMode was introduced in Cg 2.0.

SEE ALSO

cgSetParameterSettingMode, cgUpdatePassParameters, cgUpdateProgramParameters

4.192 cgGetParameterType

NAME

cgGetParameterType - get a program parameter’s type

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetParameterType( CGparameter param );

PARAMETERS

param

The parameter.
RETURN VALUES

Returns the type enumerant of `param`.

Returns `CG_UNKNOWN_TYPE` if an error occurs.

DESCRIPTION

cgGetParameterType allows the application to retrieve the type of a parameter in a Cg program. This type is necessary for the application to be able to supply the program’s inputs and use the program’s outputs.

cgGetParameterType will return `CG_ARRAY` if the parameter is an array, `CG_STRUCT` if the parameter is a struct, or `CG_UNIFORM_BUFFER` if the parameter is a uniform buffer. Otherwise it will return the data type associated with the parameter.

EXAMPLES
to-be-written

ERRORS

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.

HISTORY

cgGetParameterType was introduced in Cg 1.1.

SEE ALSO

cgGetType, cgGetParameterBaseType, cgGetTypeString, cgGetParameterClass

4.193 cgGetParameterValue

NAME

cgGetParameterValue - get the value of any numeric parameter

SYNOPSIS

```
#include <Cg/cg.h>

/* TYPE is int, float, or double */

int cgGetParameterValue(ifd){rc}{ CGparameter param, 
   int nelements, 
   TYPE * v );
```

PARAMETERS

- `param`
  The program parameter whose value will be retrieved.
- `nelements`
  The number of elements in array `v`.
- `v`
  Destination buffer to which the parameter values will be written.
RETURN VALUES

Returns the total number of values written to \( v \).

DESCRIPTION

The \texttt{cgGetParameterValue} functions allow the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned in \( v \).

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

There are versions of each function that return \texttt{int}, \texttt{float} or \texttt{double} values signified by \texttt{i}, \texttt{f} or \texttt{d} in the function name.

There are versions of each function that will cause any matrices referenced by \texttt{param} to be copied in either row-major or column-major order, as signified by the \texttt{r} or \texttt{c} in the function name.

For example, \texttt{cgGetParameterValueic} retrieves the values of the given parameter using the supplied array of integer data, and copies matrix data in column-major order.

The size of \( v \) is passed as \texttt{nelements}. If \( v \) is smaller than the total number of values in the given source parameter, \texttt{CG_NOT_ENOUGH_DATA_ERROR} is generated.

The total number of values in a parameter, \( ntotal \), may be computed as follow:

\[
\begin{align*}
\texttt{int } & \quad \texttt{nrows } = \texttt{cgGetParameterRows(} \texttt{param} \texttt{)}; \\
\texttt{int } & \quad \texttt{ ncols } = \texttt{cgGetParameterColumns(} \texttt{param} \texttt{)}; \\
\texttt{int } & \quad \texttt{asize } = \texttt{cgGetArrayTotalSize(} \texttt{param} \texttt{)}; \\
\texttt{int } & \quad \texttt{ntotal } = \texttt{nrows} \times \texttt{ncols}; \\
\text{if } & \quad \texttt{(asize > 0)} \texttt{ ntota l } *= \texttt{asize};
\end{align*}
\]

Note: Previous releases of Cg allowed you to store more values in a parameter than indicated by the parameter’s type. For example, one could use \texttt{cgGLSetParameter4f} to store four values into a parameter of type \texttt{CG_FLOAT} (not \texttt{CG_FLOAT4}). All four values could later be retrieved using a get call which requested more than one value. However, this feature conflicts with the GLSL approach and also leads to issues with parameters mapped into BUFFERS. Therefore, beginning with Cg 2.0 any components beyond the number indicated by the parameter type are ignored.

EXAMPLES

to-be-written

ERRORS

\texttt{CG_INVALID_PARAM_HANDLE_ERROR} is generated if \texttt{param} is not a valid parameter.

\texttt{CG_INVALID_POINTER_ERROR} is generated if \( v \) is NULL.

\texttt{CG_NOT_ENOUGH_DATA_ERROR} is generated if \texttt{nelements} is less than the total size of \texttt{param}.

\texttt{CG_NON_NUMERIC_PARAMETER_ERROR} is generated if \texttt{param} is of a non-numeric type.

HISTORY

The \texttt{cgGetParameterValue} functions were introduced in Cg 1.4.

SEE ALSO

4.194 cgGetParameterValuedc

NAME

cgGetParameterValuedc - get the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterValuedc( CGparameter param,
                           int nelements,
                           double * v );

PARAMETERS

  param
  The parameter whose value will be retrieved.

  nelements
  The number of elements in array v.

  v
  Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.

DESCRIPTION

cgGetParameterValuedc allows the application to get the value(s) from any numeric parameter or parameter array.
The value(s) are returned as doubles in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of values in the given source parameter, 
CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

  int nrows = cgGetParameterRows(param);
  int ncols = cgGetParameterColumns(param);
  int asize = cgGetArrayTotalSize(param);
  int ntotal = nrows*ncols;
  if (asize > 0) ntotal *= asize;

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterValuedc was introduced in Cg 1.4.

SEE ALSO

cgGetParameterDefaultValue, cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.195 cgGetParameterValuedr

NAME

cgGetParameterValuedr - get the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterValuedr( CGparameter param,
                           int nelements,
                           double * v );

PARAMETERS

param
The parameter whose value will be retrieved.
nelements
The number of elements in array v.
v
Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.

DESCRIPTION

cgGetParameterValuedr allows the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned as doubles in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in row-major order.

The size of v is passed as nelements. If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterValue was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgGetParameterDefaultValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.196 cgGetParameterValuefc

NAME

cgGetParameterValuefc - get the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterValuefc( CGparameter param,
                         int nelements,
                         float * v );

PARAMETERS

param
The parameter whose value will be retrieved.
nelements
The number of elements in array v.
v
Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.
DESCRIPTION

cgGetParameterValuefc allows the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned as floats in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterValuefc was introduced in Cg 1.4.

SEE ALSO

cgGetParameterDefaultValuefc, cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.197 cgGetParameterValuefr

NAME

cgGetParameterValuefr - get the value of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetParameterValuefr( CGparameter param,
                          int nelements,
                          float * v );
```
PARAMETERS

param
The parameter whose value will be retrieved.

elements
The number of elements in array v.

v
Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.

DESCRIPTION

cgGetParameterValuefr allows the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned as floats in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in row-major order.

The size of v is passed as elements. If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if elements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterValuefr was introduced in Cg 1.4.

SEE ALSO

.cgGetParameterDefaultValuefr, cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize
4.198 cgGetParameterValueic

NAME

cgGetParameterValueic - get the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterValueic( CGparameter param,
                          int nelements,
                          int * v );

PARAMETERS

  param
  The parameter whose value will be retrieved.

  nelements
  The number of elements in array v.

  v
  Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.

DESCRIPTION

cgGetParameterValueic allows the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned as ints in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in column-major order.

The size of v is passed as nelements. If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

  int nrows = cgGetParameterRows(param);
  int ncols = cgGetParameterColumns(param);
  int asize = cgGetArrayTotalSize(param);
  int ntotal = nrows*ncols;
  if (asize > 0) ntotal *= asize;

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgGetParameterValueic was introduced in Cg 1.4.

SEE ALSO

cgGetParameterDefaultValueic, cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.199 cgGetParameterValueir

NAME

cgGetParameterValueir - get the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

int cgGetParameterValueir( CGparameter param,
    int nelements,
    int * v );

PARAMETERS

param
  The parameter whose value will be retrieved.
nelements
  The number of elements in array v.
v
  Destination buffer into which the parameter values will be written.

RETURN VALUES

Returns the total number of values written to v.

DESCRIPTION

cgGetParameterValueir allows the application to get the value(s) from any numeric parameter or parameter array. The value(s) are returned as ints in v.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param will be copied in row-major order.

The size of v is passed as nelements. If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY
cgGetParameterValue was introduced in Cg 1.4.

SEE ALSO
cgGetParameterDefaultValue, cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

4.200 cgGetParameterValues

NAME

cgGetParameterValues - deprecated

DESCRIPTION

cgGetParameterValues is deprecated. Use a variation of cgGetParameterValue or cgGetParameterDefaultValue instead.

SEE ALSO

cgGetParameterValue, cgGetParameterDefaultValue

4.201 cgGetParameterVariability

NAME

cgGetParameterVariability - get a parameter’s variability

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetParameterVariability( CGparameter param );
PARAMETERS

param

The program parameter.

RETURN VALUES

Returns the variability of param.

Returns CG_ERROR if an error occurs.

DESCRIPTION

cgGetParameterVariability allows the application to retrieve the variability of a parameter in a Cg program. This variability is necessary for the application to be able to supply the program’s inputs and use the program’s outputs.

cgGetParameterVariability will return one of the following variabilities:

CG_VARYING

A varying parameter is one whose value changes with each invocation of the program.

CG_UNIFORM

A uniform parameter is one whose value does not change with each invocation of a program, but whose value can change between groups of program invocations.

CG_LITERAL

A literal parameter is folded out at compile time. Making a uniform parameter literal with cgSetParameterVariability will often make a program more efficient at the expense of requiring a compile every time the value is set.

CG_CONSTANT

A constant parameter is never changed by the user. It’s generated by the compiler by certain profiles that require immediate values to be placed in certain resource locations.

CG_MIXED

A structure parameter that contains parameters that differ in variability.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGetParameterVariability was introduced in Cg 1.1.

SEE ALSO

cgGetNamedParameter, cgGetNextParameter, cgGetParameterName, cgGetParameterType, cgGetParameterDirection, cgSetParameterVariability
4.202 cgGetParentType

NAME
cgGetParentType - gets a parent type of a child type

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetParentType( CGtype type,
                        int index );

PARAMETERS

type
The child type.

index
The index of the parent type. index must be greater than or equal to 0 and less than the value returned by cgGetNumParentTypes.

RETURN VALUES

Returns the number of parent types.

Returns NULL if there are no parents.

Returns CG_UNKNOWN_TYPE if type is a built-in type or an error is thrown.

DESCRIPTION
cgGetParentType returns a parent type of type.

A parent type is one from which the given type inherits, or an interface type that the given type implements. For example, given the type definitions:

interface myiface {
    float4 eval(void);
};

struct mystruct : myiface {
    float4 value;
    float4 eval(void) { return value; }
};

mystruct has a single parent type, myiface.

Note that the current Cg language specification implies that a type may only have a single parent type – an interface implemented by the given type.

EXAMPLES

to-be-written

ERRORS

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if index is outside the proper range.
HISTORY

cgGetParentType was introduced in Cg 1.2.

SEE ALSO

cgGetNumParentTypes

4.203 cgGetPassName

NAME

cgGetPassName - get a technique pass’s name

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetPassName( CGpass pass );

PARAMETERS

pass

The pass.

RETURN VALUES

Returns the null-terminated name string for the pass.

Returns NULL if pass is invalid.

DESCRIPTION

cgGetPassName allows the application to retrieve the name of a pass in a Cg program. This name can be used later to retrieve the pass from the program using cgGetNamedPass.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY

cgGetPassName was introduced in Cg 1.4.

SEE ALSO

cgGetNamedPass, cgGetFirstPass, cgGetNextPass

4.204 cgGetPassProgram

NAME

cgGetPassProgram - get domain program from a pass
SYNOPSIS

#include <Cg/cg.h>

CGprogram cgGetPassProgram( CGpass pass, 
    CGdomain domain );

PARAMETERS

pass
    The pass from which to get a program.

domain
    The domain for which a program will be retrieved.

RETURN VALUES

Returns the program associated with a specified domain from the given pass.
Returns NULL if pass or domain is invalid.

DESCRIPTION

cgGetPassProgram allows the application to retrieve the program associated with a specific domain from a pass.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

CG_INVALID_ENUMERANT_ERROR is generated if domain is not CG_VERTEX_DOMAIN, 
    CG_FRAGMENT_DOMAIN, or CG_GEOMETRY_DOMAIN.

HISTORY

cgGetPassProgram was introduced in Cg 2.1.

SEE ALSO

cgGetFirstPass, cgGetNextPass

4.205 cgGetPassTechnique

NAME

cgGetPassTechnique - get a pass’s technique

SYNOPSIS

#include <Cg/cg.h>

CGtechnique cgGetPassTechnique( CGpass pass );
PARAMETERS

pass

The pass.

RETURN VALUES

Returns a CGtechnique handle to the technique.

Returns NULL if an error occurs.

DESCRIPTION
cgGetPassTechnique allows the application to retrieve a handle to the technique to which a given pass belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY
cgGetPassTechnique was introduced in Cg 1.4.

SEE ALSO
cgIsTechnique, cgGetNextTechnique, cgIsPass

4.206  cgGetProfile

NAME
cgGetProfile - get the profile enumerant from a profile name

SYNOPSIS

#include <Cg/cg.h>

CGprofile cgGetProfile( const char * profile_string );

PARAMETERS

profile_string

A string containing the case-sensitive profile name.

RETURN VALUES

Returns the profile enumerant of profile_string.

Returns CG_PROFILE_UNKNOWN if the given profile does not exist.

DESCRIPTION
cgGetProfile returns the enumerant assigned to a profile name.
EXAMPLES

CGprofile ARBVP1Profile = cgGetProfile("arbvp1");

if(cgGetProgramProfile(myprog) == ARBVP1Profile)
{
    /* Do stuff */
}

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if profile_string is NULL.

HISTORY

cgGetProfile was introduced in Cg 1.1.

SEE ALSO

cgGetProfileString, cgGetProgramProfile

4.207 cgGetProfileDomain

NAME

cgGetProfileDomain - get the domain of a profile enumerant

SYNOPSIS

#include <Cg/cg.h>

CGdomain cgGetProfileDomain( CGprofile profile );

PARAMETERS

  profile

A profile enumerant.

RETURN VALUES

Returns the domain enumerant associated with profile.
Returns CG_UNKNOWN_DOMAIN if profile is not a valid profile.

DESCRIPTION

cgGetProfileDomain returns the domain to which a given profile belongs. One of the following enumerants will be returned:

  * CG_VERTEX_DOMAIN
  * CG_FRAGMENT_DOMAIN
  * CG_GEOMETRY_DOMAIN
  * CG_TESSELLATION_CONTROL_DOMAIN
  * CG_TESSELLATION_EVALUATION_DOMAIN
  * CG_UNKNOWN_DOMAIN
EXAMPLES

CGdomain domain = cgGetProfileDomain(CG_PROFILE_PS_3_0);
/* domain == CG_FRAGMENT_DOMAIN */

ERRORS

None.

HISTORY

cgGetProfileDomain was introduced in Cg 1.5.
CG_GEOMETRY_DOMAIN was introduced in Cg 2.0.
CG_TESSELLATION_CONTROL_DOMAIN and CG_TESSELLATION_EVALUATION_DOMAIN were introduced in Cg 3.0.

SEE ALSO

cgGetNumProgramDomains, cgGetProgramDomainProfile

4.208 cgGetProfileProperty

NAME

cgGetProfileProperty - query property of a profile

SYNOPSIS

#include <Cg/cg.h>

CGbool cgGetProfileProperty( CGprofile profile, CGenum query );

PARAMETERS

profile

The profile to query.

query

An enumerator describing the property to be queried. The following enumerants are allowed:

* CG_IS_OPENGL_PROFILE
* CG_IS_DIRECT3D_PROFILE
* CG_IS_DIRECT3D_8_PROFILE
* CG_IS_DIRECT3D_9_PROFILE
* CG_IS_DIRECT3D_10_PROFILE
* CG_IS_DIRECT3D_11_PROFILE
* CG_IS_VERTEX_PROFILE
* CG_IS_FRAGMENT_PROFILE
* CG_IS_GEOMETRY_PROFILE
* CG_IS_TESSELLATION_CONTROL_PROFILE
* CG_IS_TESSELLATION_EVALUATION_PROFILE
* CG_IS_TRANSLATION_PROFILE
* CG_IS_HLSL_PROFILE
* CG_IS_GLSL_PROFILE

**RETURN VALUES**

Returns **CG_TRUE** if `profile` holds the property expressed by `query`. Returns **CG_FALSE** otherwise.

**DESCRIPTION**

cgGetProfileProperty returns property information about the given profile.

`query` must be one of the following enumerants:

* **CG_IS_OPENGL_PROFILE**
  profile is an OpenGL profile.
* **CG_IS_DIRECT3D_PROFILE**
  profile is Direct3D profile.
* **CG_IS_DIRECT3D_8_PROFILE**
  profile is Direct3D8 profile.
* **CG_IS_DIRECT3D_9_PROFILE**
  profile is Direct3D9 profile.
* **CG_IS_DIRECT3D_10_PROFILE**
  profile is Direct3D10 profile.
* **CG_IS_DIRECT3D_11_PROFILE**
  profile is Direct3D11 profile.
* **CG_IS_VERTEX_PROFILE**
  profile is vertex profile.
* **CG_IS_FRAGMENT_PROFILE**
  profile is fragment profile.
* **CG_IS_GEOMETRY_PROFILE**
  profile is geometry profile.
* **CG_IS_TESSELLATION_CONTROL_PROFILE**
  profile is tessellation control profile.
* **CG_IS_TESSELLATION_EVALUATION_PROFILE**
  profile is tessellation evaluation profile.
* **CG_IS_TRANSLATION_PROFILE**
  profile is a translation profile.
* **CG_IS_HLSL_PROFILE**
  profile is an HLSL translation profile.
*CG_IS_GLSL_PROFILE*

profile is a GLSL translation profile.

EXAMPLES

CGprofile profile;
int nProfiles;
int ii;

nProfiles = cgGetNumSupportedProfiles();
printf("NumSupportedProfiles: %i\n", nProfiles);

for (ii=0; ii<nProfiles; ++ii) {
    profile = cgGetSupportedProfile(ii);
    if (cgGetProfileProperty(profile, CG_IS_OPENGL_PROFILE)) {
        printf("%s is an OpenGL profile\n", cgGetProfileString(profile));
    } else {
        printf("%s is not an OpenGL profile\n", cgGetProfileString(profile));
    }
}

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if profile is not supported by this version of the Cg library.

CG_INVALID_ENUMERANT_ERROR is generated if query is not CG_IS_OPENGL_PROFILE, CG_IS_DIRECT3D_PROFILE, CG_IS_DIRECT3D_8_PROFILE, CG_IS_DIRECT3D_9_PROFILE, CG_IS_DIRECT3D_10_PROFILE, CG_IS_DIRECT3D_11_PROFILE, CG_IS_VERTEX_PROFILE, CG_IS_FRAGMENT_PROFILE, CG_IS_GEOMETRY_PROFILE, CG_IS_TESSELLATION_CONTROL_PROFILE, CG_IS_TESSELLATION_EVALUATION_PROFILE, CG_IS_TRANSLATION_PROFILE, CG_IS_HLSL_PROFILE, or CG_IS_GLSL_PROFILE

HISTORY

cgGetProfileProperty was introduced in Cg 2.2.

SEE ALSO

cgGetNumSupportedProfiles, cgGetSupportedProfile, cgIsProfileSupported, cgGetProfileString, cgGetProfile

4.209 cgGetProfileSibling

NAME

cgGetProfileSibling - get the corresponding sibling profile in another domain

SYNOPSIS

#include <Cg/cg.h>

CGprofile cgGetProfileSibling( CGprofile profile, CGdomain domain );

PARAMETERS

profile
The profile enumerant.
profile
The domain enumerant.

RETURN VALUES
Returns the profile corresponding to profile in the specified domain.
Returns CG_PROFILE_UNKNOWN if profile is unrecognized or does not have a valid sibling profile for domain.

DESCRIPTION
cgGetProfileSibling returns the corresponding sibling profile in another domain

EXAMPLES
CGprofile fprofile = cgGetProfileSibling(CG_PROFILE_GP5VP,CG_FRAGMENT_DOMAIN);
/* fprofile == CG_PROFILE_GP5FP */

ERRORS
None.

HISTORY
cgGetProfileSibling was introduced in Cg 3.1.

SEE ALSO
cgGetProfileDomain, cgGetProfileProperty

4.210 cgGetProfileString

NAME
cgGetProfileString - get the profile name associated with a profile enumerant

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetProfileString( CGprofile profile );

PARAMETERS

profile
The profile enumerant.

RETURN VALUES
Returns the profile string of the enumerant profile.
Returns NULL if profile is not a valid profile.

DESCRIPTION
cgGetProfileString returns the profile name associated with a profile enumerant.
EXAMPLES

```c
static void dumpCgProgramInfo(CGprogram program)
{
    const char *p = cgGetProfileString(cgGetProgramProfile(program));
    if ( p ) {
        printf(" Profile: %s\n", p);
    }
    /* ... */
}
```

ERRORS
None.

HISTORY

cgGetProfileString was introduced in Cg 1.1.

SEE ALSO

cgGetProfile, cgGetProgramProfile

4.211 cgGetProgramBuffer

NAME

cgGetProgramBuffer - get buffer associated with a buffer index

SYNOPSIS

```c
#include <Cg/cg.h>

CGbuffer cgGetProgramBuffer( CGprogram program,
                          int bufferIndex );
```

PARAMETERS

- program
  The program from which the associated buffer will be retrieved.
- bufferIndex
  The buffer index for which the associated buffer will be retrieved.

RETURN VALUES

Returns a buffer handle on success.

Returns NULL if an error occurs.

DESCRIPTION

`cgGetProgramBuffer` returns the buffer handle associated with a given buffer index from `program`. The returned value can be NULL if no buffer is associated with this index or if an error occurs.

EXAMPLES

```c
CGbuffer myBuffer = cgGetProgramBuffer( myProgram, 0 );
```
ERRORS

**CG_INVALID_PROGRAM_HANDLE_ERROR** is generated if **program** is not a valid program handle.

**CG_BUFFER_INDEX_OUT_OF_RANGE_ERROR** is generated if **bufferIndex** is not within the valid range of buffer indices for **program**.

HISTORY

cgGetProgramBuffer was introduced in Cg 2.0.

SEE ALSO

cgSetProgramBuffer, cgGetParameterBufferIndex, cgCreateBuffer

### 4.212 cgGetProgramBufferMaxIndex

**NAME**

cgGetProgramBufferMaxIndex - get the maximum index of a buffer for a given profile

**SYNOPSIS**

```c
#include <Cg/cg.h>

int cgGetProgramBufferMaxIndex( CGprofile profile );
```

**PARAMETERS**

- **profile**
  
  The target for determining the maximum buffer index.

**RETURN VALUES**

- Returns the maximum buffer index for a given profile.
- Returns 0 if an error occurs.

**DESCRIPTION**

cgGetProgramBufferMaxIndex returns the maximum buffer index for a **profile**. cgGetProgramBufferMaxIndex will return 0 if an invalid profile is passed.

**EXAMPLES**

```c
int size = cgGetProgramBufferMaxIndex( CG_PROFILE_GPU_VP );
```

**ERRORS**

- none.

**HISTORY**

cgGetProgramBufferMaxIndex was introduced in Cg 2.0.

**SEE ALSO**

cgGetProgramBufferMaxSize, cgGetParameterBufferIndex, cgCreateBuffer
4.213 cgGetProgramBufferMaxSize

NAME

.cgGetProgramBufferMaxSize - get the maximum size of a buffer in bytes for a given profile

SYNOPSIS

#include <Cg/cg.h>

int cgGetProgramBufferMaxSize( CGprofile profile );

PARAMETERS

profile

The target for determining the maximum buffer size.

RETURN VALUES

Returns the size of a buffer for the given profile in bytes.

Returns 0 if an error occurs.

DESCRIPTION

cgGetProgramBufferMaxSize returns the maximum size of a buffer for a profile in bytes. cgGetProgramBufferMaxSize will return 0 if an invalid profile is passed.

EXAMPLES

int size = cgGetProgramBufferMaxSize( CG_PROFILE_GPU_VP );

ERRORS

none.

HISTORY

cgGetProgramBufferMaxSize was introduced in Cg 2.0.

SEE ALSO

.cgGetProgramBufferMaxIndex, cgGetParameterBufferIndex, cgCreateBuffer

4.214 cgGetProgramContext

NAME

.cgGetProgramContext - get a programs parent context

SYNOPSIS

#include <Cg/cg.h>

CGcontext cgGetProgramContext( CGprogram program );
PARAMETERS

program

The program.

RETURN VALUES

Returns a CGcontext handle to the parent context.
Returns NULL if an error occurs.

DESCRIPTION

cgGetProgramContext allows the application to retrieve a handle to the context to which a given program belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetProgramContext was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram, cgCreateContext

4.215 cgGetProgramDomain

NAME

cgGetProgramDomain - get a program’s domain

SYNOPSIS

#include <Cg/cg.h>

CGdomain cgGetProgramDomain( CGprogram program );

PARAMETERS

program

The program.

RETURN VALUES

Returns the domain enumerant associated with program.

DESCRIPTION

cgGetProgramDomain retrieves the domain enumerant currently associated with a program. This is a convenience routine which essentially calls cgGetProgramProfile followed by cgGetProfileDomain.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetProgramDomain was introduced in Cg 2.2.

SEE ALSO

cgGetDomain, cgGetDomainString

4.216 cgGetProgramDomainProfile

NAME

cgGetProgramDomainProfile - get the profile associated with a domain

SYNOPSIS

#include <Cg/cg.h>

CGprofile cgGetProgramDomainProfile( CGprogram program,
                                   int index );

PARAMETERS

  program
    The handle of the combined program object.
  index
    The index of the program’s domain to be queried.

RETURN VALUES

Returns the profile enumerant for the program with the given domain index, specifically one of:

  CG_UNKNOWN_DOMAIN
  CG_VERTEX_DOMAIN
  CG_FRAGMENT_DOMAIN
  CG_GEOMETRY_DOMAIN

Returns CG_PROFILE_UNKNOWN if an error occurs.

DESCRIPTION

cgGetProgramDomainProfile gets the profile of the passed combined program using the index to select which domain to choose.

EXAMPLES

/* This will enable all profiles for each domain in glslComboProgram */
int domains = cgGetProgramDomains(glslComboProgram);
for (int i=0; i<domains; i++) |

4.216. cgGetProgramDomainProfile
```c
CGprogram cgGetProgramDomainProgram( CGprogram program, int index );
```

### PARAMETERS

- `program`
  - The handle of the combined program object.
- `index`
  - The index of the program’s domain program to be queried.

### RETURN VALUES

- Returns the program handle for the program with the given domain index.
- Returns 0 if an error occurs.

### DESCRIPTION

A combined program consists of multiple domain programs. For example, a combined program may contain a vertex domain program and a fragment domain program. `cgGetProgramDomainProgram` gets the indexed domain program of the specified combined program.

If the program parameter is not a combined program and the index is zero, program handle is simply returned as-is without error.

**ERRORS**

- `CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.
- `CG_INVALID_PARAMETER_ERROR` is generated if `index` is less than 0 or greater than or equal to the number of domains in `program`.

**HISTORY**

`cgGetProgramDomainProfile` was introduced in Cg 1.5.

**SEE ALSO**

`cgGetNumProgramDomains`, `cgGetProfileDomain`
EXAMPLES

/* This will enable all profiles for each domain in glslComboProgram */
int domains = cgGetNumProgramDomains(glslComboProgram);
for (int i=0; i<domains; i++) {
    CGprogram subprog = cgGetProgramDomainProgram(glslComboProgram, i);
    CGparameter param = cgGetFirstParameter(subprog);
    while (param) {
        // Do something to each parameter of each domain program
        param = cgGetNextParameter(param);
    }
}

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.
CG_INVALID_PARAMETER_ERROR is generated if index is less than 0 or greater than or equal to the number of domains in program.

HISTORY

cgGetProgramDomainProgram was introduced in Cg 2.1.

SEE ALSO

cgGetNumProgramDomains, cgGetProfileDomain, cgGetProgramDomainProfile

4.218 cgGetProgramInput

NAME

cgGetProgramInput - get the program’s input

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetProgramInput( CGprogram program );

PARAMETERS

program

A program handle.

RETURN VALUES

Returns a program input enumerant. If the program is a vertex or fragment program, it returns CG_VERTEX or CG_FRAGMENT, respectively. For geometry programs the input is one of: CG_POINT, CG_LINE, CG_LINE_ADJ, CG_TRIANGLE, or CG_TRIANGLE_ADJ. For tessellation control and evaluation programs the input is CG_PATCH.

Returns CG_UNKNOWN if the input is unknown.

DESCRIPTION

cgGetProgramInput returns the program input enumerant.
EXAMPLES

```c
void printProgramInput(CGprogram program)
{
    char * input = NULL;
    switch(cgGetProgramInput(program))
    {
        case CG_FRAGMENT:
            input = "fragment";
            break;
        case CG_VERTEX:
            input = "vertex";
            break;
        case CG_POINT:
            input = "point";
            break;
        case CG_LINE:
            input = "line";
            break;
        case CG_LINE_ADJ:
            input = "line adjacency";
            break;
        case CG_TRIANGLE:
            input = "triangle";
            break;
        case CG_TRIANGLE_ADJ:
            input = "triangle adjacency";
            break;
        case CG_PATCH:
            input = "patch";
            break;
        default:
            input = "unknown";
            break;
    }
    printf("Program inputs %s.\n", input);
}
```

ERRORS

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not valid program handle.

HISTORY

cgGetProgramInput was introduced in Cg 2.0.

SEE ALSO

cgGetProgramOutput

4.219 cgGetProgramOptions

NAME

`cgGetProgramOptions` - get strings from a program object
SYNOPSIS

#include <Cg/cg.h>

char const * const * cgGetProgramOptions( CGprogram program );

PARAMETERS

program

The Cg program to query.

RETURN VALUES

Returns the options used to compile the program as an array of null-terminated strings. Returns NULL if no options exist, or if an error occurs.

DESCRIPTION

cgGetProgramOptions allows the application to retrieve the set of options used to compile the program. The options are returned in an array of ASCII-encoded null-terminated character strings. Each string contains a single option. The last element of the string array is guaranteed to be NULL.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetProgramOptions was introduced in Cg 1.4.

SEE ALSO

cgGetProgramString

4.220  cgGetProgramOutput

NAME

cgGetProgramOutput - get the program’s output

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetProgramOutput( CGprogram program );

PARAMETERS

program

A program handle.
RETURN VALUES

Returns a program output enumerant. If the program is a vertex or fragment program, it returns CG_VERTEX or CG_FRAGMENT, respectively. For geometry programs the output is one of: CG_POINT_OUT, CG_LINE_OUT, or CG_TRIANGLE_OUT. For tessellation control programs the output is CG_PATCH.

Returns CG_UNKNOWN if the output is unknown.

DESCRIPTION

cgGetProgramOutput returns the program output enumerant.

For geometry programs, an input must be specified but not an output because of implicit output defaults. For example, if either “TRIANGLE” or “TRIANGLE_ADJ” is specified as an input without an explicit output in the shader source, then cgGetProgramOutput will return CG_TRIANGLE_OUT.

EXAMPLES

```c
void printProgramOutput(CGprogram program) {
    char * output = NULL;
    switch (cgGetProgramOutput (program)) {
        case CG_VERTEX:
            output = "vertex";
            break;
        case CG_FRAGMENT:
            output = "fragment";
            break;
        case CG_POINT_OUT:
            output = "point";
            break;
        case CG_LINE_OUT:
            output = "line";
            break;
        case CG_TRIANGLE_OUT:
            output = "triangle";
            break;
        case CG_PATCH:
            output = "patch";
            break;
        default:
            output = "unknown";
            break;
    }
    printf("Program outputs %s\n", output);
}
```

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetProgramOutput was introduced in Cg 2.0.

SEE ALSO

cgGetProgramInput
4.221 cgGetProgramOutputVertices

NAME

**cgGetProgramOutputVertices** - get the maximum number of geometry program output vertices

SYNOPSIS

```c
#include <Cg/cg.h>

int cgGetProgramOutputVertices( CGprogram program );
```

PARAMETERS

- **program**
  The program.

RETURN VALUES

Returns the maximum number of geometry program output vertices.

Returns **-1** if the maximum has not been specified by **cgSetProgramOutputVertices**.

DESCRIPTION

**cgGetProgramOutputVertices** queries the application-specified maximum number of output vertices for a geometry program.

EXAMPLES

*to-be-written*

ERRORS

**CG_INVALID_PROGRAM_HANDLE_ERROR** is generated if **program** is not a valid program handle.

HISTORY

**cgGetProgramOutputVertices** was introduced in Cg 3.1.

SEE ALSO

**cgSetProgramOutputVertices**, **cgGetProgramProfile**, **gp4gp**, **gp5gp**, **glslg**.

4.222 cgGetProgramProfile

NAME

**cgGetProgramProfile** - get a program’s profile

SYNOPSIS

```c
#include <Cg/cg.h>

CGprofile cgGetProgramProfile( CGprogram program );
```
PARAMETERS

program

The program.

RETURN VALUES

Returns the profile enumerant associated with program.

DESCRIPTION

cgGetProgramProfile retrieves the profile enumerant currently associated with a program.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGetProgramProfile was introduced in Cg 1.1.

SEE ALSO

cgSetProgramProfile, cgGetProfile, cgGetProfileString, cgCreateProgram

4.223 cgGetProgramStateAssignmentValue

NAME

cgGetProgramStateAssignmentValue - get a program-valued state assignment’s values

SYNOPSIS

#include <Cg/cg.h>

CGprogram cgGetProgramStateAssignmentValue( CGstateassignment sa );

PARAMETERS

sa

The state assignment.

RETURN VALUES

Returns a CGprogram handle.

Returns NULL if an error occurs or no program is available.

DESCRIPTION

cgGetProgramStateAssignmentValue allows the application to retrieve the value(s) of a state assignment that stores a CGprogram.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a program type.

HISTORY

cgGetProgramStateAssignmentValue was introduced in Cg 1.4.

SEE ALSO

cgGetStateAssignmentState, cgGetStateType, cgGetFloatStateAssignmentValues, cgGetIntStateAssignmentValues, cgGetBoolStateAssignmentValues, cgGetStringStateAssignmentValue, cgGetSamplerStateAssignmentValue, cgGetTextureStateAssignmentValue

4.224 cgGetProgramString

NAME

cgGetProgramString - get strings from a program object

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetProgramString( CGprogram program,
                                  CGenum enum );

PARAMETERS

  program
      The program to query.

  enum
      Specifies the string to retrieve. enum can be one of CG_PROGRAM_SOURCE, CG_PROGRAM_ENTRY, CG_PROGRAM_PROFILE, or CG_COMPILED_PROGRAM.

RETURN VALUES

Returns a null-terminated string based on the value of enum.
Returns an empty string if an error occurs.

DESCRIPTION

cgGetProgramString allows the application to retrieve program strings that have been set via functions that modify program state.

When enum is CG_PROGRAM_SOURCE the original Cg source program is returned.
When enum is CG_PROGRAM_ENTRY the main entry point for the program is returned.
When enum is CG_PROGRAM_PROFILE the profile for the program is returned.
When `enum` is `CG_COMPILED_PROGRAM` the string for the compiled program is returned.

**EXAMPLES**

```c
CGcontext context = cgCreateContext();
CGprogram program = cgCreateProgramFromFile(context,
    CG_SOURCE,
    mysourcefilename,
    CG_PROFILE_ARBVP1,
    "myshader",
    NULL);

if (cgIsProgramCompiled(program))
    printf("%s\n", cgGetProgramString(program, CG_COMPILED_PROGRAM));
```

**ERRORS**

`CG_INVALID_PROGRAM_HANDLE_ERROR` is generated if `program` is not a valid program handle.

`CG_INVALID_ENUMERANT_ERROR` is generated if `enum` is not `CG_PROGRAM_SOURCE`, `CG_PROGRAM_ENTRY`, `CG_PROGRAM_PROFILE`, or `CG_COMPILED_PROGRAM`.

**HISTORY**

`cgGetProgramString` was introduced in Cg 1.1.

**SEE ALSO**

`cgCreateProgram, cgGetProgramOptions`

### 4.225 cgGetResource

**NAME**

`cgGetResource` - get the resource enumerant assigned to a resource name

**SYNOPSIS**

```c
#include <Cg/cg.h>
CGresource cgGetResource( const char * resource_string );
```

**PARAMETERS**

- `resource_string`
  
  A string containing the resource name.

**RETURN VALUES**

Returns the resource enumerant of `resource_string`.

Returns `CG_UNKNOWN` if no such resource exists.

**DESCRIPTION**

`cgGetResource` returns the enumerant assigned to a resource name.
EXAMPLES

CGresource PositionResource = cgGetResource("POSITION");

if(cgGetParameterResource(myparam) == PositionResource)
{
    /* Do stuff to the "POSITION" parameter */
}

ERRORS

None.

HISTORY

cgGetResource was introduced in Cg 1.1.

SEE ALSO

cgGetResourceString, cgGetParameterResource

4.226  cgGetResourceString

NAME

cgGetResourceString - get the resource name associated with a resource enumerant

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetResourceString( CGresource resource );

PARAMETERS

    resource

    The resource enumerant.

RETURN VALUES

Returns the null-terminated resource string of the enumerant resource.

DESCRIPTION

cgGetResourceString returns the resource name associated with a resource enumerant.

EXAMPLES

/* log info about parameter param for debugging */

printf("Resource: %s:%d (base %s)\n",
    cgGetResourceString(cgGetParameterResource(param)),
    cgGetParameterResourceIndex(param),
    cgGetResourceString(cgGetParameterBaseResource(param)));

ERRORS

None.
HISTORY

cgGetResourceString was introduced in Cg 1.1.

SEE ALSO

cgGetResource, cgGetParameterResource

4.227 cgGetSamplerStateAssignmentParameter

NAME

cgGetSamplerStateAssignmentParameter - get the sampler parameter being set up given a state assignment in its sampler_state block

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetSamplerStateAssignmentParameter( CGstateassignment sa );

PARAMETERS

sa

The state assignment in a sampler_state block

RETURN VALUES

Returns a handle to a parameter.
Returns NULL if sa is not a state assignment in a sampler_state block.

DESCRIPTION

Given the handle to a state assignment in a sampler_state block in an effect file, cgGetSamplerStateAssignmentParameter returns a handle to the sampler parameter being initialized.

EXAMPLES

Given an effect file with:

sampler2D foo = sampler_state { GenerateMipmap = true; }

cgGetSamplerStateAssignmentParameter returns a handle to foo if passed a handle to the GenerateMipmap state assignment.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetSamplerStateAssignmentParameter was introduced in Cg 1.4.

SEE ALSO

cgIsStateAssignment, cgIsParameter
4.228 cgGetSamplerStateAssignmentState

NAME

cgGetSamplerStateAssignmentState - get a sampler-valued state assignment’s state

SYNOPSIS

#include <Cg/cg.h>

CGstate cgGetSamplerStateAssignmentState( CGstateassignment sa );

PARAMETERS

   sa
     The state assignment.

RETURN VALUES

Returns a CGstate handle for the state.
Returns NULL if the handle sa is invalid.

DESCRIPTION

cgGetSamplerStateAssignmentState allows the application to retrieve the state of a state assignment that stores a sampler.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetSamplerStateAssignmentState was introduced in Cg 1.4.

SEE ALSO

cgGetFirstSamplerStateAssignment, cgGetNamedSamplerStateAssignment, cgGetSamplerStateAssignmentParameter, cgGetSamplerStateAssignmentValue

4.229 cgGetSamplerStateAssignmentValue

NAME

cgGetSamplerStateAssignmentValue - get a sampler-valued state assignment’s values

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetSamplerStateAssignmentValue( CGstateassignment sa );
PARAMETERS

   sa
       The state assignment.

RETURN VALUES

Returns a CGparameter handle for the sampler.
Returns NULL if an error occurs.

DESCRIPTION

cgGetSamplerStateAssignmentValue allows the application to retrieve the value(s) of a state assignment that stores a sampler.

EXAMPLES

  to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a sampler type.

HISTORY

cgGetSamplerStateAssignmentValue was introduced in Cg 1.4.

SEE ALSO

cgGetStateAssignmentState, cgGetStateType, cgGetFloatStateAssignmentValues, cgGetIntStateAssignmentValues, cgGetBoolStateAssignmentValues, cgGetStringStateAssignmentValue, cgGetProgramStateAssignmentValue, cgGetTextureStateAssignmentValue

4.230 cgGetSemanticCasePolicy

NAME

cgGetSemanticCasePolicy - get semantic case policy

SYNOPSIS

#include <Cg/cg.h>

CGenum cgGetSemanticCasePolicy( void );

PARAMETERS

None.

RETURN VALUES

Returns an enumerant indicating the current semantic case policy.
DESCRIPTION

cgGet Semantic Case Policy returns an enumerant indicating the current semantic case policy for the library. See cgSet Semantic Case Policy for more information.

EXAMPLES

CGenum currentSemanticCasePolicy = cgGetSemanticCasePolicy();

ERRORS

None.

HISTORY

cgGet Semantic Case Policy was introduced in Cg 2.0.

SEE ALSO

cgSet Semantic Case Policy, cgGet Parameter Semantic, cgSet Parameter Semantic

4.231 cgGet State Assignment Index

NAME

cgGet State Assignment Index - get the array index of a state assignment for array-valued state

SYNOPSIS

#include <Cg/cg.h>

int cgGetStateAssignmentIndex( CGstateassignment sa );

PARAMETERS

sa

The state assignment.

RETURN VALUES

Returns an integer index value.

Returns 0 if the CGstate for this state assignment is not an array type.

DESCRIPTION

cgGet State Assignment Index returns the array index of a state assignment if the state it is based on is an array type.

EXAMPLES

Given a “LightPosition” state defined as an array of eight float3 values and an effect file with the following state assignment:

pass { LightPosition[3] = float3(10,0,0); }

cgGet State Assignment Index will return 3 when passed a handle to this state assignment.

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
HISTORY

cgGetStateAssignmentIndex was introduced in Cg 1.4.

SEE ALSO

cgIsStateAssignment, cgCreateStateAssignmentIndex

4.232 cgGetStateAssignmentPass

NAME

cgGetStateAssignmentPass - get a state assignment’s pass

SYNOPSIS

#include <Cg/cg.h>

CGpass cgGetStateAssignmentPass( CGstateassignment sa );

PARAMETERS

   sa

      The state assignment.

RETURN VALUES

Returns a CGpass handle to the pass.
Returns NULL if an error occurs.

DESCRIPTION

cgGetStateAssignmentPass allows the application to retrieve a handle to the pass to which a given stateassignment belongs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

HISTORY

cgGetStateAssignmentPass was introduced in Cg 1.4.

SEE ALSO

cgIsStateAssignment, cgIsPass

4.233 cgGetStateAssignmentState

NAME

cgGetStateAssignmentState - returns the state type of a particular state assignment
SYNOPSIS

#include <Cg/cg.h>

CGstate cgGetStateAssignmentState( CGstateassignment sa );

PARAMETERS

    sa

    The state assignment handle.

RETURN VALUES

Returns the state corresponding to the given state assignment.
Returns NULL if an error occurs.

DESCRIPTION

cgGetStateAssignmentState returns the CGstate object that corresponds to a particular state assignment in a pass. This object can then be queried to find out its type, giving the type of the state assignment.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_INVALID_STATE_HANDLE_ERROR is generated if the effect doesn’t contain a state matching the given state assignment.

HISTORY

cgGetStateAssignmentState was introduced in Cg 1.4.

SEE ALSO

cgGetStateType, cgCreateState, cgCreateArrayState

4.234 cgGetStateContext

NAME

cgGetStateContext - get a state’s context

SYNOPSIS

#include <Cg/cg.h>

CGcontext cgGetStateContext( CGstate state );

PARAMETERS

    state

    The state.
RETURN VALUES

Returns the context to which state belongs.
Returns NULL if an error occurs.

DESCRIPTION

cgGetStateContext allows the application to retrieve the context of a state. This is the context used to create the state with cgCreateState.

EXAMPLES

CGcontext context = cgCreateContext();
CGstate state = cgCreateState(context, "GreatStateOfTexas", CG_FLOAT);
assert(context == cgGetStateContext(state));

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY
cgGetStateContext was introduced in Cg 1.5.

SEE ALSO

cgCreateState, cgCreateArrayState, cgGetEffectContext, cgGetParameterContext, cgGetProgramContext

4.235 cgGetStateEnumerant

NAME
cgGetStateEnumerant - get a state enumerant name and value by index

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetStateEnumerant( CGstate state, int index, int * value );

PARAMETERS

state

The state from which to retrieve an enumerant name and value.

index

The index for the enumerant in state.

value

Pointer to integer where the enumerant value will be stored.

RETURN VALUES

Returns the null-terminated enumerant name string associated with state at position index. The enumerant value is returned via the value parameter.

Returns NULL if an error occurs. value will be 0.
DESCRIPTION

**cgGetStateEnumerant** allows the application to retrieve the enumerant name and value associated with a **CGstate** at a specified index location. The number of enumerants associated with a state can be discovered using **cgGetNumStateEnumerants**.

EXAMPLES

```c
int value;
char* pName;

int nEnums = cgGetNumStateEnumerants(state);

for (ii=0; ii<nEnums; ++ii) {
    pName = cgGetStateEnumerant(state, ii, &value);
    printf("%i: %s %i\n", ii+1, pName, value);
}
```

ERRORS

**CG_INVALID_STATE_HANDLE_ERROR** is generated if **state** is not a valid state.

**CG_INVALID_POINTER_ERROR** is generated if **value** is NULL.

**CG_INVALID_PARAMETER_ERROR** is generated if **index** is less than 0 or **index** is greater than or equal to the number of enumerants associated with **state**.

HISTORY

**cgGetStateEnumerant** was introduced in Cg 2.2.

SEE ALSO

**cgAddStateEnumerant**, **cgGetNumStateEnumerants**, **cgGetStateEnumerantName**, **cgGetStateEnumerantValue**

---

4.236. **cgGetStateEnumerantName**

NAME

**cgGetStateEnumerantName** - get a state enumerant name by value

SYNOPSIS

```c
#include <Cg/cg.h>

const char* cgGetStateEnumerantName( CGstate state,
                                 int value);
```

PARAMETERS

state
The state from which to retrieve an enumerant name.

value
The enumerant value for which to retrieve the associated name.
RETURN VALUES
Returns the null-terminated enumerant name string associated with the given enumerant value in state.
Returns NULL if an error occurs.

DESCRIPTION
cgGetStateEnumerantName returns the enumerant name associated with a given enumerant value from a specified state.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.
CG_INVALID_PARAMETER_ERROR is generated if state does not contain an enumerant defined for value.

HISTORY
cgGetStateEnumerantName was introduced in Cg 1.5.

SEE ALSO
cgGetStateEnumerantValue, cgAddStateEnumerant, cgIsState

4.237 cgGetStateEnumerantValue

NAME
cgGetStateEnumerantValue - get state enumerant value by name

SYNOPSIS
#include <Cg/cg.h>

int cgGetStateEnumerantValue( CGstate state,
                                      const char * name );

PARAMETERS
state
The state from which to retrieve the value associated with name.
name
The enumerant name for which to retrieve the associated value from state.

RETURN VALUES
Returns the enumerant value associated with name.
Returns -1 if an error occurs.

DESCRIPTION
cgGetStateEnumerantValue retrieves the enumerant value associated with a given enumerant name from the specified state.
**EXAMPLES**

_to-be-written_

**ERRORS**

**CG_INVALID_STATE_HANDLE_ERROR** is generated if `state` is not a valid state.

**CG_INVALID_PARAMETER_ERROR** is generated if `state` does not contain `name`, if `name` is `NULL`, or if `name` points to an empty string.

**HISTORY**

cgGetStateEnumerantValue was introduced in Cg 1.5.

**SEE ALSO**

cgGetStateEnumerantName, cgAddStateEnumerant, cgIsState

### 4.238 cgGetStateLatestProfile

**NAME**

cgGetStateLatestProfile - gets a state’s designated latest profile

**SYNOPSIS**

```c
#include <Cg/cg.h>
CGprofile cgGetStateLatestProfile( CGstate state );
```

**PARAMETERS**

- `state` The state handle.

**RETURN VALUES**

Returns the designated latest profile if `state` is of type `CG_PROGRAM_TYPE`.

Returns `CG_PROFILE_UNKNOWN` otherwise.

**DESCRIPTION**

cgGetStateLatestProfile gets the specified state’s designated latest profile for states of type `CG_PROGRAM_TYPE`.

This profile is used to compile the program for a state assignment for the state where the profile in the `compile` statement is the identifier `latest`.

**EXAMPLES**

Get the latest profile for fragment programs:

```c
CGstate state = cgGetNamedState(context, "FragmentProgram");
CGprofile profile = cgGetStateLatestProfile(state);
```

**ERRORS**

**CG_INVALID_STATE_HANDLE_ERROR** is generated if `state` is not a valid state.
HISTORY
cgGetStateLatestProfile was introduced in Cg 2.2.

SEE ALSO
cgGetNamedState, cgSetStateLatestProfile

4.239 cgGetStateName

NAME
cgGetStateName - get a state’s name

SYNOPSIS

#include <Cg/cg.h>
const char * cgGetStateName( CGstate state );

PARAMETERS

state
The state.

RETURN VALUES

Returns the null-terminated name string for the state.
Returns NULL if state is invalid.

DESCRIPTION

cgGetStateName allows the application to retrieve the name of a state defined in a Cg context. This name can be used later to retrieve the state from the context using cgGetNamedState.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgGetStateName was introduced in Cg 1.4.

SEE ALSO
cgGetNamedState, cgGetFirstState, cgGetNextState

4.240 cgGetStateResetCallback

NAME
cgGetStateResetCallback - get the state resetting callback function for a state
SYNOPSIS

#include <Cg/cg.h>

CGstatecallback cgGetStateResetCallback( CGstate state );

PARAMETERS

state

The state from which to retrieve the callback.

RETURN VALUES

Returns a pointer to the state resetting callback function.

Returns NULL if state is not a valid state or if it has no callback.

DESCRIPTION

cgGetStateResetCallback returns the callback function used for resetting the state when the given state is encountered in a pass in a technique. See cgSetStateCallbacks for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgGetStateResetCallback was introduced in Cg 1.4.

SEE ALSO

cgSetStateCallbacks, cgCallStateResetCallback, cgResetPassState

4.241 cgGetStateSetCallback

NAME

cgGetStateSetCallback - get the state setting callback function for a state

SYNOPSIS

#include <Cg/cg.h>

CGstatecallback cgGetStateSetCallback( CGstate state );

PARAMETERS

state

The state from which to retrieve the callback.
**RETURN VALUES**

Returns a pointer to the state setting callback function.
Returns NULL if state is not a valid state or if it has no callback.

**DESCRIPTION**

`cgGetStateSetCallback` returns the callback function used for setting the state when the given state is encountered in a pass in a technique. See `cgSetStateCallbacks` for more information.

**EXAMPLES**

to-be-written

**ERRORS**

`CG_INVALID_STATE_HANDLE_ERROR` is generated if state is not a valid state.

**HISTORY**

`cgGetStateSetCallback` was introduced in Cg 1.4.

**SEE ALSO**

`cgSetStateCallbacks, cgCallStateSetCallback, cgSetPassState`

---

**4.242 cgGetStateType**

**NAME**

`cgGetStateType` - returns the type of a given state

**SYNOPSIS**

```c
#include <Cg/cg.h>

CGtype cgGetStateType( CGstate state );
```

**PARAMETERS**

- `state`
  - The state from which to retrieve the type.

**RETURN VALUES**

Returns the `CGtype` of the given state.

**DESCRIPTION**

`cgGetStateType` returns the type of a state that was previously defined via `cgCreateState, cgCreateArrayState, cgCreateSamplerState, or cgCreateArraySamplerState.`

**EXAMPLES**

to-be-written

**ERRORS**

`CG_INVALID_STATE_HANDLE_ERROR` is generated if `state` is not a valid state.
HISTORY

cgGetStateType was introduced in Cg 1.4.

SEE ALSO

cgCreateState, cgCreateArrayState, cgCreateSamplerState, cgCreateArraySamplerState, cgGetStateName

4.243 cgGetStateValidateCallback

NAME

cgGetStateValidateCallback - get the state validation callback function for a state

SYNOPSIS

#include <Cg/cg.h>

CGstatecallback cgGetStateValidateCallback( CGstate state );

PARAMETERS

state

The state from which to retrieve the callback.

RETURN VALUES

Returns a pointer to the state validating callback function.
Returns NULL if state is not a valid state or if it has no callback.

DESCRIPTION

cgGetStateValidateCallback returns the callback function used for validating the state when the given state is encountered in a pass in a technique. See cgSetStateCallbacks and cgCallStateValidateCallback for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgGetStateValidateCallback was introduced in Cg 1.4.

SEE ALSO

cgSetStateCallbacks, cgCallStateValidateCallback, cgValidateTechnique

4.244 cgGetString

NAME

cgGetString - gets a special string
SYNOPSIS
#include <Cg/cg.h>

const char * cgGetString( CGenum enum );

PARAMETERS
enum
An enumerant describing the string to be returned.

RETURN VALUEs
Returns the string associated with enum.
Returns NULL in the event of an error.

DESCRIPTION
cgGetString returns an informative string depending on the enum. Currently there is only one valid enumerant that may be passed in.

CG_VERSION
Returns the version string of the Cg runtime and compiler.

EXAMPLES
to-be-written

ERRORS
CG_INVAlID ENUMERANT_ERROR is generated if enum is not CG_VERSION.

HISTORY
cgGetString was introduced in Cg 1.2.

SEE ALSO
Cg

4.245 cgGetStringAnnotationValue

NAME
cgGetStringAnnotationValue - get a string-valued annotation’s value

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetStringAnnotationValue( CGannotation ann );

PARAMETERS
ann
The annotation.
RETURN VALUES

Returns a pointer to a string contained by `ann`. Returns NULL if no value is available.

DESCRIPTION

cgGetStringAnnotationValue allows the application to retrieve the value of a string typed annotation.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if `ann` is not a valid annotation.

HISTORY

cgGetStringAnnotationValue was introduced in Cg 1.4.

SEE ALSO

cgGetAnnotationType, cgGetStringAnnotationValues, cgGetFloatAnnotationValues, cgGetIntAnnotationValues, cgGetBoolAnnotationValues

4.246 cgGetStringAnnotationValues

NAME

cgGetStringAnnotationValues - get the values from a string-valued annotation

SYNOPSIS

```
#include <Cg/cg.h>

const char * const * cgGetStringAnnotationValues( CGannotation ann,
                                              int * nvalues );
```

PARAMETERS

- `ann`  
The annotation from which the values will be retrieved.
- `nvalues`  
  Pointer to integer where the number of returned values will be stored.

RETURN VALUES

Returns a pointer to an array of string values. The number of values in the array is returned via the `nvalues` parameter. Returns NULL if no values are available, `ann` is not string-typed, or an error occurs. `nvalues` will be 0.

DESCRIPTION

cgGetStringAnnotationValues allows the application to retrieve the value(s) of a string typed annotation.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if \texttt{ann} is not a valid annotation.

CG_INVALID_PARAMETER_ERROR is generated if \texttt{nvalues} is \texttt{NULL}.

HISTORY

cgGetStringAnnotationValues was introduced in Cg 2.0.

SEE ALSO

\texttt{cgGetAnnotationType}, \texttt{cgGetStringAnnotationValue}, \texttt{cgGetBoolAnnotationValues}, \texttt{cgGetFloatAnnotationValues}, \texttt{cgGetIntAnnotationValues}

4.247 cgGetStringParameterValue

NAME

cgGetStringParameterValue - get the value of a string parameter

SYNOPSIS

\begin{verbatim}
#include <Cg/cg.h>

const char * cgGetStringParameterValue( CGparameter param );
\end{verbatim}

PARAMETERS

param

The parameter whose value will be retrieved.

RETURN VALUES

Returns a pointer to the string contained by a string parameter.

Returns \texttt{NULL} if the parameter does not contain a valid string value.

DESCRIPTION

cgGetStringParameterValue allows the application to get the value of a string parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if \texttt{param} is not a valid parameter.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if \texttt{param} is not string-typed.

HISTORY

cgGetStringParameterValue was introduced in Cg 1.4.
4.248 cgGetStringStateAssignmentValue

NAME

cgGetStringStateAssignmentValue - get a string-valued state assignment’s values

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetStringStateAssignmentValue( CGstateassignment sa );

PARAMETERS

sa

The state assignment.

RETURN VALUES

Returns a pointer to a string.

Returns NULL if an error occurs.

DESCRIPTION

cgGetStringStateAssignmentValue allows the application to retrieve the value(s) of a string typed state assignment.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a string type.

HISTORY

cgGetStringStateAssignmentValue was introduced in Cg 1.4.

SEE ALSO

cgGetStateAssignmentState, cgGetStateType, cgGetFloatStateAssignmentValues, cgGetIntStateAssignmentValues, cgGetBoolStateAssignmentValues, cgGetProgramStateAssignmentValue, cgGetSamplerStateAssignmentValue, cgGetTextureStateAssignmentValue

4.249 cgGetSupportedProfile

NAME

cgGetSupportedProfile - get a supported profile by index
SYNOPSIS

#include <Cg/cg.h>

CGprofile cgGetSupportedProfile( int index );

PARAMETERS

index

The index for the supported profile.

RETURN VALUES

Returns the supported CGprofile at position index.

Returns the CG_PROFILE_UNKNOWN if an error occurs.

DESCRIPTION

cgGetSupportedProfile retrieves by index a profile supported by this version of the Cg library. The number of supported profiles can be found using cgGetNumSupportedProfiles.

Note that a profile may be recognized by Cg but not supported by the platform on which the application is currently running. A graphics API specific routine such as cgGLIsProfileSupported must still be used to determine if the current GPU and driver combination supports a given profile.

EXAMPLES

CGprofile profile;
int nProfiles;
int ii;

nProfiles = cgGetNumSupportedProfiles();
printf("NumSupportedProfiles: %i\n", nProfiles);

for (ii=0; ii<nProfiles; ++ii) {
    profile = cgGetSupportedProfile(ii);
    printf("SupportedProfile %i: %s %i\n", ii, cgGetProfileString(profile), profile);
}

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if index is less than 0 or greater than or equal to the number of supported profiles returned by cgGetNumSupportedProfiles.

HISTORY

cgGetSupportedProfile was introduced in Cg 2.2.

SEE ALSO

cgGetNumSupportedProfiles, cgIsProfileSupported, cgGetProfileProperty, cgGLIsProfileSupported, cgD3D9IsProfileSupported, cgD3D10IsProfileSupported, cgGetProfileString, cgGetProfile
4.250  cgGetTechniqueEffect

NAME

cgGetTechniqueEffect - get a technique’s effect

SYNOPSIS

#include <Cg/cg.h>

CGeffect cgGetTechniqueEffect( CGtechnique tech );

PARAMETERS

    tech
    The technique.

RETURN VALUES

Returns a CGeffect handle to the effect.
Returns NULL if an error occurs.

DESCRIPTION

cgGetTechniqueEffect allows the application to retrieve a handle to the effect to which a given technique belongs.

EXAMPLES

    to-be-written

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetTechniqueEffect was introduced in Cg 1.4.

SEE ALSO

cgCreateEffect, cgCreateEffectFromFile

4.251  cgGetTechniqueName

NAME

.cgGetTechniqueName - get a technique’s name

SYNOPSIS

#include <Cg/cg.h>

const char * cgGetTechniqueName( CGtechnique tech );
PARAMETERS

    tech
    The technique.

RETURN VALUES

Returns the null-terminated name string for the technique.
Returns NULL if tech is invalid.

DESCRIPTION

cgGetTechniqueName allows the application to retrieve the name of a technique in a Cg effect. This name can be used later to retrieve the technique from the effect using cgGetNamedTechnique.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgGetTechniqueName was introduced in Cg 1.4.

SEE ALSO

cgGetNamedTechnique, cgGetFirstTechnique, cgGetNextTechnique

4.252 cgGetTextureStateAssignmentValue

NAME

cgGetTextureStateAssignmentValue - get a texture-valued state assignment’s values

SYNOPSIS

#include <Cg/cg.h>

CGparameter cgGetTextureStateAssignmentValue( CGstateassignment sa );

PARAMETERS

    sa
    The state assignment.

RETURN VALUES

Returns a handle to the texture parameter associated with this state assignment.
Returns NULL if an error occurs.

DESCRIPTION

cgGetTextureStateAssignmentValue allows the application to retrieve the value(s) of a state assignment that stores a texture parameter.
EXAMPLES
to-be-written

ERRORS
CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a texture type.

HISTORY
cgGetTextureStateAssignmentValue was introduced in Cg 1.4.

SEE ALSO
cgGetStateAssignmentState, cgGetStateType, cgGetFloatStateAssignmentValues, cgGetIntStateAssignmentValues, cgGetStringStateAssignmentValue, cgGetSamplerStateAssignmentValue

4.253 cgGetUniformBufferBlockName

NAME
cgGetUniformBufferBlockName - get block name from a uniform buffer parameter

SYNOPSIS
#include <Cg/cg.h>

const char * cgGetUniformBufferBlockName( CGparameter param );

PARAMETERS
param
The uniform buffer parameter.

RETURN VALUES
Returns the null-terminated block name string for the uniform buffer parameter.
Returns NULL if an error occurs.

DESCRIPTION
cgGetUniformBufferBlockName allows the application to retrieve the block name of a uniform buffer parameter in a Cg program or effect. This name can be used later to retrieve the parameter from the program or effect using cgGetNamedProgramUniformBuffer or cgGetNamedEffectUniformBuffer.

EXAMPLES
If the file buf.cg contains this shader:

uniform myBuf {
    float4 var;
} a : BUFFER;

float4 vertex() : POSITION
{
return float4(a.var.r, a.var.g, a.var.b, 1.0);
}

and if program refers to the CGprogram created from buf.cg, then the call sequence:

CParameter param = cgGetNamedParameter(program, "a");
const char * BlockName = cgGetUniformBufferBlockName(param);

results in BlockName pointing at the character string “myBuf”

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a uniform buffer parameter.

HISTORY

cgGetUniformBufferBlockName was introduced in Cg 3.1.

SEE ALSO

cgGetNamedParameter, cgGetNamedProgramUniformBuffer, cgGetNamedEffectUniformBuffer

4.254 cgGetUniformBufferParameter

NAME

cgGetUniformBufferParameter - get the buffer associated with a uniform buffer parameter

SYNOPSIS

#include <Cg/cg.h>

CGbuffer cgGetUniformBufferParameter( CGparameter param );

PARAMETERS

param

The parameter from which the associated buffer will be returned.

RETURN VALUES

Returns the handle of the CGbuffer object which is associated with param.

Returns NULL if no buffer is associated with param or an error occurs.

DESCRIPTION

cgGetUniformBufferParameter gets the buffer associated with a given uniform buffer parameter.

EXAMPLES

CGbuffer myBuffer = cgGetUniformBufferParameter( myParam );

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter handle.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if param is not a uniform buffer parameter.
HISTORY

cgGetUniformBufferParameter was introduced in Cg 3.1.

SEE ALSO

cgSetUniformBufferParameter, cgCreateBuffer, cgGLBindProgram, cgD3D9BindProgram

4.255 cgGetType

NAME
cgGetType - get the type enumerant assigned to a type name

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetType( const char * type_string );

PARAMETERS

type_string

A string containing the case-sensitive type name.

RETURN VALUES

Returns the type enumerant of type_string.
Returns CG_UNKNOWN_TYPE if no such type exists.

DESCRIPTION

cgGetType returns the enumerant assigned to a type name.

EXAMPLES

CGtype Float4Type = cgGetType("float4");

if(cgGetParameterType(myparam) == Float4Type)
{
  /* Do stuff */
}

ERRORS

None.

HISTORY

cgGetType was introduced in Cg 1.1.

SEE ALSO

cgGetTypeString, cgGetParameterType
4.256 cgGetTypeBase

NAME
cgGetTypeBase - get the base type associated with a type enumerant

SYNOPSIS
#include <Cg/cg.h>

CGtype cgGetTypeBase( CGtype type );

PARAMETERS

  type
  The type enumerant.

RETURN VALUES

Returns the scalar base type of the enumerant type.

DESCRIPTION

cgGetTypeBase returns the base (scalar) type associated with a type enumerant. For example, cgGetTypeBase(CG_FLOAT3x4) returns CG_FLOAT. The base type for a non-numeric type such as CG_STRING, CG_STRUCT, CG_SAMPLER2D, or user-defined types is simply the type itself.

EXAMPLES

to-be-written

ERRORS

None.

HISTORY

cgGetTypeBase was introduced in Cg 1.5.

SEE ALSO

cgGetType, cgGetTypeClass, cgGetParameterType

4.257 cgGetTypeClass

NAME

cgGetTypeClass - get the parameter class associated with a type enumerant

SYNOPSIS

#include <Cg/cg.h>

CGparameterclass cgGetTypeClass( CGtype type );
PARAMETERS

    type
    The type enumerant.

RETURN VALUES

Returns the parameter class of the enumerant type. Possible return values are:

* CG_PARAMETERCLASS_UNKNOWN
* CG_PARAMETERCLASS_SCALAR
* CG_PARAMETERCLASS_VECTOR
* CG_PARAMETERCLASS_MATRIX
* CG_PARAMETERCLASS_STRUCT
* CG_PARAMETERCLASS_ARRAY
* CG_PARAMETERCLASS_SAMPLER
* CG_PARAMETERCLASS_OBJECT

DESCRIPTION

cgGetTypeClass returns the parameter class associated with a type enumerant. For example, cgGetTypeClass(CG_FLOAT3x4) returns CG_PARAMETERCLASS_MATRIX while cgGetTypeClass(CG_HALF) returns CG_PARAMETERCLASS_SCALAR and cgGetTypeClass(CG_BOOL3) returns CG_PARAMETERCLASS_VECTOR.

CG_PARAMETERCLASS_UNKNOWN is returned if the type is unknown.

EXAMPLES

to-be-written

ERRORS

None

HISTORY

cgGetTypeClass was introduced in Cg 1.5.

SEE ALSO

cgGetType, cgGetTypeBase, cgGetParameterType

4.258 cgGetTypeSizes

NAME

cgGetTypeSizes - get the row and/or column size of a type enumerant

SYNOPSIS
#include <Cg/cg.h>

CGbool cgGetTypeSizes( CGtype type,  
    int  * nrows,  
    int  * ncols );

PARAMETERS

type
    The type enumerant.
nrows
    The location where the number of rows will be written.
ncols
    The location where the number of columns will be written.

RETURN VALUES

Returns CG_TRUE if the type enumerant is for a matrix.
Returns CG_FALSE otherwise.

DESCRIPTION

cgGetTypeSizes returns the number of rows and columns for enumerant type in the locations specified by nrows and ncols respectively.

When the type enumerant is not a matrix type then 1 is returned in nrows, in contrast to cgGetMatrixSize where the number of rows and columns will be 0 if the type enumerant is not a matrix.

For a numeric types, ncols will be the vector length for vectors and 1 for scalars. For non-numeric types, ncols will be 0.

EXAMPLES

to-be-written

ERRORS

None.

HISTORY

cgGetTypeSizes was introduced in Cg 1.5.

SEE ALSO

cgGetArrayTotalSize, cgGetArrayDimension, cgGetArrayParameter, cgGetMatrixSize

4.259  cgGetTypeString

NAME

cgGetTypeString - get the type name associated with a type enumerant
SYNOPSIS

#include <Cg/cg.h>

const char * cgGetTypeString( CGtype type );

PARAMETERS

type

The type enumerant.

RETURN VALUES

Returns the type string of the enumerant type.

DESCRIPTION

cgGetTypeString returns the type name associated with a type enumerant.

EXAMPLES

const char *MatrixTypeStr = cgGetTypeString(CG_FLOAT4x4);

/* MatrixTypeStr will be "float4x4" */

ERRORS

None.

HISTORY

cgGetTypeString was introduced in Cg 1.1.

SEE ALSO

cgGetType, cgGetParameterType

4.260  cgGetUserType

NAME

cgGetUserType - get enumerant of user-defined type from a program or effect

SYNOPSIS

#include <Cg/cg.h>

CGtype cgGetUserType( CGhandle handle,
        int index );

PARAMETERS

handle

The CGprogram or CGeffect in which the type is defined.

index
The index of the user-defined type. index must be greater than or equal to 0 and less than the value returned by cgGetNumUserTypes.

RETURN VALUES
Returns the type enumerant associated with the type with the given index.

DESCRIPTION
cgGetUserType returns the enumerant associated with the user-defined type with the given index in the given CG-program or CGeffect.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if handle is not a valid program or effect handle.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if index is outside the proper range.

HISTORY
cgGetUserType was introduced in Cg 1.2.

SEE ALSO
cgGetNumUserTypes, cgGetNamedUserType

4.261 cgIsAnnotation

NAME
cgIsAnnotation - determine if an annotation handle references a valid annotation

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsAnnotation( CGannotation ann );

PARAMETERS

ann
The annotation handle to check.

RETURN VALUES
Returns CG_TRUE if ann references a valid annotation.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsAnnotation returns CG_TRUE if ann references a valid annotation, CG_FALSE otherwise.

EXAMPLES
to-be-written
ERRORS
None.

HISTORY
cgIsAnnotation was introduced in Cg 1.4.

SEE ALSO
cgGetNextAnnotation, cgGetAnnotationName, cgGetAnnotationType, cgCreateEffectAnnotation, cgCreateParameterAnnotation, cgCreatePassAnnotation, cgCreateProgramAnnotation, cgCreateTechniqueAnnotation

4.262 cgIsBuffer

NAME
cgIsBuffer - determine if a buffer handle references a valid buffer

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsBuffer( CGbuffer buffer );

PARAMETERS
  buffer
  The buffer handle to check.

RETURN VALUES
Returns CG_TRUE if buffer references a valid buffer object.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsBuffer return CG_TRUE if buffer references a valid buffer object.

EXAMPLES

ERRORS
None.

HISTORY
cgIsBuffer was introduced in Cg 3.1.

SEE ALSO
 cgCreateBuffer, cgDestroyBuffer, cgGetProgramBuffer
4.263 cgIsContext

NAME
cgIsContext - determine if a context handle references a valid context

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsContext( CGcontext context );

PARAMETERS
context
    The context handle to check.

RETURN VALUES
Returns CG_TRUE if context references a valid context.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsContext returns CG_TRUE if context references a valid context, CG_FALSE otherwise.

EXAMPLES
CGcontext context = NULL;
cgIsContext(context); /* returns CG_FALSE */
context = cgCreateContext();
cgIsContext(context); /* returns CG_TRUE if create succeeded */

cgDestroyContext(context);
cgIsContext(context); /* returns CG_FALSE */

ERRORS
None.

HISTORY
cgIsContext was introduced in Cg 1.1.

SEE ALSO
cgCreateContext, cgDestroyContext

4.264 cgIsEffect

NAME
cgIsEffect - determine if an effect handle references a valid effect
SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsEffect( CGeffect effect );

PARAMETERS

  effect

     The effect handle to check.

RETURN VALUES

Returns CG_TRUE if effect references a valid effect.
Returns CG_FALSE otherwise.

DESCRIPTION

cgIsEffect returns CG_TRUE if effect references a valid effect, CG_FALSE otherwise.

EXAMPLES

to-be-written

ERRORS

None.

HISTORY

cgIsEffect was introduced in Cg 1.4.

SEE ALSO

cgCreateEffect, cgCreateEffectFromFile

4.265 cgIsInterfaceType

NAME

cgIsInterfaceType - determine if a type is an interface

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsInterfaceType( CGtype type );

PARAMETERS

  type

     The type being evaluated.

RETURN VALUES

Returns CG_TRUE if type is an interface (not just a struct).
Returns CG_FALSE otherwise.
DESCRIPTION

cgIsInterfaceType returns CG_TRUE if type is an interface (not just a struct), CG_FALSE otherwise.

EXAMPLES
to-be-written

ERRORS
None.

HISTORY

cgIsInterfaceType was introduced in Cg 1.2.

SEE ALSO

cgGetType

cgIsParameter

4.266 cgIsParameter

NAME

cgIsParameter - determine if a parameter handle references a valid parameter

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsParameter( CGparameter param );

PARAMETERS

param

The parameter handle to check.

RETURN VALUES

Returns CG_TRUE if param references a valid parameter object.

Returns CG_FALSE otherwise.

DESCRIPTION

cgIsParameter returns CG_TRUE if param references a valid parameter object. cgIsParameter is typically used for iterating through the parameters of an object. It can also be used as a consistency check when the application caches CGparameter handles. Certain program operations like deleting the program or context object that the parameter is contained in will cause a parameter object to become invalid.

EXAMPLES

if (cgIsParameter(param)) {
    /* do something with param */
} else {
    /* handle situation where param is not a valid parameter */
}
ERRORS
None.

HISTORY
cgIsParameter was introduced in Cg 1.1.

SEE ALSO
cgGetNextParameter

4.267 cgIsParameterGlobal

NAME
cgIsParameterGlobal - determine if a parameter is global

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsParameterGlobal( CGparameter param );

PARAMETERS
   param
       The parameter handle to check.

RETURN VALUES
Returns CG_TRUE if param is global.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsParameterGlobal returns CG_TRUE if param is a global parameter and CG_FALSE otherwise.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgIsParameterGlobal was introduced in Cg 1.2.

SEE ALSO
cgCreateParameter, cgIsParameter, cgIsParameterReferenced, cgIsParameterUsed
4.268  cgIsParameterReferenced

NAME

cgIsParameterReferenced - determine if a program parameter is potentially referenced

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsParameterReferenced( CGparameter param );

PARAMETERS

param

The handle of the parameter to check.

RETURN VALUES

Returns CG_TRUE if param is a program parameter and is potentially referenced by the program.

Returns CG_FALSE otherwise.

DESCRIPTION

cgIsParameterReferenced returns CG_TRUE if param is a program parameter, and is potentially referenced (used) within the program. It otherwise returns CG_FALSE.

Program parameters are those parameters associated directly with a CGprogram, whose handles are retrieved by calling, for example, cgGetNamedProgramParameter.

The value returned by cgIsParameterReferenced is conservative, but not always exact. A return value of CG_TRUE indicates that the parameter may be used by its associated program. A return value of CG_FALSE indicates that the parameter is definitively not referenced by the program.

If param is an aggregate program parameter (a struct or array), CG_TRUE is returned if any of param’s children are potentially referenced by the program.

If param is a leaf parameter and the return value is CG_FALSE, cgGetParameterResource may return CG_INVALID_VALUE for this parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgIsParameterReferenced was introduced in Cg 1.1.

SEE ALSO

cgGetNamedProgramParameter, cgIsParameterUsed, cgGetParameterResource
4.269 cgIsParameterUsed

NAME
cgIsParameterUsed - determine if a parameter is potentially used

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsParameterUsed( CGparameter param,
            CGhandle container );

PARAMETERS
  param
     The parameter to check.

  container
     Specifies the CGeffect, CGtechnique, CGpass, CGstateassignment, or CGprogram that may potentially use param.

RETURN VALUES
Returns CG_TRUE if param is potentially used by container.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsParameterUsed returns CG_TRUE if param is potentially used by the given container. If param is a struct or array, CG_TRUE is returned if any of its children are potentially used by container. It otherwise returns CG_FALSE.

The value returned by cgIsParameterUsed is conservative, but not always exact. A return value of CG_TRUE indicates that the parameter may be used by container. A return value of CG_FALSE indicates that the parameter is definitely not used by container.

The given param handle may reference a program parameter, an effect parameter, or a shared parameter.

The container handle may reference a CGeffect, CGtechnique, CGpass, CGstateassignment, or CGprogram.

If container is a CGprogram, CG_TRUE is returned if any of the program’s referenced parameters inherit their values directly or indirectly (due to parameter connections) from param.

If container is a CGstateassignment, CG_TRUE is returned if the right-hand side of the state assignment may directly or indirectly depend on the value of param. If the state assignment involves a CGprogram, the program’s parameters are also considered, as above.

If container is a CGpass, CG_TRUE is returned if any of the pass’ state assignments potentially use param.

If container is a CGtechnique, CG_TRUE is returned if any of the technique’s passes potentially use param.

If container is a CGeffect, CG_TRUE is returned if any of the effect’s techniques potentially use param.

EXAMPLES
to-be-written
ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter, or if container is not the handle of a valid container.

HISTORY

cgIsParameterUsed was introduced in Cg 1.4.

SEE ALSO

cgIsParameterReferenced, cgConnectParameter

4.270 cgIsParentType

NAME

cgIsParentType - determine if a type is a parent of another type

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsParentType( CGtype parent,
                        CGtype child );

PARAMETERS

  parent
  The parent type.

  child
  The child type.

RETURN VALUES

Returns CG_TRUE if parent is a parent type of child.

Otherwise, CG_FALSE is returned.

DESCRIPTION

cgIsParentType returns CG_TRUE if parent is a parent type of child. Otherwise CG_FALSE is returned.

EXAMPLES

  to-be-written

ERRORS

None.

HISTORY

cgIsParentType was introduced in Cg 1.2.

SEE ALSO

cgGetParentType
4.271 cgIsPass

NAME
cgIsPass - determine if a pass handle references a valid pass

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsPass( CGpass pass );

PARAMETERS

pass
The pass handle to check.

RETURN VALUES
Returns CG_TRUE if pass references a valid pass.
Returns CG_FALSE otherwise.

DESCRIPTION
cgIsPass returns CG_TRUE if pass references a valid pass, CG_FALSE otherwise.

EXAMPLES

to-be-written

ERRORS
None.

HISTORY
cgIsPass was introduced in Cg 1.4.

SEE ALSO

4.272 cgIsProfileSupported

NAME
cgIsProfileSupported - determine if a profile is supported

SYNOPSIS
#include <Cg/cg.h>

CGbool cgIsProfileSupported( CGprofile profile );
PARAMETERS

profile

The profile enumerant to test.

RETURN VALUES

Returns CG_TRUE if profile is supported.

DESCRIPTION

cgIsProfileSupported checks whether profile is supported by this version of the Cg library.

Note that a profile may be recognized by Cg but not supported by the platform on which the application is currently running. A graphics API specific routine such as cgGLIsProfileSupported must still be used to determine if the current GPU and driver combination supports a given profile.

EXAMPLES

CGprofile profile;
int nProfiles;
int ii;
nProfiles = cgGetNumSupportedProfiles();
printf("NumSupportedProfiles: %i\n", nProfiles);

for (ii=0; ii<nProfiles; ++ii) {
    profile = cgGetSupportedProfile(ii);
    printf("IsProfileSupported %i: %s %i\n", ii, cgGetProfileString(profile),
            cgIsProfileSupported(profile));
}

ERRORS

None.

HISTORY

cgIsProfileSupported was introduced in Cg 2.2.

SEE ALSO

cgGetNumSupportedProfiles, cgGetSupportedProfile, cgGetProfileProperty, cgGLIsProfileSupported, cgD3D9IsProfileSupported, cgD3D10IsProfileSupported, cgGetProfileString, cgGetProfile

4.273 cgIsProgram

NAME

cgIsProgram - determine if a program handle references a program object

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsProgram( CGprogram program );
PARAMETERS

program

The program handle to check.

RETURN VALUES

Returns CG_TRUE if program references a valid program object.

Returns CG_FALSE otherwise.

DESCRIPTION

cgIsProgram return CG_TRUE if program references a valid program object. Note that this does not imply that the program has been successfully compiled.

EXAMPLES

char *programSource = ...;
CGcontext context = cgCreateContext();
CGprogram program = cgCreateProgram( context,
    CG_SOURCE,
    programSource,
    CG_PROFILE_ARBVP1,
    "myshader",
    NULL );

CGbool isProgram = cgIsProgram( program );

ERRORS

None.

HISTORY

cgIsProgram was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram, cgDestroyProgram, cgGetNextProgram

4.274 cgIsProgramCompiled

NAME

cgIsProgramCompiled - determine if a program has been compiled

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsProgramCompiled( CGprogram program );

PARAMETERS

program

The program.
RETURN VALUES

Returns **CG_TRUE** if **program** has been compiled.
Returns **CG_FALSE** otherwise.

DESCRIPTION

cgIsProgramCompiled returns **CG_TRUE** if **program** has been compiled and **CG_FALSE** otherwise.

EXAMPLES

to-be-written

ERRORS

**CG_INVALID_PROGRAM_HANDLE_ERROR** is generated if **program** is not a valid program handle.

HISTORY

cgIsProgramCompiled was introduced in Cg 1.1.

SEE ALSO

cgCompileProgram, cgSetAutoCompile

### 4.275 cgIsState

NAME

cgIsState - determine if a state handle references a valid state

SYNOPSIS

```c
#include <Cg/cg.h>

CGbool cgIsState( CGstate state );
```

PARAMETERS

- state
  - The state handle to check.

RETURN VALUES

Returns **CG_TRUE** if **state** references a valid state.
Returns **CG_FALSE** otherwise.

DESCRIPTION

cgIsState returns **CG_TRUE** if **state** references a valid state, **CG_FALSE** otherwise.

EXAMPLES

to-be-written

ERRORS

None.
HISTORY
cgIsState was introduced in Cg 1.4.

SEE ALSO
cgCreateState

4.276 cgIsStateAssignment

NAME
cgIsStateAssignment - determine if a state assignment handle references a valid Cg state assignment

SYNOPSIS

```c
#include <Cg/cg.h>

CGbool cgIsStateAssignment( CGstateassignment sa );
```

PARAMETERS

- **sa**
  The state assignment handle to check.

RETURN VALUES

- Returns **CG_TRUE** if **sa** references a valid state assignment.
- Returns **CG_FALSE** otherwise.

DESCRIPTION
cgIsStateAssignment returns **CG_TRUE** if **sa** references a valid state assignment, **CG_FALSE** otherwise.

EXAMPLES

```c
if (cgIsStateAssignment(sa)) {
  /* do something with sa */
} else {
  /* handle situation where sa is not a valid state assignment */
}
```

ERRORS

None.

HISTORY
cgIsStateAssignment was introduced in Cg 1.4.

SEE ALSO
cgCreateStateAssignment, cgCreateStateAssignmentIndex, cgGetFirstStateAssignment, cgGetFirstSamplerStateAssignment, cgGetNamedStateAssignment, cgGetNamedSamplerStateAssignment, cgGetNextStateAssignment, cgGetStateAssignmentIndex, cgGetStateAssignmentPass, cgGetStateAssignmentState
4.277 cgIsTechnique

NAME

cgIsTechnique - determine if a technique handle references a valid technique

SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsTechnique( CGtechnique tech );

PARAMETERS

    tech
    The technique handle to check.

RETURN VALUES

Returns CG_TRUE if tech references a valid technique.
Returns CG_FALSE otherwise.

DESCRIPTION

cgIsTechnique returns CG_TRUE if tech references a valid technique, CG_FALSE otherwise.

EXAMPLES

if (cgIsTechnique(tech)) {
    /* do something with tech */
} else {
    /* handle situation where tech is not a valid technique */
}

ERRORS

None.

HISTORY

cgIsTechnique was introduced in Cg 1.4.

SEE ALSO

cgCreateTechnique, cgGetFirstTechnique, cgGetNamedTechnique, cgGetNextTechnique, cgGetTechniqueEffect, cgGetTechniqueName, cgIsTechniqueValidated, cgValidateTechnique

4.278 cgIsTechniqueValidated

NAME

cgIsTechniqueValidated - indicates whether the technique has passed validation
SYNOPSIS

#include <Cg/cg.h>

CGbool cgIsTechniqueValidated( CGtechnique tech );

PARAMETERS

  tech
    The technique handle.

RETURN VALUES

Returns CG_TRUE if the technique has previously passes validation via a call to cgValidateTechnique.
Returns CG_FALSE if validation hasn’t been attempted or the technique has failed a validation attempt.

DESCRIPTION

cgIsTechniqueValidated returns CG_TRUE if the technique has previously passes validation via a call to cgValidateTechnique. CG_FALSE is returned both if validation hasn’t been attempted as well as if the technique has failed a validation attempt.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgIsTechniqueValidated was introduced in Cg 1.4.

SEE ALSO

cgValidateTechnique, cgCallStateValidateCallback

4.279 cgMapBuffer

NAME

cgMapBuffer - map buffer into application’s address space

SYNOPSIS

#include <Cg/cg.h>

void * cgMapBuffer( CGbuffer buffer,
                     CGbufferaccess access );

PARAMETERS

  buffer
    The buffer which will be mapped into the application’s address space.
  access
An enumerant indicating the operations the client may perform on the data store through the pointer while the buffer data is mapped.

The following enumerants are allowed:

**CG_MAP_READ**
The application can read but not write through the data pointer.

**CG_MAP_WRITE**
The application can write but not read through the data pointer.

**CG_MAP_READ_WRITE**
The application can read and write through the data pointer.

**CG_MAP_WRITE_DISCARD**
Same as CG_MAP_READ_WRITE if using a GL buffer.

**CG_MAP_WRITE_NO_OVERWRITE**
Same as CG_MAP_READ_WRITE if using a GL buffer.

**RETURN VALUES**

Returns a pointer through which the application can read or write the buffer’s data store.

Returns NULL if an error occurs.

**DESCRIPTION**

cgMapBuffer maps a buffer into the application’s address space for memory-mapped updating of the buffer’s data. The application should call cgUnmapBuffer when it’s done updating or querying the buffer.

**EXAMPLES**

```c
unsigned char *bufferPtr = cgMapBuffer( myBuffer, CG_MAP_READ_WRITE );
memcpy( ptr, bufferPtr, size );
cgUnmapBuffer( myBuffer );
```

**ERRORS**

**CG_INVALID_BUFFER_HANDLE_ERROR** is generated if buffer is not a valid buffer.

**CG_INVALID_ENUMERANT_ERROR** is generated if access is not CG_MAP_READ, CG_MAP_WRITE, CG_MAP_READ_WRITE, CG_MAP_WRITE_DISCARD or CG_MAP_WRITE_NO_OVERWRITE.

**CG_BUFFER_ALREADY_MAPPED_ERROR** is generated if buffer is already mapped.

**HISTORY**

cgMapBuffer was introduced in Cg 2.0.

**SEE ALSO**

cgUnmapBuffer, cgSetBufferData, cgSetBufferSubData, cgSetParameter
4.280  cgResetPassState

NAME

cgResetPassState - calls the state resetting callback functions for all of the state assignments in a pass.

SYNOPSIS

#include <Cg/cg.h>

void cgResetPassState( CGpass pass );

PARAMETERS

  pass
      The pass handle.

RETURN VALUES

None.

DESCRIPTION

cgResetPassState resets all of the graphics state defined in a pass by calling the state resetting callbacks for all of the state assignments in the pass.

The semantics of “resetting state” will depend on the particular graphics state manager that defined the valid state assignments; it will generally either mean that graphics state is reset to what it was before the pass, or that it is reset to the default value. The OpenGL state manager in the OpenGL Cg runtime implements the latter approach.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

CG_INVALID_TECHNIQUE_ERROR is generated if the technique of which pass is a part has failed validation.

HISTORY

cgResetPassState was introduced in Cg 1.4.

SEE ALSO

cgSetPassState, cgCallStateResetCallback

4.281  cgSetArraySize

NAME

cgSetArraySize - sets the size of a resizable array parameter

SYNOPSIS
#include <Cg/cg.h>

void cgSetArraySize( CGparameter param,
                     int size );

PARAMETERS

param
   The array parameter handle.

size
   The new size of the array.

RETURN VALUES
None.

DESCRIPTION

cgSetArraySize sets the size of a resizable array parameter param to size.

EXAMPLES

If you have Cg program with a parameter like this:

/* ... */

float4 main(float4 myarray[])
{
   /* ... */
}

You can set the size of the myarray array parameter to 5 like so:

CGparameter arrayParam =
   cgGetNamedProgramParameter(program, CG_PROGRAM, "myarray");

cgSetArraySize(arrayParam, 5);

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_ARRAY_HAS_WRONG_DIMENSION_ERROR is generated if the dimension of the array parameter param is not 1.

CG_PARAMETER_IS_NOT_RESIZABLE_ARRAY_ERROR is generated if param is not a resizable array.

CG_INVALID_PARAMETER_ERROR is generated if size is less than 0.

HISTORY

cgSetArraySize was introduced in Cg 1.2.

SEE ALSO

cgGetArraySize, cgGetArrayDimension, cgSetMultiDimArraySize
4.282 cgSetAutoCompile

NAME

cgSetAutoCompile - sets the auto-compile mode for a context

SYNOPSIS

#include <Cg/cg.h>

void cgSetAutoCompile( CGcontext context,
                        CGenum autoCompileMode );

PARAMETERS

context

The context.

autoCompileMode

The auto-compile mode to which to set context. Must be one of the following:

* CG_COMPILE_MANUAL
* CG_COMPILE_IMMEDIATE
* CG_COMPILE_LAZY

RETURN VALUES

None.

DESCRIPTION

cgSetAutoCompile sets the auto compile mode for a given context. By default, programs are immediately recompiled when they enter an uncompiled state. This may happen for a variety of reasons including:

* Setting the value of a literal parameter.
* Resizing arrays.
* Binding structs to interface parameters.

autoCompileMode may be one of the following three enumerants:

* CG_COMPILE_IMMEDIATE

CG_COMPILE_IMMEDIATE will force recompilation automatically and immediately when a program enters an uncompiled state. This is the default mode.

* CG_COMPILE_MANUAL

With this method the application is responsible for manually recompiling a program. It may check to see if a program requires recompilation with the entry point cgIsProgramCompiled. cgCompileProgram can then be used to force compilation.

* CG_COMPILE_LAZY

This method is similar to CG_COMPILE_IMMEDIATE but will delay program recompilation until the program object code is needed. The advantage of this method is the reduction of extraneous recompilations. The disadvantage is that compile time errors will not be encountered when the program is enters the uncompiled state but will instead be encountered at some later time.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_ENUMERANT_ERROR is generated if autoCompileMode is not CG_COMPILE_MANUAL, CG_COMPILE_IMMEDIATE, or CG_COMPILE_LAZY.

HISTORY

cgSetAutoCompile was introduced in Cg 1.2.

SEE ALSO

cgGetAutoCompile, cgCompileProgram, cgIsProgramCompiled

4.283 cgSetBoolAnnotation

NAME
cgSetBoolAnnotation - set the value of a bool annotation

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetBoolAnnotation( CGannotation ann,
                        CGbool value );

PARAMETERS

    ann

    The annotation that will be set.

    value

    The value to which ann will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the annotation.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetBoolAnnotation sets the value of an annotation of bool type.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if \texttt{ann} is not a valid annotation.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if \texttt{ann} is not an annotation of bool type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if \texttt{ann} is not a scalar.

HISTORY

cgSetBoolAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetBoolAnnotationValues, cgGetIntAnnotation, cgSetFloatAnnotation, cgSetStringAnnotation

\section*{4.284 cgSetBoolArrayStateAssignment}

NAME

cgSetBoolArrayStateAssignment - set a bool-valued state assignment array

SYNOPSIS

\begin{verbatim}
#include <Cg/cg.h>

CGbool cgSetBoolArrayStateAssignment( CGstateassignment sa,
                        const CGbool * vals );
\end{verbatim}

PARAMETERS

\begin{itemize}
  \item \texttt{sa}
    A handle to a state assignment array of type \texttt{CG_BOOL}.
  \item \texttt{vals}
    The values which will be used to set \texttt{sa}.
\end{itemize}

RETURN VALUES

Returns \texttt{CG_TRUE} if it succeeds in setting the state assignment.

Returns \texttt{CG_FALSE} otherwise.

DESCRIPTION

cgSetBoolArrayStateAssignment sets the value of a state assignment of bool array type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if \texttt{sa} is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if \texttt{sa} is not a state assignment of a bool type.
HISTORY

cgSetBoolArrayStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetBoolStateAssignmentValues, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.285 cgSetBoolStateAssignment

NAME

cgSetBoolStateAssignment - set the value of a bool state assignment

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetBoolStateAssignment( CGstateassignment sa, 
                                    CGbool value );

PARAMETERS

    sa
    A handle to a state assignment of type CG_BOOL.

    value
    The value to which sa will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetBoolStateAssignment sets the value of a state assignment of bool type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a bool type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.

HISTORY

cgSetBoolStateAssignment was introduced in Cg 1.5.
4.286 cgSetBufferData

NAME
cgSetBufferData - resize and completely update a buffer object

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetBufferData( CGbuffer buffer,
                      int size,
                      const void * data );
```

PARAMETERS

buffer
  The buffer which will be updated.
size
  Specifies a new size for the buffer object. Zero for size means use the existing size of the buffer as the effective size.
data
  Pointer to the data to copy into the buffer. The number of bytes to copy is determined by the size parameter.

RETURN VALUES

None.

DESCRIPTION

cgSetBufferData resizes and completely updates an existing buffer object.

A buffer which has been mapped into an applications address space with cgMapBuffer must be unmapped using cgUnmapBuffer before it can be updated with cgSetBufferData.

EXAMPLES

cgSetBufferData( myBuffer, sizeof( myData ), myData );

ERRORS

CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

CG_BUFFER_UPDATE_NOT_ALLOWED_ERROR is generated if buffer is currently mapped.

HISTORY

cgSetBufferData was introduced in Cg 2.0.
SEE ALSO

`cgCreateBuffer`, `cgGLCreateBuffer`, `cgSetBufferSubData`, `cgMapBuffer`, `cgUnmapBuffer`

### 4.287 cgSetBufferSubData

**NAME**

cgSetBufferSubData - partially update a Cg buffer object

**SYNOPSIS**

```c
#include <Cg/cg.h>

void cgSetBufferSubData( CGbuffer buffer, 
                         int offset, 
                         int size, 
                         const void * data );
```

**PARAMETERS**

- buffer
  - Buffer being updated.
- offset
  - Buffer offset in bytes of the beginning of the partial update.
- size
  - Number of buffer bytes to be updated. Zero means no update.
- data
  - Pointer to the start of the data being copied into the buffer.

**RETURN VALUES**

None.

**DESCRIPTION**

cgSetBufferSubData resizes and partially updates an existing buffer object.

A buffer which has been mapped into an applications address space with `cgMapBuffer` must be unmapped using `cgUnmapBuffer` before it can be updated with `cgSetBufferSubData`.

**EXAMPLES**

cgSetBufferSubData( myBuffer, 16, sizeof( myData ), myData );

**ERRORS**

- `CG_INVALID_BUFFER_HANDLE_ERROR` is generated if `buffer` is not a valid buffer.
- `CG_BUFFER_UPDATE_NOT_ALLOWED_ERROR` is generated if `buffer` is currently mapped.
- `CG_BUFFER_INDEX_OUT_OF_RANGE_ERROR` is generated if `offset` or `size` is out of range.

**HISTORY**

cgSetBufferSubData was introduced in Cg 2.0.
SEE ALSO

cgCreateBuffer, cgGLCreateBuffer, cgSetBufferData, cgMapBuffer, cgUnmapBuffer

4.288 cgSetCompilerIncludeCallback

NAME
cgSetCompilerIncludeCallback - set the include callback function

SYNOPSIS

#include <Cg/cg.h>

typedef void (*CGIncludeCallbackFunc)( CGcontext context, const char *filename );

void cgSetCompilerIncludeCallback( CGcontext context, CGIncludeCallbackFunc func );

PARAMETERS

context
The context for which the include callback will be used.

func
A pointer to the include callback function.

RETURN VALUES

None.

DESCRIPTION

cgSetCompilerIncludeCallback sets a callback function used for handing include statements. Each Cg runtime context maintains a virtual file system of shader source code for inclusion by the compiler. Source code is populated into the virtual filesystem using cgSetCompilerIncludeString and cgSetCompilerIncludeFile. When the compiler encounters an include, firstly the virtual file system is searched for a match. Secondly the include callback function is called, providing an opportunity for populating shader source via cgSetCompilerIncludeString and cgSetCompilerIncludeFile. The callback function is passed the context and the requested name. Thirdly, the filesystem is searched in the usual manner. Fourthly, an error is raised by the compiler that the include can not be satisfied. NULL is passed to cgSetCompilerIncludeCallback to disable the callback.

EXAMPLES

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgSetCompilerIncludeCallback was introduced in Cg 2.1.

SEE ALSO

cgGetCompilerIncludeCallback, cgSetCompilerIncludeString, cgSetCompilerIncludeFile
4.289  cgSetCompilerIncludeFile

NAME

cgSetCompilerIncludeFile - add shader source file to context

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetCompilerIncludeFile( CGcontext context, const char *name, const char *filename );
```

PARAMETERS

- context
  - The context in which to add the source code for inclusion by the compiler.
- name
  - The virtual file system name of the shader source.
- filename
  - System file system name of shader source file.

RETURN VALUES

None.

DESCRIPTION

Each Cg runtime context maintains a virtual filesystem of shader source code for inclusion by the compiler. `cgSetCompilerIncludeFile` populates source code into the virtual filesystem from a file. A name is removed from the virtual filesystem by using `NULL` for the `filename`. The virtual filesystem is completely cleared by using `NULL` for the `name`.

EXAMPLES

ERRORS

- `CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.
- `CG_FILE_READ_ERROR` is generated if the file `filename` can not be opened for input.

HISTORY

cgSetCompilerIncludeFile was introduced in Cg 2.1.

SEE ALSO

- `cgSetCompilerIncludeString`, `cgGetCompilerIncludeCallback`, `cgSetCompilerIncludeCallback`

4.290  cgSetCompilerIncludeString

NAME

`cgSetCompilerIncludeString` - add shader source string to context
SYNOPSIS

#include <Cg/cg.h>

void cgSetCompilerIncludeString( CGcontext context, const char *name, const char *source );

PARAMETERS

context
The context in which to add the source code for inclusion by the compiler.

name
The virtual file system name of the shader source.

source
Shader source code string.

RETURN VALUES

None.

DESCRIPTION

Each Cg runtime context maintains a virtual file system of shader source code for inclusion by the compiler. cgSetCompilerIncludeString populates source code into the virtual filesystem. A name is removed from the virtual filesystem by using NULL for the source. The virtual filesystem is completely cleared by using NULL for the name.

EXAMPLES

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgSetCompilerIncludeString was introduced in Cg 2.1.

SEE ALSO

cgSetCompilerIncludeFile, cgGetCompilerIncludeCallback, cgSetCompilerIncludeCallback

4.291 cgSetContextBehavior

NAME

cgSetContextBehavior - set the behavior for a context

SYNOPSIS

#include <Cg/cg.h>

void cgSetContextBehavior( CGcontext context, 
                          CGbehavior behavior );
PARAMETERS

context
The context for which the behavior will be set.

behavior
An enumerant which defines the behavior that will be exhibited by context. The following enumerants are allowed:

* CG_BEHAVIOR_3100
* CG_BEHAVIOR_3000
* CG_BEHAVIOR_2200
* CG_BEHAVIOR_CURRENT
* CG_BEHAVIOR_LATEST

RETURN VALUES
None.

DESCRIPTION
Each new version of Cg is supposed to be completely backwards compatible with previous versions, providing bug fixes and/or new capabilities while maintaining the behavior which applications were written against. The intent is to allow Cg to be updated and have existing applications continue to work exactly as designed.

Occasionally a case is made that some behavior of Cg is wrong, but fixing that behavior could break existing applications. This is a problem. cgSetContextBehavior provides a solution by documenting such changes to the library’s behavior and allowing applications to explicitly opt-in to the new behavior. For applications which don’t use cgSetContextBehavior Cg will continue to behave exactly as it did before this routine was introduced.

It is expected that the definition of a new context behavior will be a rare occurrence and not something that happens with every new Cg release. Routine bug fixes and additions to the API won’t result in creating new values of behavior. Instead this will only be done when the fix for a broken library behavior could cause a correctly written application to fail.

behavior must be one of the following enumerants:

* CG_BEHAVIOR_3100
Cg 3.1 added a new method to define constant blocks using the uniform keyword. Using CG_BEHAVIOR_3100 enables recognition of this new construct and deprecates use of the BUFFER semantic on a struct to define a constant block. With CG_BEHAVIOR_3100 use of a buffer semantic on a struct will result in an error for GLSL profiles and a warning for all other profiles. See the Constant Buffers section of the Language Specification for details on defining constant buffers using the uniform keyword.

* CG_BEHAVIOR_3000
Cg 3.0 added support for 16 additional texture units for a total of 32 using the semantics TEXUNIT16 through TEXUNIT31 and resource enums CG_TEXUNIT16 through CG_TEXUNIT31. To use these new resources, CG_BEHAVIOR_3000 is required. Using CG_BEHAVIOR_2200 with these new texture unit resources will result in a Cg error.

* CG_BEHAVIOR_2200
This value specifies a pattern of behavior matching what was seen from Cg through release 2.2. Applications which specify CG_BEHAVIOR_2200 do not need to be modified to handle new context behaviors since they will continue to get the oldest behavior from the library.
Note that this selection is the default behavior for applications which never call \texttt{cgSetContextBehavior}, which means that existing binaries will continue to get the behavior they expect from Cg. This is also the fallback behavior if an invalid value of \texttt{behavior} is passed to \texttt{cgSetContextBehavior}.

* \texttt{CG\_BEHAVIOR\_CURRENT}

When this value is used the most advanced context behavior supported by the library will be determined at compile time and will become part of the application binary. Updating the Cg runtime files will not change the behavior seen by the application at runtime. However, if the updated version of Cg defines a new value for \texttt{behavior} then this new behavior will be used after the application is recompiled.

* \texttt{CG\_BEHAVIOR\_LATEST}

When this value is used the most advanced context behavior supported by the library will be determined at application runtime. Updating the Cg runtime files may result in new behavior from Cg even though the same application binaries are being used.

* \texttt{CG\_BEHAVIOR\_UNKNOWN}

This value is returned by \texttt{cgGetBehaviorString} to indicate an invalid string argument has been used. Passing \texttt{CG\_BEHAVIOR\_UNKNOWN} to \texttt{cgSetContextBehavior} will generate an \texttt{CG\_INVALID\_ENUMERANT\_ERROR} and result in the context’s behavior being set to \texttt{CG\_BEHAVIOR\_2200} instead.

If the environment variable \texttt{CG\_BEHAVIOR} is set to any of the valid \texttt{CG\_behavior} enumerant names, then that context behavior will be used instead of the behavior compiled into the application binary. This is true even when the application doesn’t explicitly call \texttt{cgSetContextBehavior}. Note that \texttt{CG\_BEHAVIOR\_CURRENT} and \texttt{CG\_BEHAVIOR\_UNKNOWN} are not valid choices for \texttt{CG\_BEHAVIOR}. Trying to use either will result in an error.

**EXAMPLES**

/* create a context and set the behavior to \texttt{CG\_BEHAVIOR\_3100} */

\begin{verbatim}
CGcontext context = cgCreateContext();
cgSetContextBehavior(context, CG\_BEHAVIOR\_3100);
\end{verbatim}

**ERRORS**

\texttt{CG\_INVALID\_CONTEXT\_HANDLE\_ERROR} is generated if \texttt{context} is not a valid context.

\texttt{CG\_INVALID\_ENUMERANT\_ERROR} is generated if \texttt{behavior} is not \texttt{CG\_BEHAVIOR\_3100}, \texttt{CG\_BEHAVIOR\_3000}, \texttt{CG\_BEHAVIOR\_2200}, \texttt{CG\_BEHAVIOR\_CURRENT}, or \texttt{CG\_BEHAVIOR\_LATEST}.

**HISTORY**

\texttt{cgSetContextBehavior} was introduced in Cg 3.0.

**SEE ALSO**

\texttt{cgCreateContext}, \texttt{cgGetContextBehavior}, \texttt{cgGetBehavior}, \texttt{cgGetBehaviorString}

4.292 \texttt{cgSetEffectName}

**NAME**

\texttt{cgSetEffectName} - set the name of an effect
SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetEffectName( CGeffect effect,
                                   const char * name );

PARAMETERS

effect
    The effect in which the name will be set.

name
    The new name for effect.

RETURN VALUES

Returns CG_TRUE if it succeeds.
Returns CG_FALSE otherwise.

DESCRIPTION

cgSetEffectName allows the application to set the name of an effect.

EXAMPLES

char *effectSource = ...;
CGcontext context = cgCreateContext();
CGeffect effect = cgCreateEffect(context, effectSource, NULL);

const char * myEffectName = "myEffectName";
CGbool okay = cgSetEffectName(effect, myEffectName);
if (!okay) {
    /* handle error */
}

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if effect is not a valid effect.

HISTORY

cgSetEffectName was introduced in Cg 1.5.

SEE ALSO

cgGetEffectName, cgCreateEffect

4.293 cgSetEffectParameterBuffer

NAME

cgSetEffectParameterBuffer - sets a Cg buffer to every program in the effect that uses the passed effect buffer parameter.
SYNOPSIS

#include <Cg/cg.h>

void cgSetEffectParameterBuffer(CGparameter param,
                                  CGbuffer buffer);

PARAMETERS

  param
  The effect parameter used by programs in the effect as a buffer parameter.

  buffer
  The Cg buffer being set to param for each program in the effect that uses param.

RETURN VALUES

None.

DESCRIPTION

cgSetEffectParameterBuffer allows the application with a single API call to set a Cg buffer to an effect parameter using the BUFFER semantic for each program in the effect that uses this effect parameter.

cgSetEffectParameterBuffer does the equivalent of the following:

CGtechnique technique = cgGetFirstTechnique(myCgEffect);
for(; technique; technique = cgGetNextTechnique(technique))
{
  if(!cgIsTechniqueValidated(technique))
    continue;

  CGpass pass = cgGetFirstPass(technique);
  for( ; pass; pass = cgGetNextPass(pass))
  {
    for(int i = 0; i < numDomains; ++i)
    {
      CGprogram prog = cgGetPassProgram(pass, domains[i]);
      if(!prog)
        continue;

      CGparameter param = cgGetNamedParameter(prog, "paramName");
      CGbool isUsed = cgIsParameterUsed(param, prog);
      if(isUsed == CG_FALSE)
        continue;

      int idx = cgGetParameterBufferIndex(param);
      if(idx < 0)
        continue;

      cgSetProgramBuffer(prog, idx, myCgBuffer);
    }
  }
}

EXAMPLES
cgSetEffectParameterBuffer(cgGetNamedEffectParameter(myCgEffect, "paramName"), myCgBuffer);

See examples/OpenGL/advanced/cgfx_buffer_lighting.

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgSetEffectParameterBuffer was introduced in Cg 3.0.

SEE ALSO

cgGetEffectParameterBuffer, cgSetProgramBuffer

4.294 cgSetErrorCallback

NAME

cgSetErrorCallback - set the error callback function

SYNOPSIS

#include <Cg/cg.h>

typedef void (*CGerrorCallbackFunc)( void );

void cgSetErrorCallback( CGerrorCallbackFunc func );

PARAMETERS

func

A function pointer to the error callback function.

RETURN VALUES

None.

DESCRIPTION

cgSetErrorCallback sets a callback function that will be called every time an error occurs. The callback function is not passed any parameters. It is assumed that the callback function will call cgGetError to obtain the current error. To disable the callback function, cgSetErrorCallback may be called with NULL.

EXAMPLES

The following is an example of how to set and use an error callback:

void MyErrorCallback( void ) {
    int myError = cgGetError();
    fprintf(stderr, "CG ERROR : %s\n", cgGetErrorString(myError));
}

void main(int argc, char *argv[]) {
    cgSetErrorCallback(MyErrorCallback);
}
/* Do stuff */

ERRORS
None.

HISTORY
cgSetErrorCallback was introduced in Cg 1.1.

SEE ALSO
cgGetErrorCallback, cgGetError, cgGetErrorString

4.295 cgSetErrorHandler

NAME
cgSetErrorHandler - set the error handler callback function

SYNOPSIS
#include <Cg/cg.h>

typedef void (*CGerrorHandlerFunc)( CGcontext context, CGerror error, void * appdata);

void cgSetErrorHandler( CGerrorHandlerFunc func, void * appdata);

PARAMETERS
func
A pointer to the error handler callback function.
appdata
A pointer to arbitrary application-provided data.

RETURN VALUES
None.

DESCRIPTION
cgSetErrorHandler specifies an error handler function that will be called every time a Cg runtime error occurs. The callback function is passed:

context
The context in which the error occurred. If the context cannot be determined, NULL is used.
error
The enumerant of the error triggering the callback.
appdata
The value of the pointer passed to \texttt{cgSetErrorHandler}. This pointer can be used to make arbitrary application-side information available to the error handler.

To disable the callback function, specify a \texttt{NULL} callback function pointer via \texttt{cgSetErrorHandler}.

\textbf{EXAMPLES}

\begin{verbatim}
void MyErrorHandler(CGcontext context, CGerror error, void *data) {
    char *progname = (char *)data;
    fprintf(stderr, "%s: Error: %s\n", progname, cgGetErrorString(error));
}

void main(int argc, char *argv[]) {
    ...
    cgSetErrorHandler(MyErrorHandler, (void *)argv[0]);
    ...
}
\end{verbatim}

\textbf{ERRORS}

\textit{to-be-written}

\textbf{HISTORY}

\texttt{cgSetErrorHandler} was introduced in Cg 1.4.

\textbf{SEE ALSO}

\texttt{cgGetErrorHandler}, \texttt{cgGetError}, \texttt{cgGetErrorString}, \texttt{cgGetFirstError}

\section*{4.296 \texttt{cgSetFloatAnnotation}}

\textbf{NAME}

\texttt{cgSetFloatAnnotation} - set the value of a float annotation

\textbf{SYNOPSIS}

\begin{verbatim}
#include <Cg/cg.h>

CGbool cgSetFloatAnnotation( CGannotation ann,
                                    float value );
\end{verbatim}

\textbf{PARAMETERS}

\begin{itemize}
    \item \texttt{ann}
        The annotation that will be set.
    \item \texttt{value}
        The value to which \texttt{ann} will be set.
\end{itemize}

\textbf{RETURN VALUES}

Returns \texttt{CG_TRUE} if it succeeds in setting the annotation.

Returns \texttt{CG_FALSE} otherwise.
DESCRIPTION

cgSetFloatAnnotation sets the value of an annotation of float type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if `ann` is not a valid annotation.
CG_INVALID_PARAMETER_TYPE_ERROR is generated if `ann` is not an annotation of float type.
CG_ARRAY_SIZE_MISMATCH_ERROR is generated if `ann` is not a scalar.

HISTORY

cgSetFloatAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetFloatAnnotationValues, cgSetBoolAnnotation, cgSetIntAnnotation, cgSetStringAnnotation

4.297 cgSetFloatArrayStateAssignment

NAME

cgSetFloatArrayStateAssignment - set a float-valued state assignment array

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetFloatArrayStateAssignment( CGstateassignment sa,
                                      const float * vals );

PARAMETERS

   sa
       A handle to a state assignment array of type CG_FLOAT, CG_FIXED, CG_HALF.
   vals
       The values which will be used to set `sa`.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.
Returns CG_FALSE otherwise.

DESCRIPTION

cgSetFloatArrayStateAssignment sets the value of a state assignment of float array type.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a float type.

HISTORY

cgSetFloatArrayStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetFloatStateAssignmentValues, cgSetFloatStateAssignment, cgSetBoolArrayStateAssignment, cgSetBooleanStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.298 cgSetFloatStateAssignment

NAME

cgSetFloatStateAssignment - set the value of a state assignment

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetFloatStateAssignment( CGstateassignment sa,
    float value );

PARAMETERS

    sa
    A handle to a state assignment of type CG_FLOAT, CG_FIXED, or CG_HALF.

    value
    The value to which sa will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetFloatStateAssignment sets the value of a state assignment of float type.

EXAMPLES

to-be-written

ERRORS

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a float type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.
HISTORY

cgSetFloatStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetFloatStateAssignmentValues, cgSetFloatArrayStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.299 cgSetIntAnnotation

NAME
cgSetIntAnnotation - set the value of an int annotation

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetIntAnnotation( CGannotation ann, int value );

PARAMETERS

ann

The annotation that will be set.

value

The value to which ann will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the annotation.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetIntAnnotation sets the value of an annotation of int type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if ann is not a valid annotation.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if ann is not an annotation of int type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if ann is not a scalar.

HISTORY

cgSetIntAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetIntAnnotationValues, cgSetBoolAnnotation, cgSetFloatAnnotation, cgSetStringAnnotation
4.300  cgSetIntArrayStateAssignment

NAME

cgSetIntArrayStateAssignment - set an int-valued state assignment array

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetIntArrayStateAssignment( CGstateassignment sa,
                                      const int * vals );

PARAMETERS

sa
   A handle to a state assignment array of type CG_INT.

vals
   The values which will be used to set sa.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.
Returns CG_FALSE otherwise.

DESCRIPTION

cgSetIntArrayStateAssignment sets the value of a state assignment of int array type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of an int type.

HISTORY

cgSetIntArrayStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetIntStateAssignmentValues, cgGetIntStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.301  cgSetIntStateAssignment

NAME

cgSetIntStateAssignment - set the value of an int state assignment

PARAMETERS

sa
   A handle to a state assignment of type CG_INT.

vals
   The values which will be used to set sa.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.
Returns CG_FALSE otherwise.

DESCRIPTION

cgSetIntStateAssignment sets the value of an int state assignment.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of an int type.

HISTORY

cgSetIntStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetIntStateAssignmentValues, cgGetIntStateAssignment, cgSetIntArrayStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment
SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetIntStateAssignment( CGstateassignment sa,
                                        int value );

PARAMETERS

  sa
    A handle to a state assignment of type CG_INT.
  value
    The value to which sa will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetIntStateAssignment sets the value of a state assignment of int type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of an int type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.

HISTORY

cgSetIntStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetIntStateAssignmentValues, cgSetIntArrayStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.302 cgSetLastListing

NAME

cgSetLastListing - set the current listing text

SYNOPSIS
#include <Cg/cg.h>

void cgSetLastListing( CGhandle handle,
                        const char * listing );

PARAMETERS

    handle
    A CGcontext, CGstateassignment, CGeffect, CGpass, or CGtechnique belonging to the context whose
    listing text is to be set.

    listing
    The new listing text.

RETURN VALUES

None.

DESCRIPTION

Each Cg context maintains a null-terminated string containing warning and error messages generated by the Cg com-
piler, state managers and the like. cgSetLastListing allows applications and custom state managers to set the listing
text.

cgSetLastListing is not normally used directly by applications. Instead, custom state managers can use cgSet-
LastListing to provide detailed technique validation error messages to the application.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if handle is invalid.

HISTORY

cgSetLastListing was introduced in Cg 1.4.

SEE ALSO

cgGetLastListing, cgCreateContext, cgSetErrorHandler

4.303 cgSetLockingPolicy

NAME

cgSetLockingPolicy - set locking policy

SYNOPSIS

#include <Cg/cg.h>

CGenum cgSetLockingPolicy( CGenum lockingPolicy );
PARAMETERS

lockingPolicy
An enumerant describing the desired locking policy for the library. The following enumerants are allowed:

CG_THREAD_SAFE_POLICY
Locks will be used to serialize thread access to the library.
CG_NO_LOCKS_POLICY
Locks will not be used.

RETURN VALUES

Returns the previous locking policy, or CG_UNKNOWN if an error occurs.

DESCRIPTION

.cgSetLockingPolicy allows an application to change the locking policy used by the Cg library. The default policy is CG_THREAD_SAFE_POLICY, meaning a lock is used to serialize access to the library by multiple threads. Single threaded applications can change this policy to CG_NO_LOCKS_POLICY to avoid the overhead associated with this lock. Multithreaded applications should never change this policy.

EXAMPLES

/* multithreaded apps should *never* do this */
cgSetLockingPolicy(CG_NO_LOCKS_POLICY);

ERRORS

CG_INVALID_ENUMERANT_ERROR is generated if lockingPolicy is not CG_NO_LOCKS_POLICY or CG_THREAD_SAFE_POLICY.

HISTORY

cgSetLockingPolicy was introduced in Cg 2.0.

SEE ALSO

cgGetLockingPolicy

4.304 cgSetMatrixParameter

NAME

cgSetMatrixParameter - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

/* TYPE is int, float or double */

void cgSetMatrixParameter{ifd}{rc}( CGparameter param,
           const TYPE * matrix );

4.304. cgSetMatrixParameter 423
PARAMETERS

  param
  The parameter that will be set.

  matrix
  An array of values to which to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES

None.

DESCRIPTION

The cgSetMatrixParameter functions set the value of a given matrix parameter. The functions are available in various combinations.

There are versions of each function that take int, float or double values signified by the i, f or d in the function name.

There are versions of each function that assume the array of values are laid out in either row or column order signified by the r or c in the function name respectively.

The cgSetMatrixParameter functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

The d and f versions of cgSetMatrixParameter were introduced in Cg 1.2.

The i versions of cgSetMatrixParameter were introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.305 cgSetMatrixParameterdc

NAME

cgSetMatrixParameterdc - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameterdc( CGparameter param,
                                 const double * matrix );
PARAMETERS

    param
    The parameter that will be set.

    matrix
    An array of values used to set the matrix parameter. The array must be the number of rows times the
    number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgSetMatrixParameterdc sets the value of a given matrix parameter from an array of doubles laid out in column-
major order.
cgSetMatrixParameterdc may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgSetMatrixParameterdc was introduced in Cg 1.2.

SEE ALSO

cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameter-
Values

4.306 cgSetMatrixParameterdr

NAME

cgSetMatrixParameterdr - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameterdr( CGparameter param,
                             const double * matrix );
PARAMETERS

param
The parameter that will be set.

matrix
An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgSetMatrixParameterdr sets the value of a given matrix parameter from an array of doubles laid out in row-major order.
cgSetMatrixParameterdr may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgSetMatrixParameterdr was introduced in Cg 1.2.

SEE ALSO

cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.307 cgSetMatrixParameterfc

NAME

cgSetMatrixParameterfc - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameterfc( CGparameter param,
                          const float * matrix );
PARAMETERS

    param
    The parameter that will be set.

    matrix
    An array of values used to set the matrix parameter. The array must be the number of rows times the
    number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgSetMatrixParameterfc sets the value of a given matrix parameter from an array of floats laid out in column-major
order.

cgSetMatrixParameterfc may only be called with uniform parameters.

EXAMPLES

    to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgSetMatrixParameterfc was introduced in Cg 1.2.

SEE ALSO

cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameter-
Values

4.308 cgSetMatrixParameterfr

NAME

cgSetMatrixParameterfr - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameterfr( CGparameter param,
                            const float * matrix );
PARAMETERS

param
The parameter that will be set.

matrix
An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgSetMatrixParameterfr sets the value of a given matrix parameter from an array of floats laid out in row-major order.
cgSetMatrixParameterfr may only be called with uniform parameters.

EXAMPLES
to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgSetMatrixParameterfr was introduced in Cg 1.2.

SEE ALSO

cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValue

4.309 cgSetMatrixParameteric

NAME

cgSetMatrixParameteric - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameteric( CGparameter param,
                         const int * matrix );
PARAMETERS

param
The parameter that will be set.

matrix
An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES
None.

DESCRIPTION

cgSetMatrixParameteric sets the value of a given matrix parameter from an array of ints laid out in column-major order.
cgSetMatrixParameteric may only be called with uniform parameters.

EXAMPLES
to-be-written

ERRORS

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgSetMatrixParameteric was introduced in Cg 1.4.

SEE ALSO

.cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameter-Values

4.310 cgSetMatrixParameterir

NAME

cgSetMatrixParameterir - sets the value of matrix parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetMatrixParameterir( CGparameter param,
                          const int * matrix );
PARAMETERS

param
The parameter that will be set.

matrix
An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES
None.

DESCRIPTION
cgSetMatrixParameterir sets the value of a given matrix parameter from an array of ints laid out in row-major order.
cgSetMatrixParameterir may only be called with uniform parameters.

EXAMPLES
to-be-written

ERRORS
CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgSetMatrixParameterir was introduced in Cg 1.4.

SEE ALSO
cgSetMatrixParameter, cgGetParameterRows, cgGetParameterColumns, cgGetMatrixParameter, cgGetParameterValues

4.311 cgSetMultiDimArraySize

NAME
cgSetMultiDimArraySize - sets the size of a resizable multi-dimensional array parameter

SYNOPSIS

#include <Cg/cg.h>

void cgSetMultiDimArraySize( CGparameter param,
                              const int * sizes );

PARAMETERS

param
The array parameter handle.

sizes
An array of sizes for each dimension of the array.

RETURN VALUES

None.

DESCRIPTION

cgSetMultiDimArraySize sets the size of each dimension of resizable multi-dimensional array parameter param. sizes must be an array that has N number of elements where N is equal to the result of cgGetArrayDimension.

EXAMPLES

If you have Cg program with a parameter like this:

```c
/* ... */
float4 main(float4 myarray[][])[])
{
    /* ... */
}
```

You can set the sizes of each dimension of the myarray array parameter like so:

```c
const int sizes[] = { 3, 2, 4 };;
CGparameter myArrayParam =
    cgGetNamedProgramParameter(program, CG_PROGRAM, "myarray");

cgSetMultiDimArraySize(myArrayParam, sizes);
```

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_INVALID_POINTER_ERROR is generated if sizes is NULL.
CG_INVALID_PARAMETER_ERROR is generated if any value in sizes is less than or equal to 0.
CG_PARAMETER_IS_NOT_RESIZABLE_ARRAY_ERROR is generated if param is not a resizable array.

HISTORY

cgSetMultiDimArraySize was introduced in Cg 1.2.

SEE ALSO

cgGetArraySize, cgGetArrayDimension, cgSetArraySize

4.312 cgSetParameter

NAME

cgSetParameter - sets the value of scalar and vector parameters

SYNOPSIS
#include <Cg/cg.h>

/* TYPE is int, float or double */

void cgSetParameter1{ifd}( CGparameter param,
    TYPE x );

void cgSetParameter2{ifd}( CGparameter param,
    TYPE x,
    TYPE y );

void cgSetParameter3{ifd}( CGparameter param,
    TYPE x,
    TYPE y,
    TYPE z );

void cgSetParameter4{ifd}( CGparameter param,
    TYPE x,
    TYPE y,
    TYPE z,
    TYPE w );

void cgSetParameter{1234}{ifd}v( CGparameter param,
    const TYPE * v );

PARAMETERS

    param
    The parameter that will be set.
    x, y, z, and w
    The values to which to set the parameter.
    v
    The values to set the parameter to for the array versions of the set functions.

RETURN VALUES

None.

DESCRIPTION

The cgSetParameter functions set the value of a given scalar or vector parameter. The functions are available in various combinations.

Each function takes either 1, 2, 3, or 4 values depending on the function that is used. If more values are passed in than the parameter requires, the extra values will be ignored.

There are versions of each function that take int, float or double values signified by the i, f or d in the function name. The functions with the v at the end of their names take an array of values instead of explicit parameters.

Once cgSetParameter has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

Note: Previous releases of Cg allowed you to store more values in a parameter than indicated by the parameter’s type. For example, one could use cgGLSetParameter4f to store four values into a parameter of type CG_FLOAT (not
CG_FLOAT4). All four values could later be retrieved using a get call which requested more than one value. However, this feature conflicts with the GLSL approach and also leads to issues with parameters mapped into BUFFERS. Therefore, beginning with Cg 2.0 any components beyond the number indicated by the parameter type are ignored.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

The d and f versions of cgSetParameter were introduced in Cg 1.2.

The i versions of cgSetParameter were introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue

4.313 cgSetParameter1d

NAME

cgSetParameter1d - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter1d( Cgparameter param,
                   double x );

PARAMETERS

param
    The parameter that will be set.

x
    The value to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter1d sets the value of a given scalar or vector parameter.

Once cgSetParameter1d has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.
EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter1d was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.314 cgSetParameter1dv

NAME
cgSetParameter1dv - sets the value of scalar and vector parameters

SYNOPSIS
#include <Cg/cg.h>

void cgSetParameter1dv( CGparameter param,
                        const double * v );

PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION
cgSetParameter1dv sets the value of a given scalar or vector parameter.

Once cgSetParameter1dv has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES
to-be-written
ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter1dv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.315 cgSetParameter1f

NAME

cgSetParameter1f - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter1f( CGparameter param, 
                      float x );

PARAMETERS

param

The parameter that will be set.

x

The value to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter1f sets the value of a given scalar or vector parameter.

Once cgSetParameter1f has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.
HISTORY

cgSetParameter1f was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue, cgGetParameterValues

4.316 cgSetParameter1fv

NAME

cgSetParameter1fv - sets the value of scalar and vector parameters

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetParameter1fv( CGparameter param,
                         const float * v);
```

PARAMETERS

- `param`
  - The parameter that will be set.
- `v`
  - Array of values to use to set `param`.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter1fv sets the value of a given scalar or vector parameter.

Once `cgSetParameter1fv` has been used to set a parameter, the values may be retrieved from the parameter using the `CG_CURRENT` enumerant with `cgGetParameterValues`.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

- `CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.
- `CG_INVALID_PARAMETER_ERROR` is generated if `param` is a varying input to a fragment program.

HISTORY

cgSetParameter1fv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue
4.317 cgSetParameter1i

NAME
cgSetParameter1i - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter1i( CGparameter param,
                        int x );

PARAMETERS

param
The parameter that will be set.

x
The value to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter1i sets the value of a given scalar or vector parameter.

Once cgSetParameter1i has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter1i was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgGetParameterValues

4.318 cgSetParameter1iv

NAME

cgSetParameter1iv - sets the value of scalar and vector parameters
SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter1iv( CGparameter param,
                       const int * v );

PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter1iv sets the value of a given scalar or vector parameter.

Once cgSetParameter1iv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter1iv was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue

4.319 cgSetParameter2d

NAME

cgSetParameter2d - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter2d( CGparameter param,
PARAMETERS

param

The parameter that will be set.

x, y

The values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter2d sets the value of a given scalar or vector parameter.

If more values are passed in than param requires, the extra values will be ignored.

Once cgSetParameter2d has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter2d was introduced in Cg 1.2.

SEE ALSO

cGetParameterValue, cgGetParameterValues

4.320 cgSetParameter2dv

NAME

cgSetParameter2dv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter2dv( CGparameter param,
                      const double * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter2dv sets the value of a given scalar or vector parameter.

If more values are passed in than param requires, the extra values will be ignored.

Once cgSetParameter2dv has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerator with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter2dv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.321 cgSetParameter2f

NAME

cgSetParameter2f - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter2f( CGparameter param,
                 float x, 
                 float y );
PARAMETERS

    param
    The parameter that will be set.
    x, y
    The values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter2f sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter2f has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter2f was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue, cgGetParameterValues

4.322 cgSetParameter2fv

NAME

cgSetParameter2fv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter2fv( CGparameter param,
                        const float * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter2fv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter2fv has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter2fv was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue

4.323  cgSetParameter2i

NAME
cgSetParameter2i - set the value of scalar and vector parameters

SYNOPSIS
#include <Cg/cg.h>

void cgSetParameter2i( CGparameter param,
                        int x,
                        int y );
PARAMETERS

param
The parameter that will be set.
x, y
The values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter2i sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter2i has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter2i was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgGetParameterValues

4.324 cgSetParameter2iv

NAME

cgSetParameter2iv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter2iv( CGparameter param,
                        const int * v );
PARAMETERS

    param

    The parameter that will be set.

    v

    Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter2iv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter2iv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter2iv was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue

4.325 cgSetParameter3d

NAME

cgSetParameter3d - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3d( CGparameter param,
    double x,
    double y,
    double z );
PARAMETERS

param
The parameter that will be set.

x, y, z
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter3d sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter3d has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter3d was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.326 cgSetParameter3dv

NAME
cgSetParameter3dv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3dv( CGparameter param,
                        const double * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter3dv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter3dv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter3dv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.327 cgSetParameter3f

NAME

cgSetParameter3f - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3f( CGparameter param,
    float x,
    float y,
    float z );
PARAMETERS

param
The parameter that will be set.
	x, y, z
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter3f sets the value of a given scalar or vector parameter.

If more values are passed in than param requires, the extra values will be ignored.

Once cgSetParameter3f has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.

If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter3f was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.328 cgSetParameter3fv

NAME
cgSetParameter3fv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3fv( CGparameter param,
                       const float * v );
PARAMETERS

   param
   The parameter that will be set.

   v
   Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter3fv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter3fv has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter3fv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.329 cgSetParameter3i

NAME

cgSetParameter3i - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3i( CGparameter param,
            int x,
            int y,
            int z );
PARAMETERS

param
The parameter that will be set.
x, y, z
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter3i sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter3i has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter3i was introduced in Cg 1.4.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.330 cgSetParameter3iv

NAME
cgSetParameter3iv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter3iv( CGparameter param,
                        const int * v );
PARAMETERS

param
    The parameter that will be set.
    v
    Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter3iv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter3iv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter3iv was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue

4.331  cgSetParameter4d

NAME

cgSetParameter4d - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter4d( CGparameter param,
    double x,
    double y,
    double z,
    double w );
PARAMETERS

param
The parameter that will be set.

x, y, z, w
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter4d sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4d has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter4d was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.332 cgSetParameter4dv

NAME
cgSetParameter4dv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter4dv( CGparameter param,
                        const double * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter4dv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4dv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter4dv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.333 cgSetParameter4f

NAME

cgSetParameter4f - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter4f( CGparameter param,
  float x,
  float y,
  float z,
  float w );
PARAMETERS

param
The parameter that will be set.

x, y, z, w
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter4f sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4f has been used to set a parameter, the values may be retrieved from the parameter using the CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g. OpenGL) calls.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter4f was introduced in Cg 1.2.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.334 cgSetParameter4fv

NAME
cgSetParameter4fv - sets the value of scalar and vector parameters

SYNOPSIS
#include <Cg/cg.h>

void cgSetParameter4fv( CGparameter param,
                       const float * v );
PARAMETERS

param
    The parameter that will be set.

v
    Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter4fv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4fv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter4fv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

4.335 cgSetParameter4i

NAME

cgSetParameter4i - set the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter4i( CGparameter param,
    int x,
    int y,
    int z,
    int w );
PARAMETERS

param
The parameter that will be set.

x, y, z, w
The values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgSetParameter4i sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4i has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY
cgSetParameter4i was introduced in Cg 1.4.

SEE ALSO
cgGetParameterValue, cgGetParameterValues

4.336 cgSetParameter4iv

NAME
cgSetParameter4iv - sets the value of scalar and vector parameters

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameter4iv( CGparameter param,
        const int * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values to use to set param.

RETURN VALUES

None.

DESCRIPTION

cgSetParameter4iv sets the value of a given scalar or vector parameter.
If more values are passed in than param requires, the extra values will be ignored.
Once cgSetParameter4iv has been used to set a parameter, the values may be retrieved from the parameter using the
CG_CURRENT enumerant with cgGetParameterValues.
If an API-dependant layer of the Cg runtime (e.g. cgGL) is used, these entry points may end up making API (e.g.
OpenGL) calls.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

HISTORY

cgSetParameter4iv was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue

4.337 cgSetParameterSemantic

NAME

cgSetParameterSemantic - set a program parameter’s semantic

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameterSemantic( CGparameter param,
               const char * semantic );
PARAMETERS

   param
   The program parameter.
   semantic
   The semantic.

RETURN VALUES
None.

DESCRIPTION
cgSetParameterSemantic allows the application to set the semantic of a parameter in a Cg program.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a leaf node, or if the semantic string is NULL.

HISTORY
cgSetParameterSemantic was introduced in Cg 1.2.

SEE ALSO

cgGetParameterSemantic, cgGetSemanticCasePolicy, cgSetSemanticCasePolicy, cgGetParameterResource, cgGetParameterResourceIndex, cgGetParameterName, cgGetParameterType

4.338 cgSetParameterSettingMode

NAME
cgSetParameterSettingMode - set the parameter setting mode for a context

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameterSettingMode( CGcontext context, 
                               CGenum parameterSettingMode );

PARAMETERS

   context
   The context in which to set the parameter setting mode.
   parameterSettingMode
   The mode to which context will be set. Must be one of the following :
CG_IMMEDIATE_PARAMETER_SETTING
CG_DEFERRED_PARAMETER_SETTING

RETURN VALUES
None.

DESCRIPTION
cgSetParameterSettingMode controls the behavior of the context when setting parameters. With deferred parameter setting, the corresponding 3D API parameter is not immediately updated by cgSetParameter commands. If the application does not need to access these 3D API parameter values, then this mode allows improved performance by avoiding unnecessary 3D API calls.

Parameters of variability CG_VARYING are never deferred.

When the parameter setting mode is CG_DEFERRED_PARAMETER_SETTING, non-erroneous cgSetParameter commands record the updated parameter value but do not immediately update the corresponding 3D API parameter. Instead the parameter is marked internally as update deferred. The 3D API commands required to update any program parameters marked update deferred are performed as part of the next program bind (see cgGLBindProgram or cgD3D9BindProgram).

If a context’s parameter setting mode was CG_DEFERRED_PARAMETER_SETTING and one or more parameters are marked update deferred, changing the parameter setting mode to CG_IMMEDIATE_PARAMETER_SETTING does not cause parameters marked update deferred to be updated. The application can use cgUpdateProgramParameters or cgUpdatePassParameters to force the updating of parameters marked update deferred.

parameterSettingMode must be one of the following enumerants:
* CG_IMMEDIATE_PARAMETER_SETTING
  Non-erroneous cgSetParameter commands immediately update the corresponding 3D API parameter. This is the default mode.
* CG_DEFERRED_PARAMETER_SETTING
  Non-erroneous cgSetParameter commands record the updated parameter value but do not immediately update the corresponding 3D API parameter. These updates will happen during the next program bind. The updates can be explicitly forced to occur by using cgUpdateProgramParameters or cgUpdatePassParameters.

EXAMPLES
Change context to use deferred parameter updates:

cgSetParameterSettingMode(myCgContext, CG_DEFERRED_PARAMETER_SETTING);

Change context to its initial behavior of performing parameter updates immediately:

cgSetParameterSettingMode(myCgContext, CG_IMMEDIATE_PARAMETER_SETTING);

ERRORS
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_ENUMERANT_ERROR is generated if parameterSettingMode is not CG_IMMEDIATE_PARAMETER_SETTING or CG_DEFERRED_PARAMETER_SETTING.

HISTORY
cgSetParameterSettingMode was introduced in Cg 2.0.
SEE ALSO

cgGetParameterSettingMode, cgUpdateProgramParameters, cgUpdatePassParameters, cgSetParameter, cgSetParameterVariability, cgGetParameterVariability, cgGLBindProgram, cgD3D9BindProgram

4.339 cgSetParameterValue

NAME
cgSetParameterValue - set the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

/* TYPE is int, float or double */

void cgSetParameterValue{ifd}{rc}( CGparameter param,
    int nelements,
    const TYPE * v );

PARAMETERS

param
    The program parameter whose value will be set.

nelements
    The number of elements in array v.

v
    Source buffer from which the parameter values will be read.

RETURN VALUES

None.

DESCRIPTION

cgSetParameterValue allows the application to set the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

There are versions of each function that take int, float or double values signified by the i, f or d in the function name.

There are versions of each function that will cause any matrices referenced by param to be initialized in either row-major or column-major order, as signified by the r or c in the function name.

For example, cgSetParameterValueic sets the given parameter using the supplied array of integer data, and initializes matrices in column-major order.

If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;

Note: Previous releases of Cg allowed you to store more values in a parameter than indicated by the parameter’s
type. For example, one could use cgGLSetParameter4f to store four values into a parameter of type CG_FLOAT (not
CG_FLOAT4). All four values could later be retrieved using a get call which requested more than one value. How-
ever, this feature conflicts with the GLSL approach and also leads to issues with parameters mapped into BUFFERS.
Therefore, beginning with Cg 2.0 any components beyond the number indicated by the parameter type are ignored.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY
The cgSetParameterValue functions were introduced in Cg 1.4.

SEE ALSO
cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

4.340 cgSetParameterValuedc

NAME
cgSetParameterValuedc - set the value of any numeric parameter

SYNOPSIS
#include <Cg/cg.h>

void cgSetParameterValuedc( CGparameter param,
                          int nelements,
                          const double * v );

PARAMETERS

  param
  The program parameter whose value will be set.

  nelements
  The number of elements in array v.

  v
Source buffer from which the parameter values will be read.

RETURN VALUES

None.

DESCRIPTION

cgSetParameterValuedc allows the application to set the value of any numeric parameter or parameter array. The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param to be initialized in column-major order.

If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgSetParameterValuedc was introduced in Cg 1.4.

SEE ALSO

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

4.341 cgSetParameterValuedr

NAME

cgSetParameterValuedr - set the value of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetParameterValuedr( CGparameter param,
```

4.341. cgSetParameterValuedr
int nelements,
    const double * v );

PARAMETERS

    param
        The program parameter whose value will be set.

    nelements
        The number of elements in array v.

    v
        Source buffer from which the parameter values will be read.

RETURN VALUES

None.

DESCRIPTION

cgSetParameterValuedr allows the application to set the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param to be initialized in row-major order.

If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgSetParameterValuedr was introduced in Cg 1.4.

SEE ALSO

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue
4.342 cgSetParameterValuefc

NAME
cgSetParameterValuefc - set the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameterValuefc( CGparameter param,
    int nelements,
    const float * v );

PARAMETERS

param
The program parameter whose value will be set.
nelements
The number of elements in array v.
v
Source buffer from which the parameter values will be read.

RETURN VALUES

None.

DESCRIPTION
cgSetParameterValuefc allows the application to set the value of any numeric parameter or parameter array.
The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.
Any matrices referenced by param to be initialized in column-major order.
If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.
The total number of values in a parameter, ntotal, may be computed as follow:

```
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if `nelements` is less than the total size of `param`. CG_NON_NUMERIC_PARAMETER_ERROR is generated if `param` is of a non-numeric type.

**HISTORY**

cgSetParameterValuefc was introduced in Cg 1.4.

**SEE ALSO**

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

4.343 cgSetParameterValuefr

**NAME**

cgSetParameterValuefr - set the value of any numeric parameter

**SYNOPSIS**

```c
#include <Cg/cg.h>

void cgSetParameterValuefr( CGparameter param,
                            int nelements,
                            const float * v );
```

**PARAMETERS**

- `param`
  The program parameter whose value will be set.
- `nelements`
  The number of elements in array `v`.
- `v`
  Source buffer from which the parameter values will be read.

**RETURN VALUES**

None.

**DESCRIPTION**

cgSetParameterValuefr allows the application to set the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by `param` to be initialized in row-major order.

If `v` is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
```
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.
CG_INVALID_POINTER_ERROR is generated if v is NULL.
CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.
CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgSetParameterValuefr was introduced in Cg 1.4.

SEE ALSO

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

4.344  cgSetParameterValueic

NAME

cgSetParameterValueic - set the value of any numeric parameter

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameterValueic( CGparameter param,

int nelements,

const int * v );

PARAMETERS

param
The program parameter whose value will be set.
nelements
The number of elements in array v.
v
Source buffer from which the parameter values will be read.

RETURN VALUES

None.
DESCRIPTION

cgSetParameterValueic allows the application to set the value of any numeric parameter or parameter array.

The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or matrices.

Any matrices referenced by param to be initialized in column-major order.

If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is generated.

The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY

cgSetParameterValueic was introduced in Cg 1.4.

SEE ALSO

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

4.345  cgSetParameterValueir

NAME

cgSetParameterValueir - set the value of any numeric parameter

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetParameterValueir( CGparameter param,
                           int nelements,
                           const int * v );
```
PARAMETERS

param
The program parameter whose value will be set.

nelements
The number of elements in array v.

v
Source buffer from which the parameter values will be read.

RETURN VALUES
None.

DESCRIPTION
cgSetParameterValue
denotes the application to set the value of any numeric parameter or parameter array.
The given parameter must be a scalar, vector, matrix, or a (possibly multidimensional) array of scalars, vectors, or
matrices.
Any matrices referenced by param to be initialized in row-major order.
If v is smaller than the total number of values in the given source parameter, CG_NOT_ENOUGH_DATA_ERROR is
generated.
The total number of values in a parameter, ntotal, may be computed as follow:

```c
int nrows = cgGetParameterRows(param);
int ncols = cgGetParameterColumns(param);
int asize = cgGetArrayTotalSize(param);
int ntotal = nrows*ncols;
if (asize > 0) ntotal *= asize;
```

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is a varying input to a fragment program.

CG_INVALID_POINTER_ERROR is generated if v is NULL.

CG_NOT_ENOUGH_DATA_ERROR is generated if nelements is less than the total size of param.

CG_NON_NUMERIC_PARAMETER_ERROR is generated if param is of a non-numeric type.

HISTORY
cgSetParameterValue was introduced in Cg 1.4.

SEE ALSO
cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue
4.346  cgSetParameterVariability

NAME

cgSetParameterVariability - set a parameter’s variability

SYNOPSIS

#include <Cg/cg.h>

void cgSetParameterVariability( CGparameter param,
                                  CGenum vary );

PARAMETERS

  param
  The parameter.
  vary
  The variability to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgSetParameterVariability allows the application to change the variability of a parameter.

Currently parameters may not be changed to or from CG_VARYING variability. Also, the variability of uniform
buffer parameters and their member parameters may not be changed. However, other parameters of CG_UNIFORM
or CG_LITERAL variability may be changed.

Valid values for vary include :

  CG_UNIFORM
  A uniform parameter is one whose value does not change with each invocation of a program, but whose
  value can change between groups of program invocations.

  CG_LITERAL
  A literal parameter is folded out at compile time. Making a uniform parameter literal will often make a
  program more efficient at the expense of requiring a compile every time the value is set.

  CG_DEFAULT
  By default, the variability of a parameter will be overridden by the a source parameter connected to it
  unless it is changed with cgSetParameterVariability. If it is set to CG_DEFAULT it will restore the
  default state of assuming the source parameters variability.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_ENUMERANT_ERROR is generated if vary is not CG_UNIFORM, CG_LITERAL, or
CG_DEFAULT.
CG_INVALID_PARAMETER_VARIABILITY_ERROR is generated if the parameter could not be changed to the variability indicated by vary.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if vary is CG_LITERAL and param is not a numeric parameter, or if param is a uniform buffer or a member of a uniform buffer.

HISTORY

cgSetParameterVariability was introduced in Cg 1.2.

SEE ALSO

cgGetParameterVariability

4.347 cgSetPassProgramParameters

NAME

cgSetPassProgramParameters - set uniform parameters specified via a compile statement

SYNOPSIS

#include <Cg/cg.h>

void cgSetPassProgramParameters( CGprogram program );

PARAMETERS

program

The program

RETURN VALUES

None.

DESCRIPTION

Given the handle to a program specified in a pass in a CgFX file, cgSetPassProgramParameters sets the values of the program’s uniform parameters given the expressions in the compile statement in the CgFX file.

(This entrypoint is normally only needed by state managers and doesn’t need to be called by users.)

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgSetPassProgramParameters was introduced in Cg 1.4.

SEE ALSO

cgCreateEffect, cgCreateEffectFromFile
4.348  cgSetPassState

NAME
cgSetPassState - calls the state setting callback functions for all state assignments in a pass

SYNOPSIS
#include <Cg/cg.h>

void cgSetPassState( CGpass pass );

PARAMETERS
   pass
       The pass handle.

RETURN VALUES
None.

DESCRIPTION
cgSetPassState sets all of the graphics state defined in a pass by calling the state setting callbacks for all of the state assignments in the pass.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.
CG_INVALID_TECHNIQUE_ERROR is generated if the technique of which pass is a part has failed validation.

HISTORY
cgSetPassState was introduced in Cg 1.4.

SEE ALSO
cgResetPassState, cgCallStateSetCallback

4.349  cgSetProgramBuffer

NAME
cgSetProgramBuffer - set a buffer for a program

SYNOPSIS
#include <Cg/cg.h>

void cgSetProgramBuffer( Cgprogram program,
       int bufferIndex,
       CGbuffer buffer );
PARAMETERS

program
The program for which the buffer will be set.
bufferIndex
The buffer index of program to which buffer will be bound.
buffer
The buffer to be bound.

RETURN VALUES
None.

DESCRIPTION
cgSetProgramBuffer sets the buffer for a given buffer index of a program. A NULL buffer handle means the given buffer index should not be bound to a buffer.

bufferIndex must be non-negative and within the program’s range of buffer indices. For OpenGL programs, bufferIndex can be 0 to 11. For Direct3D10 programs, bufferIndex can be 0 to 15.

When the next program bind operation occurs, each buffer index which is set to a valid buffer handle is bound (along with the program) for use by the 3D API. No buffer bind operation occurs for buffer indices bound to a NULL buffer handle.

EXAMPLES
cgSetProgramBuffer( myProgram, 2, myBuffer );

ERRORS
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

CG_BUFFER_INDEX_OUT_OF_RANGE_ERROR is generated if bufferIndex is not within the valid range of buffer indices for program.

HISTORY
cgSetProgramBuffer was introduced in Cg 2.0.

SEE ALSO
cgCreateBuffer, cgGetProgramBuffer, cgGLBindProgram, cgD3D9BindProgram

4.350 cgSetProgramOutputVertices

NAME
cgSetProgramOutputVertices - set the maximum number of geometry program output vertices

SYNOPSIS

#include <Cg/cg.h>

void cgSetProgramOutputVertices( CGprogram prog, int vertices );
PARAMETERS

program

The program.

vertices

The maximum number of geometry program output vertices or -1 to use the compiler-detected upper bound.

RETURN VALUES

None.

DESCRIPTION

cgSetProgramOutputVertices allows the application to specify the maximum number of output vertices for a geometry program. In many cases an upper bound can be determined by the compiler and does not need to be specified by the application. If an upper bound can not be determined by the compiler and isn’t specified by the application, then the maximum supported number of outputs will be used, with possible performance implications.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgSetProgramOutputVertices was introduced in Cg 3.1.

SEE ALSO

cgGetProgramOutputVertices, cgSetProgramProfile, gp4gp, gp5gp, glslg.

4.351 cgSetProgramProfile

NAME

cgSetProgramProfile - set a program’s profile

SYNOPSIS

#include <Cg/cg.h>

void cgSetProgramProfile( CGprogram program,
                          CGprofile profile );

PARAMETERS

program

The program.

profile

The profile to be used when compiling the program.
RETURN VALUES
None.

DESCRIPTION

cgSetProgramProfile allows the application to specify the profile to be used when compiling the given program. When called, the program will be unloaded if it is currently loaded, and marked as uncompiled. When the program is next compiled (see cgSetAutoCompile), the given profile will be used. cgSetProgramProfile can be used to override the profile specified in a CgFX compile statement, or to change the profile associated with a program created by a call to cgCreateProgram.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.
CG_INVALID_PROFILE_ERROR is generated if profile is not a valid profile enumerant.

HISTORY
cgSetProgramProfile was introduced in Cg 1.4.

SEE ALSO
cgGetProgramProfile, cgGetProfile, cgGetProfileString, cgCreateProgram, cgSetAutoCompile

4.352 cgSetProgramStateAssignment

NAME
cgSetProgramStateAssignment - set the value of a program state assignment

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetProgramStateAssignment( CGstateassignment sa, CGprogram program );

PARAMETERS

sa
A handle to a state assignment of type CG_PROGRAM_TYPE.

program
The program object to which sa will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.

Returns CG_FALSE otherwise.
DESCRIPTION

cgSetProgramStateAssignment sets the value of a state assignment of program type.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a program type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgSetProgramStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetProgramStateAssignmentValue, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

4.353 cgSetSamplerState

NAME

cgSetSamplerState - initializes the state specified for a sampler parameter

SYNOPSIS

#include <Cg/cg.h>

void cgSetSamplerState( CGparameter param );

PARAMETERS

param

The parameter handle.

RETURN VALUES

None.

DESCRIPTION

cgSetSamplerState sets the sampler state for a sampler parameter that was specified via a sampler_state block in a CgFX file. The corresponding sampler should be bound via the graphics API before this call is made.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgSetSamplerState was introduced in Cg 1.4.

SEE ALSO

cgCreateSamplerState, cgGetFirstSamplerState, cgGetNamedSamplerState, cgGetNextState

4.354 cgSetSamplerStateAssignment

NAME

cgSetSamplerStateAssignment - sets a state assignment to a sampler effect parameter.

SYNOPSIS

#include <Cg/cg.h>

CGbool cgSetSamplerStateAssignment( CGstateassignment sa,
                                   CGparameter param );

PARAMETERS

sa

A state assignment of a sampler type (one of CG_SAMPLER1D, CG_SAMPLER2D,
CG_SAMPLER3D, CG_SAMPLERCUBE, or CG_SAMPLERRECT).

param

An effect parameter of a sampler type.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the state assignment.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetSamplerStateAssignment sets a state assignment of a sampler type to an effect parameter of the same sampler type.

EXAMPLES

CGparameter effectParam = cgCreateEffectParameter(effect,
                                    "normalizeCube",
                                    CG_SAMPLERCUBE);
CGstate state = cgGetNamedSamplerState(context, "TextureCubeMap");
CGstateassignment sa = cgCreateStateAssignment(technique, state);
CGbool ok = cgSetSamplerStateAssignment(sa, effectParam);
ERRORS

CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of a sampler type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgSetSamplerStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetSamplerStateAssignmentValue, cgSetTextureStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetStringStateAssignment

4.355 cgSetSemanticCasePolicy

NAME

cgSetSemanticCasePolicy - set semantic case policy

SYNOPSIS

#include <Cg/cg.h>

CGenum cgSetSemanticCasePolicy( CGenum casePolicy );

PARAMETERS

casePolicy

An enumerant describing the desired semantic case policy for the library. The following enumerants are allowed:

CG_FORCE_UPPER_CASE_POLICY

Semantics strings will be converted to all upper-case letters. This is the default policy.

CG_UNCHANGED_CASE_POLICY

Semantic strings will be left unchanged.

RETURN VALUES

Returns the previous semantic case policy, or CG_UNKNOWN if an error occurs.

DESCRIPTION

cgSetSemanticCasePolicy allows an application to change the semantic case policy used by the Cg library. A policy of CG_FORCE_UPPER_CASE_POLICY means that semantic strings returned by cgGetParameterSemantic will have been converted to all upper-case letters. This is the default policy for the library. If the policy is changed to CG_UNCHANGED_CASE_POLICY no case conversion will be done to the semantic strings.
EXAMPLES

/* set to return original semantic strings */
cgSetSemanticCasePolicy(CG_UNCHANGED_CASE_POLICY);

ERRORS

CG_INVALID_ENUMERANT_ERROR is generated if casePolicy is not
CG_FORCE_UPPER_CASE_POLICY or CG_UNCHANGED_CASE_POLICY.

HISTORY

cgSetSemanticCasePolicy was introduced in Cg 2.0.

SEE ALSO

cgGetSemanticCasePolicy, cgGetParameterSemantic, cgSetParameterSemantic

4.356 cgSetStateCallbacks

NAME

cgSetStateCallbacks - registers the callback functions for a state assignment

SYNOPSIS

#include <Cg/cg.h>

void cgSetStateCallbacks( CGstate state,
                          CGstatecallback set,
                          CGstatecallback reset,
                          CGstatecallback validate );

PARAMETERS

state
The state handle.

set
The pointer to the callback function to call for setting the state of state assignments based on state. This
may be a NULL pointer.

reset
The pointer to the callback function to call for resetting the state of state assignments based on state. This
may be a NULL pointer.

validate
The pointer to the callback function to call for validating the state of state assignments based on state. This
may be a NULL pointer.

RETURN VALUES

None.
DESCRIPTION

cgSetStateCallbacks sets the three callback functions for a state definition. These functions are later called when the state a particular state assignment based on this state must be set, reset, or validated. Any of the callback functions may be specified as NULL.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

HISTORY

cgSetStateCallbacks was introduced in Cg 1.4.

SEE ALSO

cgSetPassState, cgCallStateSetCallback, cgCallStateResetCallback, cgCallStateValidateCallback, cgValidateTechnique

4.357 cgSetStateLatestProfile

NAME

cgSetStateLatestProfile - sets a state’s designated latest profile

SYNOPSIS

#include <Cg/cg.h>

CGprofile cgSetStateLatestProfile( CGstate state, CGprofile profile );

PARAMETERS

state

The state handle.

profile

The profile to designate as the state’s latest profile.

RETURN VALUES

None.

DESCRIPTION

cgSetStateLatestProfile sets the specified state’s designated latest profile for states of type CG_PROGRAM_TYPE. This profile is used to compile the program for a state assignment for the state where the profile in the compile statement is the identifier latest.
EXAMPLES

This shows how to force the designated latest state profile for the FragmentProgram state assignment to be the arbfp1 profile (even if cgGLRegisterStates was to register a different profile).

```c
cgGLRegisterStates(context);
CState state = cgGetNamedState(context, "FragmentProgram");
cgSetStateLatestProfile(state, CG_PROFILE_ARBFP1);
```

ERRORS

CG_INVALID_STATE_HANDLE_ERROR is generated if state is not a valid state.

CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if the type of state is not CG_PROGRAM_TYPE.

HISTORY

cgSetStateLatestProfile was introduced in Cg 2.2.

SEE ALSO

cgGetNamedState, cgGetStateLatestProfile, cgGLRegisterStates

4.358 cgSetStringAnnotation

NAME

cgSetStringAnnotation - set the value of a string annotation

SYNOPSIS

```c
#include <Cg/cg.h>

CGbool cgSetStringAnnotation( CGannotation ann,
                           const char * value );
```

PARAMETERS

ann

The annotation that will be set.

value

The value to which ann will be set.

RETURN VALUES

Returns CG_TRUE if it succeeds in setting the annotation.

Returns CG_FALSE otherwise.

DESCRIPTION

cgSetStringAnnotation sets the value of an annotation of string type.

EXAMPLES

to-be-written
CG_INVALID_ANNOTATION_HANDLE_ERROR is generated if `ann` is not a valid annotation.
CG_INVALID_PARAMETER_TYPE_ERROR is generated if `ann` is not an annotation of string type.
CG_ARRAY_SIZE_MISMATCH_ERROR is generated if `ann` is not a scalar.

**HISTORY**

`cgSetStringAnnotation` was introduced in Cg 1.5.

**SEE ALSO**

`cgGetStringAnnotationValue`, `cgSetBoolAnnotation`, `cgSetIntAnnotation`, `cgSetFloatAnnotation`

### 4.359 cgSetStringParameterValue

**NAME**

`cgSetStringParameterValue` - set the value of a string parameter

**SYNOPSIS**

```c
#include <Cg/cg.h>

void cgSetStringParameterValue( CGparameter param,
                                const char * value );
```

**PARAMETERS**

- `param`
  The parameter whose value will be set.
- `value`
  The string to set the parameter’s value as.

**RETURN VALUES**

None.

**DESCRIPTION**

`cgSetStringParameterValue` allows the application to set the value of a string parameter.

**EXAMPLES**

`to-be-written`

**ERRORS**

CG_INVALID_PARAM_HANDLE_ERROR is generated if `param` is not a valid parameter.
CG_INVALID_PARAMETER_TYPE_ERROR is generated if `param` is not string-typed.
CG_INVALID_PARAMETER_ERROR is generated if `value` is NULL.

**HISTORY**

`cgSetStringParameterValue` was introduced in Cg 1.4.
SEE ALSO

cgGetStringParameterValue

4.360 cgSetStringStateAssignment

NAME
cgSetStringStateAssignment - set the value of a string state assignment

SYNOPSIS

```c
#include <Cg/cg.h>

CGbool cgSetStringStateAssignment( CGstateassignment sa,
                                  const char * value );
```

PARAMETERS

- `sa`
  A handle to a state assignment of type `CG_STRING`.
- `value`
  The value to which `sa` will be set.

RETURN VALUES

Returns `CG_TRUE` if it succeeds in setting the state assignment.
Returns `CG_FALSE` otherwise.

DESCRIPTION

cgSetStringStateAssignment sets the value of a state assignment of string type.

EXAMPLES

to-be-written

ERRORS

- `CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR` is generated if `sa` is not a valid state assignment.
- `CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR` is generated if `sa` is not a state assignment of a string type.
- `CG_ARRAY_SIZE_MISMATCH_ERROR` is generated if `sa` is an array and not a scalar.

HISTORY

cgSetStringStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetStringStateAssignmentValue, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetTextureStateAssignment
4.361 cgSetTextureStateAssignment

NAME
cgSetTextureStateAssignment - sets a state assignment to a texture effect parameter

SYNOPSIS
#include <Cg/cg.h>

CGbool cgSetTextureStateAssignment( CGstateassignment sa,
                       CGparameter param );

PARAMETERS
 sa
    A state assignment of type CG_TEXTURE.
 param
    An effect parameter of type CG_TEXTURE.

RETURN VALUES
Returns CG_TRUE if it succeeds in setting the state assignment.
Returns CG_FALSE otherwise.

DESCRIPTION
cgSetTextureStateAssignment sets the value of a state assignment of texture type to an effect parameter of type CG_TEXTURE.

EXAMPLES
CGparameter effectParam = cgCreateEffectParameter(effect,
                       "normalizeCube",
                       CG_SAMPLERCUBE);
CGstate state = cgGetNamedSamplerState(context, "Texture");
CGstateassignment sa = cgCreateSamplerStateAssignment(effectParam, state);
CGbool ok = cgSetTextureStateAssignment(sa, value);

ERRORS
CG_INVALID_STATE_ASSIGNMENT_HANDLE_ERROR is generated if sa is not a valid state assignment.
CG_STATE_ASSIGNMENT_TYPE_MISMATCH_ERROR is generated if sa is not a state assignment of texture type.
CG_ARRAY_SIZE_MISMATCH_ERROR is generated if sa is an array and not a scalar.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgSetTextureStateAssignment was introduced in Cg 1.5.
4.362 cgSetUniformBufferParameter

NAME

cgSetUniformBufferParameter - associate a buffer with a uniform buffer parameter

SYNOPSIS

```c
#include <Cg/cg.h>

void cgSetUniformBufferParameter( CGparameter param,
                                 CGbuffer buffer );
```

PARAMETERS

- `param`:
  The parameter with which `buffer` will be associated.
- `buffer`:
  The buffer to be associated with `param`.

RETURN VALUES

None.

DESCRIPTION

cgSetUniformBufferParameter sets the buffer for a given uniform buffer parameter. A NULL buffer handle means the given uniform buffer parameter should not be bound to a buffer.

When the next program bind operation occurs, each uniform buffer which is set to a valid buffer handle is bound (along with the program) for use by the 3D API. No buffer bind operation occurs for uniform buffer parameters bound to a NULL buffer handle.

EXAMPLES

```c
cgSetUniformBufferParameter( myParam, myBuffer );
```

ERRORS

- `CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter handle.
- `CG_INVALID_PARAMETER_TYPE_ERROR` is generated if `param` is not a uniform buffer parameter.
- `CG_INVALID_BUFFER_HANDLE_ERROR` is generated if `buffer` is not a valid buffer.

HISTORY

cgSetUniformBufferParameter was introduced in Cg 3.1.

SEE ALSO

cgGetUniformBufferParameter, cgCreateBuffer, cgGLBindProgram, cgD3D9BindProgram
4.363 cgUnmapBuffer

NAME
cgUnmapBuffer - unmap buffer from application's address space

SYNOPSIS
#include <Cg/cg.h>

void cgUnmapBuffer( CGbuffer buffer );

PARAMETERS
buffer
The buffer which will be unmapped from the application's address space.

RETURN VALUES
None.

DESCRIPTION
cgUnmapBuffer unmaps a buffer from the application's address space.

EXAMPLES
unsigned char *bufferPtr = cgMapBuffer( myBuffer, CG_MAP_READ_WRITE );
memcpy( data, bufferPtr, size );
cgUnmapBuffer( myBuffer );

ERRORS
CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

HISTORY
cgUnmapBuffer was introduced in Cg 2.0.

SEE ALSO
cgMapBuffer, cgSetBufferData, cgSetBufferSubData, cgSetParameter

4.364 cgUpdatePassParameters

NAME
cgUpdatePassParameters - update the deferred parameters for a pass

SYNOPSIS
#include <Cg/cg.h>

void cgUpdatePassParameters( CGpass pass );
PARAMETERS

pass

The pass for which deferred parameters will be updated.

RETURN VALUES

None.

DESCRIPTION

cgUpdatePassParameters is a convenience routine which calls cgUpdateProgramParameters for all programs of a pass.

EXAMPLES

cgSetParameterSettingMode(context, CG_DEFERRED_PARAMETER_SETTING);
CGeffect effect = cgCreateEffectFromFile( context, "tst.cgfx", NULL );
CGtechnique tech1 = cgGetNamedTechnique( effect, "tech1" );
CGpass pass1 = cgGetNamedPass( tech1, "pass1" );
cgSetPassState(pass1);

for (some number of times)
{
    cgSetParameter(uniform1,...);
    cgSetParameter(uniform2,...);
    cgUpdatePassParameters(pass1);
    DrawSomeGeometry();
}
cgResetPassState(pass1);

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if pass is not a valid pass.

HISTORY

cgUpdatePassParameters was introduced in Cg 2.1.

SEE ALSO

cgSetParameterSettingMode, cgGetParameterSettingMode, cgUpdateProgramParameters, cgSetParameter, cgGLBindProgram, cgD3D9BindProgram

4.365 cgUpdateProgramParameters

NAME

cgUpdateProgramParameters - update the 3D API state for deferred parameters
SYNOPSIS

#include <Cg/cg.h>

void cgUpdateProgramParameters( CGprogram program );

PARAMETERS

program

The program for which deferred parameters will be sent to the corresponding 3D API parameters.

RETURN VALUES

None.

DESCRIPTION

cgUpdateProgramParameters performs the appropriate 3D API commands to set the 3D API resources for all of program’s parameters that are marked update deferred and clears the update deferred state of these parameters. cgUpdateProgramParameters does nothing when none of program’s parameters are marked update deferred.

cgUpdateProgramParameters assumes the specified program has already been bound using the appropriate 3D API commands. Results are undefined if the program is not actually bound by the 3D API when cgUpdateProgramParameters is called, with the likely result being that the 3D API state for program’s parameters will be mis-loaded.

EXAMPLES

/* assumes cgGetProgramContext(program) == context */

if ( cgGetParameterSettingMode(context) == CG_DEFERRED_PARAMETER_SETTING) {
    cgUpdateProgramParameters(program);
}

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgUpdateProgramParameters was introduced in Cg 2.0.

SEE ALSO

cgSetParameterSettingMode, cgGetParameterSettingMode, cgUpdatePassParameters, cgSetParameter, cgGLBindProgram, cgD3D9BindProgram

4.366 cgValidateTechnique

NAME

cgValidateTechnique - validate a technique from an effect

SYNOPSIS

#include <Cg/cg.h>

CGbool cgValidateTechnique( CGtechnique tech );
PARAMETERS

tech
    The technique handle to validate.

RETURN VALUES

Returns CG_TRUE if all of the state assignments in all of the passes in tech are valid and can be used on the current hardware.

Returns CG_FALSE if any state assignment fails validation, or if an error occurs.

DESCRIPTION

cgValidateTechnique iterates over all of the passes of a technique and tests to see if every state assignment in the pass passes validation.

EXAMPLES

CGcontext context = cgCreateContext();
CGeffect effect = cgCreateEffectFromFile(context, filename, NULL);

CGtechnique tech = cgGetFirstTechnique(effect);
while (tech && cgValidateTechnique(tech) == CG_FALSE) {
    fprintf(stderr, "Technique %s did not validate. Skipping.\n", cgGetTechniqueName(tech));
    tech = cgGetNextTechnique(tech);
}

if (tech) {
    fprintf(stderr, "Using technique %s.\n", cgGetTechniqueName(tech));
} else {
    fprintf(stderr, "No valid technique found\n");
    exit(1);
}

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if tech is not a valid technique.

HISTORY

cgValidateTechnique was introduced in Cg 1.4.

SEE ALSO

cgCallStateValidateCallback, cgSetStateCallbacks
5.1 cgGLBindProgram

NAME

cgGLBindProgram - bind a program to the current state

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLBindProgram( CGprogram program );

PARAMETERS

program

The program to bind to the current state.

RETURN VALUES

None.

DESCRIPTION

cgGLBindProgram binds a program to the current state. The program must have been loaded with cgGLLoadProgram before it can be bound. Also, the profile of the program must be enabled for the binding to work. This may be done with the cgGLEnableProfile function.

For profiles that do not support program local parameters (e.g. the vp20 profile), cgGLBindProgram will reset all uniform parameters that were set with any of the Cg parameter setting functions

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_INVALID_PROFILE_ERROR is generated if program's profile is not a supported OpenGL profile.

CG_PROGRAM_BIND_ERROR is generated if the program fails to bind for any reason.

HISTORY

cgGLBindProgram was introduced in Cg 1.1.
SEE ALSO

cgGLLoadProgram, cgGLSetParameter, cgGLSetMatrixParameter, cgGLSetTextureParameter, cgGLEnableProfile

5.2 cgGLCreateBufferFromObject

NAME

cgGLCreateBufferFromObject - create a Cg buffer from an OpenGL buffer object

SYNOPSIS

#include <Cg/cgGL.h>

CGbuffer cgGLCreateBufferFromObject( CGcontext context, 
    GLuint obj, 
    CGbool manageObject );

PARAMETERS

code

    context
    The context to which the new buffer will be added.

    obj
    One of the usage flags specified as valid for glBufferData.

    manageObject
    A boolean switch which controls whether obj will be deleted by the runtime when the CGbuffer object returned by cgGLCreateBufferFromObject is destroyed.

RETURN VALUES

Returns a CGbuffer handle on success.

Returns NULL if any error occurs.

DESCRIPTION

cgGLCreateBufferFromObject creates a Cg buffer from a preexisting OpenGL buffer object. This GL object will be deleted by the runtime when the Cg buffer is destroyed if manageObject is CG_TRUE. Otherwise the application is responsible for deleting the GL object obj.

EXAMPLES

CGbuffer myBuffer = cgGLCreateBufferFromObject( myCgContext, glBufferId, CG_TRUE );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_MEMORY_ALLOC_ERROR is generated if a buffer couldn’t be created.

HISTORY

cgGLCreateBufferFromObject was introduced in Cg 3.1.
5.3 cgGLCreateBuffer

NAME

cgGLCreateBuffer - create an OpenGL buffer object

SYNOPSIS

```c
#include <Cg/cgGL.h>

CGbuffer cgGLCreateBuffer( CGcontext context,
                           int size,
                           const void *data,
                           GLenum bufferUsage );
```

PARAMETERS

- context
  The context to which the new buffer will be added.
- size
  The length in bytes of the buffer to create.
- data
  The initial data to be copied into the buffer. NULL will fill the buffer with zero.
- bufferUsage
  One of the usage flags specified as valid for `glBufferData`.

RETURN VALUES

Returns a `CGbuffer` handle on success.
Returns NULL if any error occurs.

DESCRIPTION

cgGLCreateBuffer creates an OpenGL buffer object.

EXAMPLES

```c
CGbuffer myBuffer = cgGLCreateBuffer( myCgContext, sizeof( float ) * 16,
                                       myData, GL_STATIC_DRAW );
```

ERRORS

- `CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.
- `CG_MEMORY_ALLOC_ERROR` is generated if a buffer couldn’t be created.

HISTORY

cgGLCreateBuffer was introduced in Cg 2.0.
5.4 cgGLDetectGLSLVersion

NAME

cgGLDetectGLSLVersion - detect the GLSL version of the current OpenGL context

SYNOPSIS

#include <Cg/cgGL.h>

CGGLglslversion cgGLDetectGLSLVersion( void );

PARAMETERS

None

RETURN VALUES

Returns a CGGLglslversion enumerant.

Returns CG_GL_GLSL_DEFAULT if any error occurs.

DESCRIPTION

cgGLDetectGLSLVersion returns the highest GLSL version supported by the current OpenGL context and the Cg Runtime. See glsl for supported GLSL version details.

EXAMPLES

/* Create Cg context ... */
/* Create OpenGL context ... */
CGGLglslversion glslVersion = cgGLDetectGLSLVersion();
cgGLSetContextGLSLVersion(context, glslVersion);

ERRORS

None.

HISTORY

cgGLDetectGLSLVersion was introduced in Cg 3.1.

SEE ALSO

glsl, cgGLGetContextGLSLVersion, cgGLSetContextGLSLVersion

5.5 cgGLDisableClientState

NAME

cgGLDisableClientState - disables a vertex attribute in the OpenGL state
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLDisableClientState( CGparameter param );

PARAMETERS

  param

  The varying parameter for which the client state will be disabled.

RETURN VALUES

None.

DESCRIPTION

cgGLDisableClientState disables the vertex attribute associated with the given varying parameter.

EXAMPLES

cgGLDisableClientState(param);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLDisableClientState was introduced in Cg 1.1.

SEE ALSO

cgGLEnableClientState

5.6 cgGLDisableProfile

NAME

cgGLDisableProfile - disable a profile within OpenGL

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLDisableProfile( CGprofile profile );

PARAMETERS

  profile

  The enumerant for the profile to disable.

RETURN VALUES

None.
DESCRIPTION

cgGLDisableProfile disables a profile by making the necessary OpenGL calls. For most profiles, this will simply make a call to glDisable with the appropriate enumerant.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if profile is not a supported OpenGL profile.

HISTORY

cgGLDisableProfile was introduced in Cg 1.1.

SEE ALSO

cgGLEnableProfile

5.7 cgGLDisableProgramProfiles

NAME

cgGLDisableProgramProfiles - disable all profiles associated with a combined program

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLDisableProgramProfiles( CGprogram program );

PARAMETERS

program

The combined program for which the profiles will be disabled.

RETURN VALUES

None.

DESCRIPTION

cgGLDisableProgramProfiles disables the profiles for all of the programs in a combined program.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_INVALID_PROFILE_ERROR is generated if the profile for any of the programs in program is not a supported OpenGL profile.

HISTORY

cgGLDisableProgramProfiles was introduced in Cg 1.5.
SEE ALSO

cgGLEnableProgramProfiles, cgCombinePrograms

5.8 cgGLDisableTextureParameter

NAME
cgGLDisableTextureParameter - disables the texture unit associated with a texture parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLDisableTextureParameter( CGparameter param );

PARAMETERS

param

The texture parameter which will be disabled.

RETURN VALUES

None.

DESCRIPTION

cgGLDisableTextureParameter unbinds and disables the texture object associated param.

See cgGLEnableTextureParameter for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAMETER_ERROR is generated if param is not a texture parameter or if the parameter fails to set for any other reason.

HISTORY

cgGLDisableTextureParameter was introduced in Cg 1.1.

SEE ALSO

cgGLEnableTextureParameter, cgGLSetTextureParameter

5.9 cgGLEnableClientState

NAME

cgGLEnableClientState - enables a vertex attribute in the OpenGL state
SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLEnableClientState( CGparameter param );
```

PARAMETERS

param

The varying parameter for which the client state will be enabled.

RETURN VALUES

None.

DESCRIPTION
cgGLEnableClientState enables the vertex attribute associated with the given varying parameter.

EXAMPLES

cgGLEnableClientState(param);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLEnableClientState was introduced in Cg 1.1.

SEE ALSO

cgGLDisableClientState

5.10 cgGLEnableProfile

NAME
cgGLEnableProfile - enable a profile within OpenGL

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLEnableProfile( CGprofile profile );
```

PARAMETERS

profile

The enumerant for the profile to enable.

RETURN VALUES

None.
DESCRIPTION

cgGLEnableProfile enables a profile by making the necessary OpenGL calls. For most profiles, this will simply make a call to glEnable with the appropriate enumerant.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if profile is not a supported OpenGL profile.

HISTORY
cgGLEnableProfile was introduced in Cg 1.1.

SEE ALSO
cgGLDisableProfile

5.11 cgGLEnableProgramProfiles

NAME
cgGLEnableProgramProfiles - enable all profiles associated with a combined program

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLEnableProgramProfiles( CGprogram program );

PARAMETERS

program

The combined program for which the profiles will be enabled.

RETURN VALUES

None.

DESCRIPTION

cgGLEnableProgramProfiles enables the profiles for all of the programs in a combined program.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_INVALID_PROFILE_ERROR is generated if the profile for any of the programs in program is not a supported OpenGL profile.

HISTORY
cgGLEnableProgramProfiles was introduced in Cg 1.5.
**SEE ALSO**

cgGLDisableProgramProfiles, cgCombinePrograms

### 5.12 cgGLEnableTextureParameter

**NAME**
cgGLEnableTextureParameter - enables the texture unit associated with a texture parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

void cgGLEnableTextureParameter( CGparameter param );
```

**PARAMETERS**

- `param`:
  
  The texture parameter which will be enabled.

**RETURN VALUES**

None.

**DESCRIPTION**

cgGLEnableTextureParameter binds and enables the texture object associated with `param`. It must be called after `cgGLSetTextureParameter` is called but before the geometry is drawn.

cgGLDisableTextureParameter should be called once all of the geometry is drawn to avoid applying the texture to the wrong geometry and shaders.

**EXAMPLES**

*to-be-written*

**ERRORS**

- **CG_INVALID_PROFILE_ERROR** is generated if `param`'s profile is not a supported OpenGL profile. In particular, if `param` is not a parameter handle retrieved from a CGprogram but was instead retrieved from a CGeffect or is a shared parameter created at runtime, this error will be generated since those parameters do not have a profile associated with them.

- **CG_INVALID_PARAM_HANDLE_ERROR** is generated if `param` is not a valid parameter.

- **CG_INVALID_PARAMETER_ERROR** is generated if `param` is not a texture parameter or if the enable operation fails for any other reason.

**HISTORY**

cgGLEnableTextureParameter was introduced in Cg 1.1.

**SEE ALSO**

cgGLDisableTextureParameter, cgGLSetTextureParameter
5.13 cgGLGetBufferObject

NAME

cgGLGetBufferObject - get OpenGL buffer object for a buffer

SYNOPSIS

#include <Cg/cgGL.h>

GLuint cgGLGetBufferObject( CGbuffer buffer );

PARAMETERS

buffer

The buffer for which the associated OpenGL buffer object will be retrieved.

RETURN VALUES

Returns the OpenGL buffer object associated with buffer.

Returns 0 if an error occurs.

DESCRIPTION

cgGLGetBufferObject returns the OpenGL buffer object associated with a buffer.

EXAMPLES

GLuint id = cgGLGetBufferObject( myBuffer );

ERRORS

CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

HISTORY

cgGLGetBufferObject was introduced in Cg 2.0.

SEE ALSO

cgCreateBuffer, cgGLCreateBuffer

5.14 cgGLGetContextGLSLVersion

NAME

cgGLGetContextGLSLVersion - get the current GLSL version from a context

SYNOPSIS

#include <Cg/cgGL.h>

CGGLglslversion cgGLGetContextGLSLVersion( CGcontext context );

5.13. cgGLGetBufferObject
PARAMETERS
  context
  The context from which the current GLSL version will be retrieved.

RETURN VALUES
Returns the GLSL version for context.
Returns CG_GL_GLSL_DEFAULT if an error occurs.

DESCRIPTION
cgGLGetContextGLSLVersion gets the current GLSL version setting from context. See cgGLSetContextGLSLVersion for more information.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY
cgGLGetContextGLSLVersion was introduced in Cg 3.1.

SEE ALSO
cgGLSetContextGLSLVersion, cgGLDetectGLSLVersion, cgGLGetContextOptimalOptions, cgGLSetContextOptimalOptions

5.15 cgGLGetContextOptimalOptions

NAME
cgGLGetContextOptimalOptions - get the best set of compiler options for a profile

SYNOPSIS
#include <Cg/cgGL.h>

char const ** cgGLGetContextOptimalOptions( CGcontext context, CGprofile profile );

PARAMETERS
  contest
  The context scope for profile-specific options.
  profile
  The profile whose optimal arguments are requested.

RETURN VALUES
Returns a null-terminated array of strings representing the optimal set of compiler options for profile in context.
Returns NULL if profile isn’t supported by the current driver or GPU.
DESCRIPTION

`cgGLGetContextOptimalOptions` returns the best set of compiler options for a given profile on the current driver and GPU. Note that different driver/GPU combinations might return different sets of options for the same `profile` value.

The elements of the returned array are meant to be used as part of the `args` parameter to `cgCreateProgram` or `cgCreateProgramFromFile`.

The strings returned for each value of `profile` remain valid until the next time `cgGLGetContextOptimalOptions` is called with the same values of `context` and `profile`.

The application does not need to destroy the returned strings.

Note: `cgGLGetContextOptimalOptions` replaces `cgGLGetOptimalOptions`.

EXAMPLES

```c
char const ** ppOptions = cgGLGetContextOptimalOptions(context, vertProfile);

if (ppOptions && *ppOptions) {
    while (*ppOptions) {
        printf("%s\n", *ppOptions);
        ppOptions++;
    }
}

const char* vertOptions[] = { myCustomArgs,
                               ppOptions,
                               NULL };  

CGprogram myVS = cgCreateProgramFromFile( context,
                                          CG_SOURCE,
                                          "vshader.cg",
                                          vertProfile,
                                          "VertexShader",
                                          vertOptions);
```

ERRORS

`CG_INVALID_CONTEXT_HANDLE_ERROR` is generated if `context` is not a valid context.

HISTORY

`cgGLGetContextOptimalOptions` was introduced in Cg 3.1.

SEE ALSO

`cgGLSetContextOptimalOptions`, `cgGLGetLatestProfile`, `cgCreateProgram`, `cgCreateProgramFromFile`

5.16 `cgGLGetGLSLVersion`

NAME

cgGLGetGLSLVersion - get the GLSL version enumerant from a string

SYNOPSIS
```c
#include <Cg/cgGL.h>

CGGLglslversion cgGLGetGLSLVersion(const char *version_string);

PARAMETERS

version_string
   A string containing the GLSL version name.

RETURN VALUES

Returns the CGGLglslversion enumerant associated with version_string.
Returns CG_GL_GLSL_DEFAULT if an error occurs.

DESCRIPTION

cgGLGetGLSLVersion returns the CGGLglslversion enumerant from a string.

EXAMPLES

CGGLglslversion v100 = cgGLGetGLSLVersion("1");
CGGLglslversion v110 = cgGLGetGLSLVersion("1.1");
CGGLglslversion v120 = cgGLGetGLSLVersion("1.20");
CGGLglslversion vFoo = cgGLGetGLSLVersion("foo");

assert(v100==CG_GL_GLSL_100);
assert(v110==CG_GL_GLSL_110);
assert(v120==CG_GL_GLSL_120);
assert(vFoo==CG_GL_GLSL_DEFAULT);

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if version_string is NULL.

HISTORY

cgGLGetGLSLVersion was introduced in Cg 3.1.

SEE ALSO

cgGLGetGLSLVersionString, cgGLGetContextGLSLVersion, cgGLSetContextGLSLVersion

5.17 cgGLGetGLSLVersionString

NAME

cgGLGetGLSLVersionString - get the GLSL version string for a CGGLglslversion enumerant

SYNOPSIS

#include <Cg/cgGL.h>

const char * cgGLGetGLSLVersionString( CGGLglslversion version );
```
PARAMETERS

version

CGGLGLslversion enumerant value.

RETURN VALUES

Returns the string associated with the given GLSL version enumerant.

Returns NULL if version is not a recognized enumerant.

DESCRIPTION

cgGLGetGLSLVersionString returns a GLSL version string from a CGGLGLslversion enumerant value.

EXAMPLES

printf("GLSL Version %s
", cgGLGetGLSLVersionString(CG_GL_GLSL_120));

ERRORS

None.

HISTORY

cgGLGetGLSLVersionString was introduced in Cg 3.1.

SEE ALSO

cgGLGetGLSLVersion, cgGLGetContextGLSLVersion, cgGLSetContextGLSLVersion

5.18 cgGLGetLatestProfile

NAME

cgGLGetLatestProfile - get the latest profile for a profile class

SYNOPSIS

#include <Cg/cgGL.h>

CGprofile cgGLGetLatestProfile( CGGLenum profileClass );

PARAMETERS

profileClass

The class of profile that will be returned. Must be one of the following:

* CG_GL_VERTEX
* CG_GL_GEOMETRY
* CG_GL_FRAGMENT
* CG_GL_TESSELLATION_CONTROL
* CG_GL_TESSELLATION_EVALUATION
RETURN VALUES

Returns a profile enumerant for the latest profile of the given class.

Returns CG_PROFILE_UNKNOWN if no appropriate profile is available or an error occurs.

DESCRIPTION

cgGLGetLatestProfile returns the best available profile of a given class. The OpenGL extensions are checked to determine the best profile which is supported by the current GPU, driver, and cgGL library combination.

profileClass may be one of the following enumerants:

* CG_GL_VERTEX
  The latest available vertex profile will be returned.
* CG_GL_GEOMETRY
  The latest available geometry profile will be returned.
* CG_GL_FRAGMENT
  The latest available fragment profile will be returned.
* CG_GL_TESSELLATION_CONTROL
  The latest available tessellation control profile will be returned.
* CG_GL_TESSELLATION_EVALUATION
  The latest available tessellation evaluation profile will be returned.

cgGLGetLatestProfile can be used in conjunction with cgCreateProgram to ensure that more optimal profiles are used as they are made available, even though they might not have been available at compile time or with a different version of the runtime.

Starting in Cg 2.2, certain environment variables can override the value returned by cgGLGetLatestProfile:

If cgGLGetLatestProfile is called with profileClass being CG_GL_VERTEX and an environment variable named CGGL_LATEST_VERTEX_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), the CGprofile value returned by cgGetProfile is returned by cgGLGetLatestProfile.

If cgGLGetLatestProfile is called with profileClass being CG_GL_GEOMETRY and an environment variable named CGGL_LATEST_GEOMETRY_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), the CGprofile value returned by cgGetProfile is returned by cgGLGetLatestProfile.

If cgGLGetLatestProfile is called with profileClass being CG_GL_FRAGMENT and an environment variable named CGGL_LATEST_FRAGMENT_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), the CGprofile value returned by cgGetProfile is returned by cgGLGetLatestProfile.

If cgGLGetLatestProfile is called with profileClass being CG_GL_TESSELLATION_CONTROL and an environment variable named CGGL_LATEST_TESSELLATION_CONTROL_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), the CGprofile value returned by cgGetProfile is returned by cgGLGetLatestProfile.

If cgGLGetLatestProfile is called with profileClass being CG_GL_TESSELLATION_EVALUATION and an environment variable named CGGL_LATEST_TESSELLATION_EVALUATION_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), the CGprofile value returned by cgGetProfile is returned by cgGLGetLatestProfile.
EXAMPLES

/* Output information about available profiles */
printf("vertex profile: %s\n",
cgGetProfileString(cgGLGetLatestProfile(CG_GL_VERTEX)));
printf("geometry profile: %s\n",
cgGetProfileString(cgGLGetLatestProfile(CG_GL_GEOMETRY)));
printf("fragment profile: %s\n",
cgGetProfileString(cgGLGetLatestProfile(CG_GL_FRAGMENT)));
printf("tessellation control profile: %s\n",
cgGetProfileString(cgGLGetLatestProfile(CG_GL_TESSELLATION_CONTROL)));
printf("tessellation evaluation profile: %s\n",
cgGetProfileString(cgGLGetLatestProfile(CG_GL_TESSELLATION_EVALUATION)));

ERRORS

CG_INVALID_ENUMERANT_ERROR is generated if profileClass is not CG_GL_VERTEX, CG_GL_GEOMETRY, CG_GL_FRAGMENT, CG_GL_TESSELLATION_CONTROL, or CG_GL_TESSELLATION_EVALUATION.

HISTORY

cgGLGetLatestProfile was introduced in Cg 1.1.
CG_GL_GEOMETRY support was introduced in Cg 2.0.
CG_GL_TESSELLATION_CONTROL and CG_GL_TESSELLATION_EVALUATION support was introduced in Cg 3.0.

SEE ALSO

cgGLSetOptimalOptions, cgCreateProgram

5.19 cgGLGetManageTextureParameters

NAME

cgGLGetManageTextureParameters - gets the manage texture parameters flag from a context

SYNOPSIS

#include <Cg/cgGL.h>

CGbool cgGLGetManageTextureParameters( CGcontext context );

PARAMETERS

context

The context from which the automatic texture management setting will be retrieved.

RETURN VALUES

Returns the manage textures setting for context.

DESCRIPTION

cgGLGetManageTextureParameters gets the current manage textures setting from context. See cgGLSetManageTextureParameters for more information.
EXAMPLES
to-be-written

ERRORS
None.

HISTORY
cgGLGetManageTextureParameters was introduced in Cg 1.2.

SEE ALSO
cgGLSetManageTextureParameters, cgGLBindProgram, cgGLUnbindProgram

5.20 cgGLGetMatrixParameterArraydc

NAME
cgGLGetMatrixParameterArraydc - get the values from a matrix array parameter

SYNOPSIS
#include <Cg/cgGL.h>

void cgGLGetMatrixParameterArraydc( CGparameter param,
   long offset,
   long nelements,
   double * v );

PARAMETERS
param
   The matrix array parameter from which the values will be retrieved.

offset
   An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
   The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
   The array into which to retrieve the values. The size of v must be nelements times the number of elements in the matrix.

RETURN VALUES
None.

DESCRIPTION
cgGLGetMatrixParameterArraydc retrieves an array of values from a matrix array parameter using column-major ordering.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetMatrixParameterArraydc was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.21 cgGLGetMatrixParameterArraydr

NAME

cgGLGetMatrixParameterArraydr - get the values from a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterArraydr( CGparameter param,
    long offset,
    long nelements,
    double * v );

PARAMETERS

param
The matrix array parameter from which the values will be retrieved.

offset
An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array into which to retrieve the values. The size of v must be nelements times the number of elements in the matrix.
RETURN VALUES

None.

DESCRIPTION

cgGLGetMatrixParameterArraydr retrieves an array of values from a matrix array parameter using row-major ordering.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetMatrixParameterArraydr was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.22 cgGLGetMatrixParameterArrayfc

NAME

cgGLGetMatrixParameterArrayfc - get the values from a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterArrayfc( CGparameter param,
       long offset,
       long nelements,
       float * v );

PARAMETERS

param
   The matrix array parameter from which the values will be retrieved.
offset
   An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.
nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array into which to retrieve the values. The size of v must be nelements times the number of elements in the matrix.

RETURN VALUES
None.

DESCRIPTION
cgGLGetMatrixParameterArrayfc retrieves an array of values from a matrix array parameter using column-major ordering.

EXAMPLES

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGLGetMatrixParameterArrayfc was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.23 cgGLGetMatrixParameterArrayfr

NAME
cgGLGetMatrixParameterArrayfr - get the values from a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterArrayfr( CGparameter param,
    long offset,
    long nelements,
    float * v );
PARAMETERS

    param
    The matrix array parameter from which the values will be retrieved.
    offset
    An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first
    element of the array.
    nelements
    The number of elements to get. A value of 0 will default to the total number of elements in the array
    minus the value of offset.
    v
    The array into which to retrieve the values. The size of v must be nelements times the number of elements
    in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGLGetMatrixParameterArrayfr retrieves an array of values from a matrix array parameter using row-major or-
dering.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetMatrixParameterArrayfr was introduced in Cg 1.1.

SEE ALSO

.cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.24 cgGLGetMatrixParameterArray

NAME

cgGLGetMatrixParameterArray - get the values from a matrix array parameter
SYNOPSIS

```c
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLGetMatrixParameterArray{fd}{rc}( CGparameter param,
    long offset,
    long nelements,
    TYPE * v );
```

PARAMETERS

- **param**
  The matrix array parameter from which the values will be retrieved.

- **offset**
  An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

- **nelements**
  The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of `offset`.

- **v**
  The array into which to retrieve the values. The size of the array must be `nelements` times the number of elements in the matrix.

RETURN VALUES

None.

DESCRIPTION

The `cgGLGetMatrixParameterArray` functions retrieve an array of values from a matrix array parameter.

There are versions of the function that return either `float` or `double` values signified by `f` or `d` in the function name.

There are versions of the function that assume the array of values is laid out in either row or column order signified by `r` or `c` respectively in the function name.

EXAMPLES

*to-be-written*

ERRORS

- **CG_INVALID_PROFILE_ERROR** is generated if `param`’s profile is not a supported OpenGL profile.

- **CG ARRAY PARAM ERROR** is generated if `param` is not an array parameter.

- **CG NOT MATRIX PARAM ERROR** is generated if the elements of `param` are not matrix parameters.

- **CG OUT OF ARRAY_BOUNDS_ERROR** is generated if the `offset` or the `nelements` parameter is out of the array bounds.

- **CG INVALID PARAM HANDLE ERROR** is generated if `param` is not a valid parameter.

HISTORY

The `cgGLGetMatrixParameterArray` functions were introduced in Cg 1.1.
SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.25 cgGLGetMatrixParameterdc

NAME
cgGLGetMatrixParameterdc - get the values from a matrix parameter

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLGetMatrixParameterdc( CGparameter param,
                                double * matrix );
```

PARAMETERS

- **param**
  - The matrix parameter from which the values will be retrieved.

- **matrix**
  - An array of doubles into which the matrix values will be retrieved. The size must be the number of rows times the number of columns of **param**.

RETURN VALUES

None.

DESCRIPTION

**cgGLGetMatrixParameterdc** retrieves the values from a matrix parameter using column-major ordering.

**cgGLGetMatrixParameterdc** may only be called with uniform parameters.

EXAMPLES

*to-be-written*

ERRORS

- **CG_INVALID_PROFILE_ERROR** is generated if **param**'s profile is not a supported OpenGL profile.
- **CG_NOT_MATRIX_PARAM_ERROR** is generated if **param** is not a matrix parameter.
- **CG_INVALID_PARAM_HANDLE_ERROR** is generated if **param** is not a valid parameter.

HISTORY

**cgGLGetMatrixParameterdc** was introduced in Cg 1.1.

SEE ALSO

- cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter
5.26 cgGLGetMatrixParameterdr

NAME

cgGLGetMatrixParameterdr - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterdr( CGparameter param,
                                  double * matrix );

PARAMETERS

    param

    The matrix parameter from which the values will be retrieved.

    matrix

    An array of doubles into which the matrix values will be retrieved. The size
    must be the number of rows times the number of columns of param.

RETURN VALUES

None.

DESCRIPTION

cgGLGetMatrixParameterdr retrieves the values from a matrix parameter using row-major ordering.
cgGLGetMatrixParameterdr may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetMatrixParameterdr was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter

5.27 cgGLGetMatrixParameterfc

NAME

cgGLGetMatrixParameterfc - get the values from a matrix parameter
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterfc( CGparameter param,
                                float * matrix );

PARAMETERS

param
The matrix parameter from which the values will be retrieved.

matrix
An array of floats into which the matrix values will be retrieved. The size must be the number of rows times the number of columns of param.

RETURN VALUES

None.

DESCRIPTION

cgGLGetMatrixParameterfc retrieves the values from a matrix parameter using column-major ordering.
cgGLGetMatrixParameterfc may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetMatrixParameterfc was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter

5.28 cgGLGetMatrixParameterfr

NAME

cgGLGetMatrixParameterfr - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetMatrixParameterfr( CGparameter param,
                                float * matrix );
PARAMETERS

param
The matrix parameter from which the values will be retrieved.

matrix
An array of floats into which the matrix values will be retrieved. The size must be the number of rows times the number of columns of param.

RETURN VALUES
None.

DESCRIPTION
cgGLGetMatrixParameterf retrieves the values from a matrix parameter using row-major ordering. cgGLGetMatrixParameterf may only be called with uniform parameters.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGLGetMatrixParameterf was introduced in Cg 1.1.

SEE ALSO
cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter

5.29 cgGLGetMatrixParameter

NAME
cgGLGetMatrixParameter - get the values from a matrix parameter

SYNOPSIS

#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLGetMatrixParameter(fd)(rc){ CGparameter param,
                                   TYPE * matrix );
PARAMETERS

param
    The matrix parameter from which the values will be retrieved.

matrix
    An array into which the values will be retrieved. The size must be the number of rows times the number of columns of param.

RETURN VALUES

None.

DESCRIPTION

The cgGLGetMatrixParameter functions retrieve the values from a matrix parameter. There are versions of the function that return either float or double values signified by f or d in the function name. There are versions of the function that assume the array of values is laid out in either row or column order signified by r or c respectively in the function name.

The cgGLGetMatrixParameter functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

The cgGLGetMatrixParameter functions were introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter

5.30 cgGLGetOptimalOptions

NAME

cgGLGetOptimalOptions - deprecated, use cgGLGetContextOptimalOptions

SYNOPSIS

#include <Cg/cgGL.h>

char const ** cgGLGetOptimalOptions( CGprofile profile );
PARAMETERS

profile

The profile whose optimal arguments are requested.

RETURN VALUES

Returns a null-terminated array of strings representing the optimal set of compiler options for profile.

Returns NULL if profile isn’t supported by the current driver or GPU.

DESCRIPTION

cgGLGetOptimalOptions returns the best set of compiler options for a given profile on the current driver and GPU. Note that different driver/GPU combinations might return different sets of options for the same profile value.

The elements of the returned array are meant to be used as part of the args parameter to cgCreateProgram or cgCreateProgramFromFile.

The strings returned for each value of profile remain valid until the next time cgGLGetOptimalOptions is called with this profile value.

The application does not need to destroy the returned strings.

EXAMPLES

```c
char const ** ppOptions = cgGLGetOptimalOptions(vertProfile);

if (ppOptions && *ppOptions) {
    while (*ppOptions) {
        printf("%s\n", *ppOptions);
        ppOptions++;
    }
}

const char* vertOptions[] = { myCustomArgs, ppOptions, NULL };

CGprogram myVS = cgCreateProgramFromFile( context, 
                                          CG_SOURCE, 
                                          "vshader.cg", 
                                          vertProfile, 
                                          "VertexShader", 
                                          vertOptions);
```

ERRORS

None.

HISTORY

cgGLGetOptimalOptions was introduced in Cg 2.2.

cgGLGetOptimalOptions was deprecated in Cg 3.1, use cgGLGetContextOptimalOptions.

SEE ALSO

cgGLGetContextOptimalOptions
5.31 cgGLGetParameter1d

NAME
cgGLGetParameter1d - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter1d(CGparameter param, double* v);

PARAMETERS

param
    The parameter from which the values will be retrieved.

v
    Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameter1d extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter1d may only be called with uniform numeric parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameter1d was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.32 cgGLGetParameter1f

NAME
cgGLGetParameter1f - get the values from a scalar or vector parameter
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter1f( CGparameter param,
                        float * v );

PARAMETERS

    param
    The parameter from which the values will be retrieved.

    v
    Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameter1f extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter1f may only be called with uniform numeric parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameter1f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.33 cgGLGetParameter2d

NAME

cgGLGetParameter2d - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter2d( CGparameter param,
                        double * v );
PARAMETERS

param
The parameter from which the values will be retrieved.

v
Destination buffer into which the values will be written.

RETURN VALUES
None.

DESCRIPTION
cgGLGetParameter2d extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter2d may only be called with uniform numeric parameters.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGLGetParameter2d was introduced in Cg 1.1.

SEE ALSO
cgGLSetParameter, cgGLGetParameterArray

5.34 cgGLGetParameter2f

NAME
cgGLGetParameter2f - get the values from a scalar or vector parameter

SYNOPSIS
#include <Cg/cgGL.h>

void cgGLGetParameter2f( CGparameter param, 
                         float * v );

PARAMETERS

param
The parameter from which the values will be retrieved.

v
Destination buffer into which the values will be written.
RETURN VALUES
None.

DESCRIPTION

cgGLGetParameter2f extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter2f may only be called with uniform numeric parameters.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameter2f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.35 cgGLGetParameter3d

NAME

cgGLGetParameter3d - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter3d( CGparameter param,
                      double * v );

PARAMETERS

   param
   The parameter from which the values will be retrieved.
   v
   Destination buffer into which the values will be written.

RETURN VALUES
None.

DESCRIPTION

cgGLGetParameter3d extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter3d may only be called with uniform numeric parameters.
EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGLGetParameter3d was introduced in Cg 1.1.

SEE ALSO
cgGLSetParameter, cgGLGetParameterArray

5.36 cgGLGetParameter3f

NAME
cgGLGetParameter3f - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter3f( CGparameter param,
                            float * v );

PARAMETERS

param
    The parameter from which the values will be retrieved.

v
    Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION
cgGLGetParameter3f extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter3f may only be called with uniform numeric parameters.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
HISTORY

cgGLGetParameter3f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.37 cgGLGetParameter4d

NAME

cgGLGetParameter4d - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter4d( CGparameter param,
                         double * v );

PARAMETERS

param

    The parameter from which the values will be retrieved.

v

    Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameter4d extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter4d may only be called with uniform numeric parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameter4d was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray
5.38 cgGLGetParameter4f

NAME

cgGLGetParameter4f - get the values from a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameter4f( CGparameter param,
                          float * v );

PARAMETERS

  param
  The parameter from which the values will be retrieved.

  v
  Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameter4f extracts parameter values set by the cgGLSetParameter functions.
cgGLGetParameter4f may only be called with uniform numeric parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameter4f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.39 cgGLGetParameterArray1d

NAME

cgGLGetParameterArray1d - get the values from an array parameter
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray1d( CGparameter param,
    long offset,
    long nelements,
    const double * v );

PARAMETERS

    param
      The array parameter from which the values will be retrieved.

    offset
      An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

    nelements
      The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

    v
      Destination buffer into which the values will be written. The size of v must be nelements.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameterArray1d retrieves the values from a scalar or vector array parameter.
The function retrieves 1 value per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameterArray1d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray
5.40 cgGLGetParameterArray1f

NAME

cgGLGetParameterArray1f - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray1f( CGparameter param,
                               long offset,
                               long nelements,
                               const float * v );

PARAMETERS

param

The array parameter from which the values will be retrieved.

offset

An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements

The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v

Destination buffer into which the values will be written. The size of v must be nelements.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameterArray1f retrieves the values from a scalar or vector array parameter. The function retrieves 1 value per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameterArray1f was introduced in Cg 1.1.
SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.41 cgGLGetParameterArray2d

NAME
cgGLGetParameterArray2d - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray2d( CGparameter param,
                              long offset,
                              long nelements,
                              const double * v );

PARAMETERS

param
The array parameter from which the values will be retrieved.

offset
An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
Destination buffer into which the values will be written. The size of v must be 2 * nelements.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameterArray2d retrieves the values from a scalar or vector array parameter.
The function retrieves 2 values per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
HISTORY
cgGLGetParameterArray2d was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.42 cgGLGetParameterArray2f

NAME
cgGLGetParameterArray2f - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray2f( CGparameter param,
   long offset,
   long nelements,
   const float * v );

PARAMETERS

param
The array parameter from which the values will be retrieved.

offset
An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
Destination buffer into which the values will be written. The size of v must be 2 * nelements.

RETURN VALUES
None.

DESCRIPTION
cgGLGetParameterArray2f retrieves the values from a scalar or vector array parameter.
The function retrieves 2 values per array element.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_PROFILE_ERROR is generated if *param*’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if *param* is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if *offset* or *nelements* is outside the bounds of *param*.

CG_INVALID_PARAM_HANDLE_ERROR is generated if *param* is not a valid parameter.

HISTORY

cgGLGetParameterArray2f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.43 cgGLGetParameterArray3d

NAME

cgGLGetParameterArray3d - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray3d( CGparameter param,
  long offset,
  long nelements,
  const double * v );

PARAMETERS

*param*

The array parameter from which the values will be retrieved.

*offset*

An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

*nelements*

The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of *offset*.

*v*

Destination buffer into which the values will be written. The size of *v* must be 3 * *nelements*.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameterArray3d retrieves the values from a scalar or vector array parameter.

The function retrieves 3 values per array element.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameterArray3d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.44 cgGLGetParameterArray3f

NAME

cgGLGetParameterArray3f - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray3f( CGparameter param,
  long offset,
  long nelements,
  const float * v );

PARAMETERS

param
  The array parameter from which the values will be retrieved.

offset
  An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first
  element of the array.

nelements
  The number of elements to get. A value of 0 will default to the total number of elements in the array
  minus the value of offset.

v
  Destination buffer into which the values will be written. The size of v must be 3 * nelements.

RETURN VALUES

None.
DESCRIPTION

cgGLGetParameterArray3f retrieves the values from a scalar or vector array parameter.
The function retrieves 3 values per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameterArray3f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.45 cgGLGetParameterArray4d

NAME

cgGLGetParameterArray4d - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray4d( CGparameter param,
                              long offset,
                              long nelements,
                              const double * v );

PARAMETERS

param

The array parameter from which the values will be retrieved.

offset

An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements

The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v

Destination buffer into which the values will be written. The size of v must be 4 * nelements.
RETURN VALUES
None.

DESCRIPTION

cgGLGetParameterArray4d retrieves the values from a scalar or vector array parameter. The function retrieves 4 values per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
cgGLGetParameterArray4d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.46 cgGLGetParameterArray4f

NAME

cgGLGetParameterArray4f - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLGetParameterArray4f( CGparameter param,
                                long offset,
                                long nelements,
                                const float * v );

PARAMETERS

param
The array parameter from which the values will be retrieved.

offset
An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.
v

Destination buffer into which the values will be written. The size of v must be 4 * nelements.

RETURN VALUES

None.

DESCRIPTION

cgGLGetParameterArray4f retrieves the values from a scalar or vector array parameter.
The function retrieves 4 values per array element.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

cgGLGetParameterArray4f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.47 cgGLGetParameterArray

NAME

cgGLGetParameterArray - get the values from an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLGetParameterArray{1234}{fd}( CGparameter param,
     long offset,
     long nelements,
     const TYPE * v );

PARAMETERS

    param
    The array parameter from which the values will be retrieved.

    offset
An offset into the array parameter at which to start getting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to get. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
Destination buffer into which the values will be written. The size of v must be nelements times the vector size indicated by the number in the function name.

RETURN VALUES
None.

DESCRIPTION
The cgGLGetParameterArray functions retrieve the values from a scalar or vector array parameter. There are versions of each function that return either float or double values signified by f or d in the function name. Either 1, 2, 3, or 4 values per array element is returned depending on which function is used.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY
The cgGLGetParameterArray functions were introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

5.48 cgGLGetParameter

NAME
cgGLGetParameter - get the values from a scalar or vector parameter

SYNOPSIS
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLGetParameter{1234}{fd}( CGparameter param,
    TYPE * v );
PARAMETERS

param

The parameter from which the values will be retrieved.

v

Destination buffer into which the values will be written.

RETURN VALUES

None.

DESCRIPTION

The cgGLGetParameter functions extract the values set by cgGLSetParameter functions.

There are versions of the function that take either float or double values signified by f or d in the function name.

Each function returns either 1, 2, 3, or 4 values.

These functions may only be called with uniform numeric parameters.

Note: Previous releases of Cg allowed you to store more values in a parameter than indicated by the parameter’s type. For example, one could use cgGLSetParameter4f to store four values into a parameter of type CG_FLOAT (not CG_FLOAT4). All four values could later be retrieved using a get call which requested more than one value. However, this feature conflicts with the GLSL approach and also leads to issues with parameters mapped into BUFFERS. Therefore, beginning with Cg 2.0 any components beyond the number indicated by the parameter type are ignored.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

HISTORY

The cgGLGetParameter functions were introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.49 cgGLGetProgramID

NAME

cgGLGetProgramID - get the OpenGL program ID associated with a program

SYNOPSIS

#include <Cg/cgGL.h>

GLuint cgGLGetProgramID( CGprogram program );
PARAMETERS

  program

  The program for which the OpenGL program ID will be retrieved.

RETURN VALUES

Returns a GLuint associated with the GL program object for profiles that use program object.
Returns 0 for profiles that do not have OpenGL programs (e.g. fp20).

DESCRIPTION

cgGLGetProgramID returns the identifier to the OpenGL program object associated with program. cgGLGetProgramID should not be called before cgGLLoadProgram is called.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if program’s profile is not a supported OpenGL profile.
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGLGetProgramID was introduced in Cg 1.2.

SEE ALSO

cgGLLoadProgram, cgGLBindProgram

5.50 cgGLGetTextureEnum

NAME

cgGLGetTextureEnum - get the OpenGL enumerant for the texture unit associated with a parameter

SYNOPSIS

#include <Cg/cgGL.h>

GLenum cgGLGetTextureEnum( CGparameter param );

PARAMETERS

  param

  The texture parameter for which the OpenGL texture unit enumerant will be retrieved.

RETURN VALUES

Returns a GLenum of the form GL_TEXTURE#_ARB.
Returns GL_INVALID_OPERATION if an error occurs.
DESCRIPTION

cgGLGetTextureEnum returns the OpenGL enumerant for the texture unit assigned to param. The enumerant has the form GL_TEXTURE#_ARB where # is the texture unit number.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a texture parameter or if the operation fails for any other reason.

HISTORY

cgGLGetTextureEnum was introduced in Cg 1.1.

SEE ALSO

cgGLSetTextureParameter

5.51 cgGLGetTextureParameter

NAME

cgGLGetTextureParameter - get the OpenGL object from a texture parameter

SYNOPSIS

#include <Cg/cgGL.h>

GLuint cgGLGetTextureParameter( CGparameter param );

PARAMETERS

param

The texture parameter for which the OpenGL texture object will be retrieved.

RETURN VALUES

Returns the OpenGL object to which the texture was set.

Returns 0 if the parameter has not been set.

DESCRIPTION

cgGLGetTextureParameter gets the OpenGL object from a texture parameter.

EXAMPLES

to-be-written

ERRORS

None.
HISTORY
cgGLGetTextureParameter was introduced in Cg 1.1.

SEE ALSO
cgGLSetTextureParameter, cgGLGetParameter

5.52 cgGLIsProfileSupported

NAME
cgGLIsProfileSupported - determine if a profile is supported by cgGL

SYNOPSIS
#include <Cg/cgGL.h>

CGbool cgGLIsProfileSupported( CGprofile profile );

PARAMETERS
  profile
    The profile which will be checked for support.

RETURN VALUES
Returns CG_TRUE if profile is supported by the cgGL library.
Returns CG_FALSE otherwise.

DESCRIPTION
cgGLIsProfileSupported returns CG_TRUE if the profile indicated by profile is supported by the cgGL library. A profile may not be supported if required OpenGL extensions are not available.

EXAMPLES
to-be-written

ERRORS
None.

HISTORY
cgGLIsProfileSupported was introduced in Cg 1.1.

SEE ALSO
cgGLEnableProfile, cgGLDisableProfile

5.53 cgGLIsProgramLoaded

NAME
cgGLIsProgramLoaded - determine if a program is loaded
SYNOPSIS

#include <Cg/cgGL.h>

CGbool cgGLIsProgramLoaded( CGprogram program );

PARAMETERS

program

The program which will be checked.

RETURN VALUES

Returns CG_TRUE if program has been loaded.
Returns CG_FALSE otherwise.

DESCRIPTION

cgGLIsProgramLoaded returns CG_TRUE if program has been loaded with cgGLLoadProgram and CG_FALSE otherwise.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGLIsProgramLoaded was introduced in Cg 1.2.

SEE ALSO

cgGLLoadProgram cgGLBindProgram

5.54  cgGLLoadProgram

NAME

cgGLLoadProgram - prepares a program for binding

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLLoadProgram( CGprogram program );

PARAMETERS

program

The program which will be loaded.

RETURN VALUES

None.
DESCRIPTION

cgGLLoadProgram prepares a program for binding. All programs must be loaded before they can be bound to the current state. See cgGLBindProgram for more information about binding programs.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if program’s profile is not a supported OpenGL profile.

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

CG_PROGRAM_LOAD_ERROR is generated if the program fails to load for any reason.

HISTORY

cgGLLoadProgram was introduced in Cg 1.1.

SEE ALSO

cgGLIsProgramLoaded, cgGLBindProgram

5.55 cgGLRegisterStates

NAME

cgGLRegisterStates - registers graphics pass states for CgFX files

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLRegisterStates( CGcontext context );

PARAMETERS

context

The context in which to register the states.

RETURN VALUES

None.

DESCRIPTION

cgGLRegisterStates registers a set of states for the passes in a CgFX effect file. These states correspond to the set of OpenGL state that is relevant and/or useful to be setting in passes in effect files. See the Cg User’s Guide for complete documentation of the states that are made available after calling cgGLRegisterStates.

EXAMPLES

CGcontext context = cgCreateContext();
HGLRC glcontext = wglCreateContext(hdc);
wglMakeCurrent(hdc, glcontext);
cgGLRegisterStates(context);
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgGLRegisterStates was introduced in Cg 1.4.

Starting with Cg 2.2, cgGLRegisterStates calls cgSetStateLatestProfile for program states it creates and registers the latest profile returned by cgGLGetLatestProfile for the appropriate program domain.

SEE ALSO

cgCreateState, cgSetStateLatestProfile, cgSetPassState, cgResetPassState, cgCallStateValidateCallback, cgD3D9RegisterStates

5.56 cgGLSetContextGLSLVersion

NAME

cgGLSetContextGLSLVersion - set the current GLSL version for a context

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetContextGLSLVersion( CGcontext context,
                                CGGLglslversion version );

PARAMETERS

context
    The context in which the GLSL version will be changed.

version
    The GLSL version to be used by the runtime for context. See the glsl documentation for further details.

RETURN VALUES

None.

DESCRIPTION

cgGLSetContextGLSLVersion sets the current GLSL version for a context. This version will be sent as a profile option when compiling programs or effects for GLSL profiles.

If an environment variable named CGGL_GLSL_VERSION is set in the application’s environment to a string that translates to a valid, supported GLSL version then cgGLSetContextGLSLVersion is effectively a noop as the GLSL version specified by CGGL_GLSL_VERSION is always used instead of version and will always be returned by cgGLGetContextGLSLVersion. Valid values for CGGL_GLSL_VERSION are “100”, “110”, “120”, “1.00”, “1.10”, and “1.20”. If CGGL_GLSL_VERSION is set to any other value it is ignored.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_ENUMERANT_ERROR is generated if version is not CG_GL_GLSL_DEFAULT, CG_GL_GLSL_100, CG_GL_GLSL_110 or CG_GL_GLSL_120.

HISTORY

cgGLSetContextGLSLVersion was introduced in Cg 3.1.

SEE ALSO

 cgGLGetContextGLSLVersion, cgGLDetectGLSLVersion, cgGLGetContextOptimalOptions, cgGLSetContextOptimalOptions

5.57 cgGLSetContextOptimalOptions

NAME

cgGLSetContextOptimalOptions - set the implicit compiler optimization options for a profile

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetContextOptimalOptions( CGcontext context, CGprofile profile );

PARAMETERS

    context

        The context for which the optimal options will be set.

    profile

        The profile for which the optimal options will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetContextOptimalOptions sets implicit compiler arguments that are appended to the argument list passed to cgCreateProgram. The arguments are chosen based on the the available compiler arguments, GPU, and driver.

The arguments will be appended to the argument list every time cgCreateProgram is called until the CGcontext is destroyed.

cgGLSetContextOptimalOptions needs to be called for each CGcontext.

Note: cgGLSetContextOptimalOptions replaces cgGLSetContextOptimalOptions.

EXAMPLES

to-be-written
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

CG_INVALID_PROFILE_ERROR is generated if profile is not a supported OpenGL profile.

HISTORY

cgGLSetContextOptimalOptions was introduced in Cg 3.1.

SEE ALSO

cgGLGetContextOptimalOptions, cgCreateProgram

5.58  cgGLSetDebugMode

NAME

cgGLSetDebugMode - control whether the cgGL runtime calls glGetError

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetDebugMode( CGbool debug );

PARAMETERS

d debug

Flag indicating whether the library should use OpenGL error checking.

RETURN VALUES

None.

DESCRIPTION

The OpenGL Cg runtime calls glGetError at various points to verify that no errors have occurred. While this is helpful during development, the resulting performance penalty may be deemed too severe. cgGLSetDebugMode allows the application to turn off the OpenGL error checking if so desired.

EXAMPLES

cgGLSetDebugMode( CG_TRUE );  // Enables debug mode
cgGLSetDebugMode( CG_FALSE );  // Disables debug mode

ERRORS

None.

HISTORY

cgGLSetDebugMode was introduced in Cg 1.5.

SEE ALSO

cGetErrorHandler, cgGetError
5.59 cgGLSetManageTextureParameters

NAME

cgGLSetManageTextureParameters - set the manage texture parameters flag for a context

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLSetManageTextureParameters( CGcontext context,
                                       CGbool flag );
```

PARAMETERS

- **context**
  The context in which the automatic texture management behavior will be changed.

- **flag**
  A boolean switch which controls automatic texture management by the runtime.

RETURN VALUES

None.

DESCRIPTION

By default, cgGL does not manage any texture state in OpenGL. It is up to the user to enable and disable textures using `cgGLEnableTextureParameter` and `cgGLDisableTextureParameter` respectively. This behavior is the default in order to avoid conflicts with texture state on geometry that’s rendered with the fixed function pipeline or without cgGL.

If automatic texture management is desired, `cgGLSetManageTextureParameters` may be called with **flag** set to **CG_TRUE** before `cgGLBindProgram` is called. Whenever `cgGLBindProgram` is called, the cgGL runtime will make all the appropriate texture parameter calls on the application’s behalf.

`cgGLUnbindProgram` may be used to reset the texture state

Calling `cgGLSetManageTextureParameters` with **flag** set to **CG_FALSE** will disable automatic texture management.

NOTE: When `cgGLSetManageTextureParameters` is set to **CG_TRUE**, applications should not make texture state change calls to OpenGL (such as `glBindTexture`, `glActiveTexture`, etc.) after calling `cgGLBindProgram`, unless the application is trying to override some parts of cgGL’s texture management.

EXAMPLES

*to-be-written*

ERRORS

None.

HISTORY

`cgGLSetManageTextureParameters` was introduced in Cg 1.2.

SEE ALSO

`cgGLGetManageTextureParameters`, `cgGLBindProgram`, `cgGLUnbindProgram`
5.60  cgGLSetMatrixParameterArraydc

NAME

cgGLSetMatrixParameterArraydc - set the values of a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetMatrixParameterArraydc( CGparameter param,
               long offset,
               long nelements,
               const double * v );

PARAMETERS

param
The matrix array parameter that will be set.

offset
An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array of values to which to set the parameter. This must be a contiguous set of values with size nelements times the number of elements in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGLSetMatrixParameterArraydc sets the value of a matrix array parameter from an array of doubles laid out in column-major order.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.
HISTORY
cgGLSetMatrixParameterArraydc was introduced in Cg 1.1.

SEE ALSO

cgGLSetMatrixParameter, cgGLGetMatrixParameterArray

5.61 cgGLSetMatrixParameterArraydr

NAME
cgGLSetMatrixParameterArraydr - set the values of a matrix array parameter

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLSetMatrixParameterArraydr( CGparameter param,
                                   long offset,
                                   long nelements,
                                   const double * v );
```

PARAMETERS

- param
  - The matrix array parameter that will be set.
- offset
  - An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.
- nelements
  - The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of `offset`.
- v
  - The array of values to which to set the parameter. This must be a contiguous set of values with size `nelements` times the number of elements in the matrix.

RETURN VALUES

None.

DESCRIPTION

cgGLSetMatrixParameterArraydr sets the value of a matrix array parameter from an array of doubles laid out in row-major order.

EXAMPLES

to-be-written
**ERRORS**

- **CG_INVALID_PROFILE_ERROR** is generated if `param`’s profile is not a supported OpenGL profile.
- **CG_ARRAY_PARAM_ERROR** is generated if `param` is not an array parameter.
- **CG_NOT_MATRIX_PARAM_ERROR** is generated if the elements of `param` are not matrix parameters.
- **CG_OUT_OF_ARRAY_BOUNDS_ERROR** is generated if `offset` or `nelements` is outside the bounds of `param`.
- **CG_INVALID_PARAM_HANDLE_ERROR** is generated if `param` is not a valid parameter.
- **CG_INVALID_PARAMETER_ERROR** is generated if the parameter fails to set for any other reason.

**HISTORY**

cgGLSetMatrixParameterArraydr was introduced in Cg 1.1.

**SEE ALSO**

cgGLSetMatrixParameter, cgGLGetMatrixParameterArray

## 5.62 cgGLSetMatrixParameterArrayfc

**NAME**

cgGLSetMatrixParameterArrayfc - set the values of a matrix array parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

void cgGLSetMatrixParameterArrayfc( CGparameter param,
         long offset,
         long nelements,
         const float * v );
```

**PARAMETERS**

- **param**
  The matrix array parameter that will be set.
- **offset**
  An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.
- **nelements**
  The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of `offset`.
- **v**
  The array of values to which to set the parameter. This must be a contiguous set of values with size `nelements` times the number of elements in the matrix.

**RETURN VALUES**

None.
DESCRIPTION

cgGLSetMatrixParameterArrayfc sets the value of a matrix array parameter from an array of floats laid out in column-major order.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetMatrixParameterArrayfc was introduced in Cg 1.1.

SEE ALSO

cgGLSetMatrixParameter, cgGLGetMatrixParameterArray

5.63 cgGLSetMatrixParameterArrayfr

NAME

cgGLSetMatrixParameterArrayfr - set the values of a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetMatrixParameterArrayfr( CGparameter param,
                long offset,
                long nelements,
                const float * v );

PARAMETERS

param

The matrix array parameter that will be set.

offset

An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements

The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.
The array of values to which to set the parameter. This must be a contiguous set of values with size `nelements` times the number of elements in the matrix.

**RETURN VALUES**

None.

**DESCRIPTION**

`cgGLSetMatrixParameterArrayf` sets the value of a matrix array parameter from an array of `floats` laid out in row-major order.

**EXAMPLES**

`to-be-written`

**ERRORS**

- `CG_INVALID_PROFILE_ERROR` is generated if `param`’s profile is not a supported OpenGL profile.
- `CG_ARRAY_PARAM_ERROR` is generated if `param` is not an array parameter.
- `CG_NOT_MATRIX_PARAM_ERROR` is generated if the elements of `param` are not matrix parameters.
- `CG_OUT_OF_ARRAY_BOUNDS_ERROR` is generated if `offset` or `nelements` is outside the bounds of `param`.
- `CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.
- `CG_INVALID_PARAMETER_ERROR` is generated if the parameter fails to set for any other reason.

**HISTORY**

`cgGLSetMatrixParameterArrayf` was introduced in Cg 1.1.

**SEE ALSO**

`cgGLSetMatrixParameter`, `cgGLGetMatrixParameterArray`

### 5.64 cgGLSetMatrixParameterArray

**NAME**

`cgGLSetMatrixParameterArray` - set the value of an array matrix parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLSetMatrixParameterArray{fd}{rc}( CGparameter param,
                                         long offset,
                                         long nelements,
                                         const TYPE * v );
```

5.64. cgGLSetMatrixParameterArray 549
PARAMETERS

param
The matrix array parameter that will be set.

offset
An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array of values to which to set the parameter. This must be a contiguous set of values with size nelements times the number of elements in the matrix.

RETURN VALUES

None.

DESCRIPTION

The cgGLSetMatrixParameterArray functions set the value of a scalar or vector array parameter.

There are versions of the function that take either float or double values signified by f or d in the function name.

There are versions of the function that assume the array of values are laid out in either row or column order signified by r or c in the function name respectively.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_NOT_MATRIX_PARAM_ERROR is generated if the elements of param are not matrix parameters.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

The cgGLSetMatrixParameterArray functions were introduced in Cg 1.1.

SEE ALSO

cgGLSetMatrixParameter, cgGLGetMatrixParameterArray
5.65  cgGLSetMatrixParameterdc

NAME

cgGLSetMatrixParameterdc - set the values of a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetMatrixParameterdc( CGparameter param,
                                 const double * matrix );

PARAMETERS

    param
    The matrix parameter that will be set.

    matrix
    An array of values used to set the matrix parameter. The array must be the number of rows times the
    number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgGLSetMatrixParameterdc sets the value of a matrix parameter from an array of doubles laid out in column-major
order.

cgGLSetMatrixParameterdc functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetMatrixParameterdc was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetParameter
5.66 cgGLSetMatrixParameterdr

NAME

cgGLSetMatrixParameterdr - set the values of a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetMatrixParameterdr( CGparameter param,
                               const double *matrix );

PARAMETERS

param

The matrix parameter that will be set.

matrix

An array of values used to set the matrix parameter. The array must be the number of rows times the
number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgGLSetMatrixParameterdr sets the value of a matrix parameter from an array of doubles laid out in row-major
order.

cgGLSetMatrixParameterdr functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetMatrixParameterdr was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetParameter
5.67 cgGLSetMatrixParameterfc

NAME
cgGLSetMatrixParameterfc - set the values of a matrix array parameter

SYNOPSIS
#include <Cg/cgGL.h>

void cgGLSetMatrixParameterfc( CGparameter param,
                              const float * matrix );

PARAMETERS

param
The matrix parameter that will be set.

matrix
An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES
None.

DESCRIPTION
cgGLSetMatrixParameterfc sets the value of a matrix parameter from an array of floats laid out in column-major order.
cgGLSetMatrixParameterfc functions may only be called with uniform parameters.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetMatrixParameterfc was introduced in Cg 1.1.

SEE ALSO
cgGLGetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetParameter
5.68 cgGLSetMatrixParameterfr

NAME
cgGLSetMatrixParameterfr - set the values of a matrix array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetMatrixParameterfr( CGparameter param,
                                const float * matrix );

PARAMETERS

param

The matrix parameter that will be set.

matrix

An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES

None.

DESCRIPTION

cgGLSetMatrixParameterfr sets the value of a matrix parameter from an array of floats laid out in row-major order. cgGLSetMatrixParameterfr functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetMatrixParameterfr was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetParameter

5.69 cgGLSetMatrixParameter

NAME
cgGLSetMatrixParameter - set the value of a matrix parameter
SYNOPSIS

#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLSetMatrixParameter{fd}{rc}( CGparameter param,
    const TYPE * matrix );

PARAMETERS

param

The matrix parameter that will be set.

matrix

An array of values used to set the matrix parameter. The array must be the number of rows times the number of columns in size.

RETURN VALUES

None.

DESCRIPTION

The cgGLSetMatrixParameter functions set the value of a matrix parameter.

There are versions of the function that take either float or double values signified by f or d in the function name.

There are versions of the function that assume the array of values are laid out in either row or column order signified by r or c in the function name respectively.

The cgGLSetMatrixParameter functions may only be called with uniform parameters.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_NOT_MATRIX_PARAM_ERROR is generated if param is not a matrix parameter.

CG_INVALID_POINTER_ERROR is generated if matrix is NULL.

CG_INVALID_PARAMETER_ERROR is generated if the operation fails for any other reason.

HISTORY

The cgGLSetMatrixParameter functions were introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetParameter
5.70 cgGLSetOptimalOptions

NAME

cgGLSetOptimalOptions - deprecated, use cgGLSetContextOptimalOptions

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetOptimalOptions( CGprofile profile );

PARAMETERS

profile

The profile for which the optimal options will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetOptimalOptions sets implicit compiler arguments that are appended to the argument list passed to cgCreateProgram. The arguments are chosen based on the the available compiler arguments, GPU, and driver.
The arguments will be appended to the argument list every time cgCreateProgram is called until the last CGcontext is destroyed, after which this function should be called again.

ERRORS

CG_INVALID_PROFILE_ERROR is generated if profile is not a supported OpenGL profile.

HISTORY

cgGLSetOptimalOptions was introduced in Cg 1.1.
cgGLSetOptimalOptions was deprecated in Cg 3.1, use cgGLSetContextOptimalOptions.

SEE ALSO

cgGLSetContextOptimalOptions

5.71 cgGLSetParameter1d

NAME

cgGLSetParameter1d - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter1d( CQparameter param,
                          double x );
PARAMETERS

param
The parameter that will be set.

x
The value to which param will be set.

RETURN VALUES
None.

DESCRIPTION
cgGLSetParameter1d sets the value of a scalar or vector parameter.
cgGLSetParameter1d may be called with uniform or varying parameters. When called with a varying parameter,
the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only
work with uniform parameters.

EXAMPLES

double x = 1.0;
cgGLSetParameter1d(param, x);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameter1d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSet-
TextureParameter, cgGLBindProgram

5.72 cgGLSetParameter1dv

NAME
cgGLSetParameter1dv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter1dv( CGparameter param,
             const double * v );
PARAMETERS

param

The parameter that will be set.

v

Array of values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter1dv sets the values of a scalar or vector parameter from the given array of values.

cgGLSetParameter1dv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

double x = 1.0;

cgGLSetParameter1dv(param, &x);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter1dv was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.73 cgGLSetParameter1f

NAME

cgGLSetParameter1f - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter1f( CGparameter param,
              float x );
PARAMETERS

param

The parameter that will be set.

x

The value to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter1f sets the value of a scalar or vector parameter.

cgGLSetParameter1f may be called with uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

float x = 1.0f;

cgGLSetParameter1f(param, x);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter1f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.74 cgGLSetParameter1fv

NAME

cgGLSetParameter1fv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter1fv( CGparameter param,
                        const float * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values used to set param.

RETURN VALUES
None.

DESCRIPTION

cgGLSetParameter1fv sets the values of a scalar or vector parameter from the given array of values.
cgGLSetParameter1fv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

float x = 1.0f;
cgGLSetParameter1fv(param, &x);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameter1fv was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.75 cgGLSetParameter2d

NAME
cgGLSetParameter2d - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter2d( CGparameter param,
                      double x,
                      double y );
PARAMETERS

param
    The parameter that will be set.
x, y
    The values to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter2d sets the value of a scalar or vector parameter.
If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter2d may be called with uniform or varying parameters. When called with a varying parameter,
the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only
work with uniform parameters.

EXAMPLES

cgGLSetParameter2d(param, 1.0, 2.0);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter2d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSet-
TextureParameter, cgGLBindProgram

5.76 cgGLSetParameter2dv

NAME

cgGLSetParameter2dv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter2dv( CGparameter param,
                const double * v );
PARAMETERS

*param*

The parameter that will be set.

*v*

Array of values used to set *param*.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter2dv sets the values of a scalar or vector parameter from the given array of values. If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter2dv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the `cgGLGetParameter` functions only work with uniform parameters.

EXAMPLES

```c
double v[] = {1.0, 2.0};
cgGLSetParameter2dv(param, v);
```

ERRORS

`CG_INVALID_PROFILE_ERROR` is generated if *param*’s profile is not a supported OpenGL profile.

`CG_INVALID_PARAM_HANDLE_ERROR` is generated if *param* is not a valid parameter.

`CG_INVALID_PARAMETER_ERROR` is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter2dv was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.77 cgGLSetParameter2f

NAME

*cgGLSetParameter2f* - set the values of a scalar or vector parameter

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLSetParameter2f( CGparameter param,
                        float x,
                        float y );
```
PARAMETERS

param
The parameter that will be set.

x, y
The values to which param will be set.

RETURN VALUES
None.

DESCRIPTION

cgGLSetParameter2f sets the value of a scalar or vector parameter.
If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter2f may be called with uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

cgGLSetParameter2f(param, 1.0f, 2.0f);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

.cgGLSetParameter2f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.78 cgGLSetParameter2fv

NAME

cgGLSetParameter2fv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter2fv( CGparameter param,
                          const float * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgGLSetParameter2fv sets the values of a scalar or vector parameter from the given array of values.
If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter2fv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

float v[] = {1.0f, 2.0f};
cgGLSetParameter2fv(param, &v);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameter2fv was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.79 cgGLSetParameter3d

NAME
cgGLSetParameter3d - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter3d( CGparameter param,
                       double x,
                       double y,
                       double z );
PARAMETERS

param

The parameter that will be set.

x, y, z

The values to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter3d sets the value of a scalar or vector parameter.

If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter3d may be called with uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

double x, y, z;

// assign appropriate values to x, y, & z, then...
cgGLSetParameter3d(param, x, y, z);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter3d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.80 cgGLSetParameter3dv

NAME

cgGLSetParameter3dv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter3dv( CGparameter param,
                        const double * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgGLSetParameter3dv sets the values of a scalar or vector parameter from the given array of values.
If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter3dv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

double v[] = {1.0, 2.0, 3.0};
cgLSetParameter3dv(param, &v);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgLSetParameter3dv was introduced in Cg 1.1.

SEE ALSO
cgLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.81 cgGLSetParameter3f

NAME
cgLSetParameter3f - set the values of a scalar or vector parameter

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLSetParameter3f( CGparameter param,
    float x,
    float y,
    float z );
```
PARAMETERS

    param
    The parameter that will be set.
    x, y, z
    The values to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter3f sets the value of a scalar or vector parameter.
If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter3f may be called with uniform or varying parameters. When called with a varying parameter,
the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only
work with uniform parameters.

EXAMPLES

    float x, y, z;
    // assign appropriate values to x, y, & z, then...
    cgGLSetParameter3f(param, x, y, z);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter3f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSet-
TextureParameter, cgGLBindProgram

5.82  cgGLSetParameter3fv

NAME

cgGLSetParameter3fv - set the values of a scalar or vector parameter

SYNOPSIS

    #include <Cg/cgGL.h>

    void cgGLSetParameter3fv( CGparameter param,
                              const float * v );
PARAMETERS

param
The parameter that will be set.

v
Array of values used to set param.

RETURN VALUES
None.

DESCRIPTION
cgGLSetParameter3fv sets the values of a scalar or vector parameter from the given array of values. If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter3fv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

float v[] = {1.0f, 2.0f, 3.0f};
cgGLSetParameter3fv(param, &v);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameter3fv was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.83 cgGLSetParameter4d

NAME
cgGLSetParameter4d - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter4d( CGparameter param,
                         double x,
                         double y,
                         double z,
                         double w);
double z,
    double w);

PARAMETERS

    param
    The parameter that will be set.
    x, y, z, w
    The values to which param will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter4d sets the value of a scalar or vector parameter.

If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter4d may be called with uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

    double x, y, z, w;

    // assign appropriate values to x, y, & z, then...

    cgGLSetParameter4d(param, x, y, z, w);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter4d was introduced in Cg 1.1.

SEE ALSO

    cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

5.84 cgGLSetParameter4dv

NAME

cgGLSetParameter4dv - set the values of a scalar or vector parameter
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter4dv( CGparameter param,
                           const double * v );

PARAMETERS

   param
   The parameter that will be set.

   v
   Array of values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter4dv sets the values of a scalar or vector parameter from the given array of values.
If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter4dv may be called with either uniform or varying parameters. When called with a varying param-
eter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions
only work with uniform parameters.

EXAMPLES

double v[] = {1.0, 2.0, 3.0, 4.0};
cgGLSetParameter4dv(param, &v);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter4dv was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSet-
TextureParameter, cgGLBindProgram

5.85 cgGLSetParameter4f

NAME

cgGLSetParameter4f - set the values of a scalar or vector parameter
SYNOPSIS
#include <Cg/cgGL.h>

void cgGLSetParameter4f( CGparameter param,
    float x,
    float y,
    float z,
    float w );

PARAMETERS
param
    The parameter that will be set.

x, y, z, w
    The values to which param will be set.

RETURN VALUES
None.

DESCRIPTION
cgGLSetParameter4f sets the value of a scalar or vector parameter.
If more values are passed in than the parameter requires, the extra values will be ignored.
cgGLSetParameter4f may be called with uniform or varying parameters. When called with a varying parameter,
the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only
work with uniform parameters.

EXAMPLES
float x, y, z, w;
// assign appropriate values to x, y, & z, then...
cgGLSetParameter4f(param, x, y, z, w);

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameter4f was introduced in Cg 1.1.

SEE ALSO
cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram
5.86 cgGLSetParameter4fv

NAME

cgGLSetParameter4fv - set the values of a scalar or vector parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameter4fv( CGparameter param,
                         const float * v );

PARAMETERS

    param
    The parameter that will be set.

    v
    Array of values used to set param.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameter4fv sets the values of a scalar or vector parameter from the given array of values.

If more values are passed in than the parameter requires, the extra values will be ignored.

cgGLSetParameter4fv may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work with uniform parameters.

EXAMPLES

    float v[] = {1.0f, 2.0f, 3.0f, 4.0f};
    cgGLSetParameter4fv(param, &v);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter4fv was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram
5.87 cgGLSetParameterArray1d

NAME
cgGLSetParameterArray1d - set the values of an array parameter

SYNOPSIS
#include <Cg/cgGL.h>

void cgGLSetParameterArray1d( CGparameter param,
    long offset,
    long nelements,
    const double * v );

PARAMETERS

    param
    The array parameter that will be set.

    offset
    An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first
    element of the array.

    nelements
    The number of elements to set. A value of 0 will default to the total number of elements in the array minus
    the value of offset.

    v
    The array of values used to set the parameter. This must be a contiguous set of nelements values.

RETURN VALUES

None.

DESCRIPTION
cgGLSetParameterArray1d sets 1 value per element of a scalar or vector array parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameterArray1d was introduced in Cg 1.1.
SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.88 cgGLSetParameterArray1f

NAME
cgGLSetParameterArray1f - set the values of an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameterArray1f( CGparameter param,
                             long offset,
                             long nelements,
                             const float * v );

PARAMETERS

param
The array parameter that will be set.

offset
An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array of values used to set the parameter. This must be a contiguous set of nelements values.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameterArray1f sets 1 value per element of a scalar or vector array parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.
HISTORY

cgGLSetParameterArray1f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.89 cgGLSetParameterArray2d

NAME

cgGLSetParameterArray2d - set the values of an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameterArray2d( CGparameter param,
                               long offset,
                               long nelements,
                               const double * v );

PARAMETERS

    param
    The array parameter that will be set.

    offset
    An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

    nelements
    The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

    v
    The array of values used to set the parameter. This must be a contiguous set of 2 * nelements values.

RETURN VALUES

    None.

DESCRIPTION

    cgGLSetParameterArray2d sets 2 values per element of a scalar or vector array parameter.

EXAMPLES

    to-be-written

ERRORS

    CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

    CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

    CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if \texttt{param} is not a valid parameter.

\texttt{CG_INVALID_PARAMETER_ERROR} is generated if the parameter fails to set for any other reason.

**HISTORY**

cgGLSetParameterArray2d was introduced in Cg 1.1.

**SEE ALSO**

cgGLSetParameter, cgGLGetParameterArray

### 5.90 cgGLSetParameterArray2f

**NAME**

cgGLSetParameterArray2f - set the values of an array parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

void cgGLSetParameterArray2f( CGparameter param,
                               long offset,
                               long nelements,
                               const float * v );
```

**PARAMETERS**

- **param**
  The array parameter that will be set.
- **offset**
  An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.
- **nelements**
  The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of \texttt{offset}.
- **v**
  The array of values used to set the parameter. This must be a contiguous set of \(2 \times \text{nelements}\) values.

**RETURN VALUES**

None.

**DESCRIPTION**

cgGLSetParameterArray2f sets 2 values per element of a scalar or vector array parameter.

**EXAMPLES**

to-be-written
ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameterArray2f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.91 cgGLSetParameterArray3d

NAME

cgGLSetParameterArray3d - set the values of an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameterArray3d( CGparameter param,
                                long offset,
                                long nelements,
                                const double* v );

PARAMETERS

param

The array parameter that will be set.

offset

An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements

The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v

The array of values used to set the parameter. This must be a contiguous set of 3 * nelements values.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameterArray3d sets 3 values per element of a scalar or vector array parameter.
EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.
CG.Array_PARAM_ERROR is generated if param is not an array parameter.
CG.OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameterArray3d was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.92 cgGLSetParameterArray3f

NAME

cgGLSetParameterArray3f - set the values of an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameterArray3f( CGparameter param,
                            long offset,
                            long nelements,
                            const float * v );

PARAMETERS

param

The array parameter that will be set.

offset

An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first
element of the array.

nelements

The number of elements to set. A value of 0 will default to the total number of elements in the array minus
the value of offset.

v

The array of values used to set the parameter. This must be a contiguous set of 3 * nelements values.

RETURN VALUES

None.
DESCRIPTION

cgGLSetParameterArray3f sets 3 values per element of a scalar or vector array parameter.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.

CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameterArray3f was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter, cgGLGetParameterArray

5.93 cgGLSetParameterArray4d

NAME

cgGLSetParameterArray4d - set the values of an array parameter

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetParameterArray4d( CGparameter param,
  long offset,
  long nelements,
  const double * v );

PARAMETERS

param

The array parameter that will be set.

offset

An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements

The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v

The array of values used to set the parameter. This must be a contiguous set of 4 * nelements values.
RETURN VALUES
None.

DESCRIPTION
cgGLSetParameterArray4d sets 4 values per element of a scalar or vector array parameter.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
cgGLSetParameterArray4d was introduced in Cg 1.1.

SEE ALSO
cgGLSetParameter, cgGLGetParameterArray

5.94 cgGLSetParameterArray4f

NAME
cgGLSetParameterArray4f - set the values of an array parameter

SYNOPSIS
#include <Cg/cgGL.h>

void cgGLSetParameterArray4f( CGparameter param,
                               long offset,
                               long nelements,
                               const float * v );

PARAMETERS
param
The array parameter that will be set.
offset
An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first
element of the array.
nelements
The number of elements to set. A value of 0 will default to the total number of elements in the array minus
the value of offset.
The array of values used to set the parameter. This must be a contiguous set of \( 4 \times \text{nelements} \) values.

**RETURN VALUES**

None.

**DESCRIPTION**

`cgGLSetParameterArray4f` sets 4 values per element of a scalar or vector array parameter.

**EXAMPLES**

`to-be-written`

**ERRORS**

- `CG_INVALID_PROFILE_ERROR` is generated if `param`’s profile is not a supported OpenGL profile.
- `CG_ARRAY_PARAM_ERROR` is generated if `param` is not an array parameter.
- `CG_OUT_OF_ARRAY_BOUNDS_ERROR` is generated if `offset` or `nelements` is outside the bounds of `param`.
- `CG_INVALID_PARAM_HANDLE_ERROR` is generated if `param` is not a valid parameter.
- `CG_INVALID_PARAMETER_ERROR` is generated if the parameter fails to set for any other reason.

**HISTORY**

`cgGLSetParameterArray4f` was introduced in Cg 1.1.

**SEE ALSO**

`cgGLSetParameter`, `cgGLGetParameterArray`

### 5.95 cgGLSetParameterArray

**NAME**

`cgGLSetParameterArray` - set the values of an array parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLSetParameterArray{1234}{fd}( CGparameter param, 
  long offset, 
  long nelements, 
  const TYPE * v );
```

**PARAMETERS**

- `param`
  - The array parameter that will be set.
- `offset`
- `nelements`
- `v`
An offset into the array parameter at which to start setting elements. A value of 0 will begin at the first element of the array.

nelements
The number of elements to set. A value of 0 will default to the total number of elements in the array minus the value of offset.

v
The array of values used to set the parameter. This must be a contiguous set of values that total nelements times the vector size indicated by the number in the function name.

RETURN VALUES
None.

DESCRIPTION
The cgGLSetParameterArray functions set the value of a scalar or vector array parameter. Either 1, 2, 3, or 4 values per array element will be set, depending on which function is used. There are versions of the function that take either float or double values signified by f or d in the function name.

EXAMPLES
to-be-written

ERRORS
CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.
CG_ARRAY_PARAM_ERROR is generated if param is not an array parameter.
CG_OUT_OF_ARRAY_BOUNDS_ERROR is generated if offset or nelements is outside the bounds of param.
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.
CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY
The cgGLSetParameterArray functions were introduced in Cg 1.1.

SEE ALSO
cgGLSetParameter, cgGLGetParameterArray

5.96 cgGLSetParameter

NAME
cgGLSetParameter - set the values of a scalar or vector parameter

SYNOPSIS
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLSetParameter1{fd}( CGparameter param,
void cgGLSetParameter2(fd)( CGparameter param,
    TYPE x,
    TYPE y );

void cgGLSetParameter3(fd)( CGparameter param,
    TYPE x,
    TYPE y,
    TYPE z );

void cgGLSetParameter4(fd)( CGparameter param,
    TYPE x,
    TYPE y,
    TYPE z,
    TYPE w );

void cgGLSetParameter{1234}{fd}v( CGparameter param,
    const TYPE * v );

PARAMETERS

param
The parameter that will be set.

x, y, z, and w
The values used to set the parameter.

v
An array of values used to set the parameter for the array versions of the set functions.

RETURN VALUES

None.

DESCRIPTION

The cgGLSetParameter functions set the value of a scalar or vector parameter.
The function takes either 1, 2, 3, or 4 values depending on which version is used. If more values are passed in than the parameter requires, the extra values will be ignored.

There are versions of each function that take either float or double values signified by f or d in the function name.
The functions with v at the end of their names take an array of values instead of explicit parameters.

The cgGLSetParameter functions may be called with either uniform or varying parameters. When called with a varying parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions will only work with uniform parameters.

Note: Previous releases of Cg allowed you to store more values in a parameter than indicated by the parameter’s type. For example, one could use cgGLSetParameter4f to store four values into a parameter of type CG_FLOAT (not CG_FLOAT4). All four values could later be retrieved using a get call which requested more than one value. However, this feature conflicts with the GLSL approach and also leads to issues with parameters mapped into BUFFERS. Therefore, beginning with Cg 2.0 any components beyond the number indicated by the parameter type are ignored.
EXAMPLES

```c
.cgGLSetParameter1f(param, x);
.cgGLSetParameter1d(param, x);
.cgGLSetParameter2f(param, x, y);
.cgGLSetParameter2d(param, x, y);
.cgGLSetParameter3f(param, x, y, z);
.cgGLSetParameter3d(param, x, y, z);
.cgGLSetParameter4f(param, x, y, z, w);
.cgGLSetParameter4d(param, x, y, z, w);
.cgGLSetParameter1fv(param, &v);
.cgGLSetParameter1dv(param, &v);
.cgGLSetParameter2fv(param, &v);
.cgGLSetParameter2dv(param, &v);
.cgGLSetParameter3fv(param, &v);
.cgGLSetParameter3dv(param, &v);
.cgGLSetParameter4fv(param, &v);
.cgGLSetParameter4dv(param, &v);
```

ERRORS

- **CG_INVALID_PROFILE_ERROR** is generated if `param`’s profile is not a supported OpenGL profile.
- **CG_INVALID_PARAM_HANDLE_ERROR** is generated if `param` is not a valid parameter.
- **CG_INVALID_PARAMETER_ERROR** is generated if the parameter fails to set for any other reason.

HISTORY

The `cgGLSetParameter` functions were introduced in Cg 1.1.

SEE ALSO

- `cgGLGetParameter`, `cgGLSetParameterArray`, `cgGLSetMatrixParameter`, `cgGLSetMatrixParameterArray`, `cgGLSetTextureParameter`, `cgGLBindProgram`

5.97 **cgGLSetParameterPointer**

NAME

`cgGLSetParameterPointer` - sets a varying parameter with an attribute array

SYNOPSIS

```c
#include <Cg/cgGL.h>

void cgGLSetParameterPointer( CGparameter param, 
    GLint fsize, 
    GLenum type, 
    GLsizei stride, 
    const GLvoid * pointer);
```
PARAMETERS

param
The parameter that will be set.

fsize
The number of coordinates per vertex.

type
The data type of each coordinate. Possible values are GL_UNSIGNED_BYTE, GL_SHORT, GL_INT, GL_FLOAT, and GL_DOUBLE.

stride
The byte offset between consecutive vertices. When stride is 0 the array is assumed to be tightly packed.

pointer
The pointer to the first coordinate in the vertex array.

RETURN VALUES

None.

DESCRIPTION

cgGLSetParameterPointer sets a varying parameter to a given vertex array in the typical OpenGL style. See the OpenGL documentation on the various vertex array functions (e.g. glVertexPointer, glNormalPointer, etc...) for more information.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param's profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_UNSUPPORTED_GL_EXTENSION_ERROR is generated if param required an OpenGL extension that is not available.

CG_INVALID_PARAMETER_ERROR is generated if the parameter fails to set for any other reason.

HISTORY

cgGLSetParameterPointer was introduced in Cg 1.1.

SEE ALSO

cgGLSetParameter

cgGLSetStateMatrixParameter

5.98 cgGLSetStateMatrixParameter

NAME

cgGLSetStateMatrixParameter - set the values of a matrix parameter to a matrix in the OpenGL state
SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetStateMatrixParameter( CGparameter param,
                                   CGGLenum matrix,
                                   CGGLenum transform );

PARAMETERS

param
The matrix parameter that will be set.

matrix
An enumerant indicating which matrix should be retrieved from the OpenGL state. Must be one of the following:

* CG_GL_MODELVIEW_MATRIX
* CG_GL_PROJECTION_MATRIX
* CG_GL_TEXTURE_MATRIX
* CG_GL_MODELVIEW_PROJECTION_MATRIX

transform
An enumerant indicating an optional transformation that may be applied to the matrix before it is set. Must be one of the following:

* CG_GL_MATRIX_IDENTITY
* CG_GL_MATRIX_TRANSPOSE
* CG_GL_MATRIX_INVERSE
* CG_GL_MATRIX_INVERSE_TRANSPOSE

RETURN VALUES

None.

DESCRIPTION
cgGLSetStateMatrixParameter sets a matrix parameter to the values retrieved from an OpenGL state matrix. The state matrix to retrieve is indicated by matrix, which may be one of the following:

* CG_GL_MODELVIEW_MATRIX
  Get the current modelview matrix.
* CG_GL_PROJECTION_MATRIX
  Get the current projection matrix.
* CG_GL_TEXTURE_MATRIX
  Get the current texture matrix.
* CG_GL_MODELVIEW_PROJECTION_MATRIX
  Get the concatenated modelview and projection matrices.

The transform parameter specifies an optional transformation which will be applied to the retrieved matrix before setting the values in the parameter. transform must be one of the following:
* **CG_GL_MATRIX_IDENTITY**
  Don’t apply any transform, leaving the matrix as is.

* **CG_GL_MATRIX_TRANSPOSE**
  Transpose the matrix.

* **CG_GL_MATRIX_INVERSE**
  Invert the matrix.

* **CG_GL_MATRIX_INVERSE_TRANSPOSE**
  Transpose and invert the matrix.

**cgGLSetStateMatrixParameter** may only be called with a uniform matrix parameter. If the size of the matrix is less than 4x4, the upper left corner of the matrix that fits into the given matrix parameter will be returned.

**EXAMPLES**

* to-be-written

**ERRORS**

**CG_INVALID_PROFILE_ERROR** is generated if **param**’s profile is not a supported OpenGL profile.

**CG_NOT_MATRIX_PARAM_ERROR** is generated if **param** is not a matrix parameter.

**CG_INVALID_ENUMERANT_ERROR** is generated if either **matrix** or **transform** is not one of the allowed enumerant values.

**CG_INVALID_PARAM_HANDLE_ERROR** is generated if **param** is not a valid parameter.

**CG_INVALID_PARAMETER_ERROR** is generated if the parameter fails to set for any other reason.

**HISTORY**

**cgGLSetStateMatrixParameter** was introduced in Cg 1.1.

**SEE ALSO**

**cgGLSetMatrixParameter**, **cgGLGetMatrixParameter**

### 5.99 cgGLSetTextureParameter

**NAME**

**cgGLSetTextureParameter** - sets the value of a texture parameter

**SYNOPSIS**

```c
#include <Cg/cgGL.h>

void cgGLSetTextureParameter( CGparameter param,
                               GLuint texobj );
```

---

5.99. cgGLSetTextureParameter 587
PARAMETERS

    param
    The texture parameter that will be set.
    texobj
    An OpenGL texture object name to which the parameter will be set.

RETURN VALUES

None.

DESCRIPTION

cgGLSetTextureParameter sets the value of a texture parameter to an OpenGL texture object.

Note that in order to use the texture, either cgGLEnableTextureParameter must be called after cgGLSetTextureParameter and before the geometry is drawn, or cgGLSetManageTextureParameters must be called with a value of CG_TRUE.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a texture parameter or if the parameter fails to set for any other reason.

HISTORY

cgGLSetTextureParameter was introduced in Cg 1.1.

SEE ALSO

cgGLGetTextureParameter, cgGLSetParameter

5.100 cgGLSetupSampler

NAME

cgGLSetupSampler - initializes a sampler’s state and texture object handle

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLSetupSampler( CGparameter param,
                      GLuint texobj );
PARAMETERS

param
The sampler parameter that will be set.
texobj
An OpenGL texture object name to which the parameter will be set.

RETURN VALUES

None.

DESCRIPTION
cgGLSetupSampler initializes a sampler; like cgGLSetTextureParameter, it informs the OpenGL Cg runtime which OpenGL texture object to associate with the sampler. Furthermore, if the sampler was defined in the source file with a sampler_state block that specifies sampler state, this sampler state is initialized for the given texture object.

Note that in order to use the texture, either cgGLEnableTextureParameter must be called after cgGLSetTextureParameter and before the geometry is drawn, or cgGLSetManageTextureParameters must be called with a value of CG_TRUE.

EXAMPLES
to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if param’s profile is not a supported OpenGL profile.

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter.

CG_INVALID_PARAMETER_ERROR is generated if param is not a texture parameter or if the parameter fails to set for any other reason.

HISTORY
cgGLSetupSampler was introduced in Cg 1.4.

SEE ALSO
cgGLSetTextureParameter, cgGLGetTextureParameter, cgGLSetManageTextureParameters

5.101 cgGLUnbindProgram

NAME
cgGLUnbindProgram - unbinds the program bound in a profile

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLUnbindProgram( CGprofile profile );
PARAMETERS

profile

The profile from which to unbind any bound program.

RETURN VALUES

None.

DESCRIPTION

cgGLUnbindProgram unbinds the program which is bound in the profile specified by profile. It also resets the texture state back to the state it was in at the point cgGLBindProgram was first called with a program in the given profile.

EXAMPLES

cgGLUnbindProgram(CG_PROFILE_ARBVP1);
cgGLUnbindProgram(CG_PROFILE_ARBFP1);

ERRORS

CG_INVALID_PROFILE_ERROR is generated if profile is not a supported OpenGL profile.

HISTORY

cgGLUnbindProgram was introduced in Cg 1.2.

SEE ALSO

cgGLSetManageTextureParameters, cgGLBindProgram

5.102 cgGLUnloadProgram

NAME

cgGLUnloadProgram - destroy the OpenGL shader object associated with a program

SYNOPSIS

#include <Cg/cgGL.h>

void cgGLUnloadProgram( CGprogram program );

PARAMETERS

program

The program for which to destroy the shader object. The CGprogram handle is still valid after this call.

RETURN VALUES

None.

DESCRIPTION

cgGLUnloadProgram destroys the OpenGL shader object for a program. This call does not destroy the CGprogram itself, only the associated GL shader object. Use cgDestroyProgram to free the CGprogram itself. Also note that freeing a CGprogram using the core runtime implicitly calls this routine to avoid resource leaks.
EXAMPLES

// prog is a CGprogram initialized elsewhere
...
cgGLUnloadProgram(prog);

CGbool loaded = cgGLIsProgramLoaded(prog); // loaded == CG_FALSE

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgGLUnloadProgram was introduced in Cg 2.1.

SEE ALSO

cgGLLoadProgram, cgGLIsProgramLoaded, cgDestroyProgram
6.1 cgD3D9BindProgram

NAME

cgD3D9BindProgram - activate a program with D3D

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9BindProgram( CGprogram program );

PARAMETERS

program

The program to activate with D3D.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9BindProgram activates a program with D3D. The program is activated using IDirect3DDevice9::SetVertexShader or IDirect3DDevice9::SetPixelShader depending on the program’s profile type.

D3D allows only one vertex shader and one pixel shader to be active at any given time, so activating a program of a given type implicitly deactivates any other program of a that type.

If parameter shadowing is enabled for program, this call will set the D3D state for all shadowed parameters associated with program. If a parameter associated with program has not been shadowed when this function is called, the D3D state associated with that parameter is not modified.

If parameter shadowing is disabled, only the D3D shader is activated, and no other D3D state is modified.

EXAMPLES

// vertexProg and pixelProg are CGprograms initialized elsewhere
// pDev is an IDirect3DDevice9 interface initialized elsewhere
...
HRESULT hr = cgD3D9BindProgram(vertexProg);
HRESULT hr2 = cgD3D9BindProgram(pixelProg);
// Draw a quad using the vertex and pixel shader
// A vertex and index buffer are set up elsewhere.
HRESULT hr3 = pDev->DrawIndexedPrimitve(D3DPT_TRIANGLELIST, 0, 4, 0, 2);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

HISTORY

cgD3D9BindProgram was introduced in Cg 1.1.

SEE ALSO

cgD3D9LoadProgram, cgD3D9EnableParameterShadowing, cgD3D9IsParameterShadowingEnabled, cgD3D9SetUniform, cgD3D9SetUniformMatrix, cgD3D9SetTextureParameter

6.2 cgD3D9EnableDebugTracing

NAME

cgD3D9EnableDebugTracing - enable or disable debug output

SYNOPSIS

#include <Cg/cgD3D9.h>

void cgD3D9EnableDebugTracing( CGbool enable );

PARAMETERS

enable

A boolean switch which controls debugging output by the library.

RETURN VALUES

None.

DESCRIPTION

cgD3D9EnableDebugTracing enables or disables debug output for an application when using the debug DLL.

If an error callback is registered, breakpoints can be set for Debug DLL debug traces by testing the result of cgGetError for cgD3D9DebugTrace. Breakpoints can be set for D3D errors by testing for cgD3D9Failed and using cgD3D9GetLastError to determine the particular D3D error that occurred.

EXAMPLES

cgD3D9EnableDebugTracing(CG_TRUE);
// use code to be debugged
...
cgD3D9EnableDebugTracing(CG_FALSE);
ERRORS
None.

HISTORY
cgD3D9EnableDebugTracing was introduced in Cg 1.1.

SEE ALSO
cgSetErrorCallback, cgGetError, cgD3D9GetLastError

### 6.3 cgD3D9EnableParameterShadowing

#### NAME
cgD3D9EnableParameterShadowing - enable or disable parameter shadowing for a program

#### SYNOPSIS

```c
#include <Cg/cgD3D9.h>

HRESULT cgD3D9EnableParameterShadowing( CGprogram program,
                                      CGbool enable );
```

#### PARAMETERS

- **program**
  - The program in which to set the parameter shadowing state.
- **enable**
  - A boolean switch which controls parameter shadowing for *program*.

#### RETURN VALUES

Returns **D3D_OK** if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

#### DESCRIPTION
cgD3D9EnableParameterShadowing enables or disables parameter shadowing for a program.

If parameter shadowing is enabled for a program, any call to set the value of a parameter for that program does not set any D3D state. Instead it merely shadows the value so it can be set during a subsequent call to **cgD3D9BindProgram**.

If parameter shadowing is disabled, these calls immediately sets the D3D state and do not shadow the value.

When using this call to disable parameter shadowing, all shadowed parameters for that program are immediately invalidated. No D3D calls are made, so any active program retains its current D3D state. However, subsequent calls to **cgD3D9BindProgram** for that program will not apply any shadowed state. Parameter shadowing for the program will continue to be disabled until explicitly enabled with another call to **cgD3D9EnableParameterShadowing**.

Parameter shadowing can also be specified during a call to **cgD3D9LoadProgram**.

#### EXAMPLES
// prog is a CGprogram initialized elsewhere

HRESULT hres = cgD3D9EnableParameterShadowing(prog, CG_FALSE);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.
CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

HISTORY

cgD3D9EnableParameterShadowing was introduced in Cg 1.1.

SEE ALSO

cgD3D9IsParameterShadowingEnabled, cgD3D9LoadProgram

6.4 cgD3D9GetDevice

NAME

cgD3D9GetDevice - retrieves the current D3D9 device associated with the runtime

SYNOPSIS

#include <Cg/cgD3D9.h>

IDirect3DDevice9 * cgD3D9GetDevice( void );

PARAMETERS

None.

RETURN VALUES

Returns the current D3D9 device associated with the runtime.

DESCRIPTION

cgD3D9GetDevice retrieves the current D3D9 device associated with the runtime. Note that the returned device pointer may be NULL.

EXAMPLES

IDirect3DDevice9 * curDevice = cgD3D9GetDevice();

ERRORS

None.

HISTORY

cgD3D9GetDevice was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetDevice
6.5 cgD3D9GetLastError

NAME

*cgD3D9GetLastError* - get the last D3D error that occurred

SYNOPSIS

```c
#include <Cg/cgD3D9.h>

HRESULT cgD3D9GetLastError();
```

PARAMETERS

None.

RETURN VALUES

Returns the last D3D error that occurred during an expanded interface function call.

Returns *D3D_OK* if no D3D error has occurred since the last call to *cgD3D9GetLastError*.

DESCRIPTION

*cgD3D9GetLastError* retrieves the last D3D error that occurred during an expanded interface function call. The last error is always cleared immediately after the call.

EXAMPLES

```c
HRESULT lastError = cgD3D9GetLastError();
```

ERRORS

None.

HISTORY

*cgD3D9GetLastError* was introduced in Cg 1.1.

SEE ALSO

*cgD3D9TranslateHRESULT*

6.6 cgD3D9GetLatestPixelProfile

NAME

*cgD3D9GetLatestPixelProfile* - get the latest supported pixel shader version

SYNOPSIS

```c
#include <Cg/cgD3D9.h>

CGprofile cgD3D9GetLatestPixelProfile();
```

PARAMETERS

None.
RETURN VALUES

Returns the latest pixel shader version supported by the current D3D device.

Returns \texttt{CG\_PROFILE\_UNKNOWN} if no D3D device is currently set.

DESCRIPTION

\texttt{cgD3D9GetLatestPixelProfile} retrieves the latest pixel shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 2.2, if the environment variable \texttt{CGD3D9\_LATEST\_PIXEL\_PROFILE} is set in the application’s environment to a string that \texttt{cgGetProfile} translates to a valid profile (meaning not \texttt{CG\_PROFILE\_UNKNOWN}), then it is this \texttt{CG\_profile} value that will be returned by \texttt{cgD3D9GetLatestPixelProfile}.

EXAMPLES

\begin{verbatim}
CG\_profile bestPixelProfile = cgD3D9GetLatestPixelProfile();
\end{verbatim}

ERRORS

None.

HISTORY

\texttt{cgD3D9GetLatestPixelProfile} was introduced in Cg 1.1.

SEE ALSO

\texttt{cgD3D9GetLatestVertexProfile}

6.7 \texttt{cgD3D9GetLatestVertexProfile}

NAME

\texttt{cgD3D9GetLatestVertexProfile} - get the latest supported vertex shader version

SYNOPSIS

\begin{verbatim}
#include \textless\texttt{Cg/cgD3D9.h}\textgreater

CG\_profile cgD3D9GetLatestVertexProfile( \texttt{void} );
\end{verbatim}

PARAMETERS

None.

RETURN VALUES

Returns the latest vertex shader version supported by the current D3D device.

Returns \texttt{CG\_PROFILE\_UNKNOWN} if no D3D device is currently set.

DESCRIPTION

\texttt{cgD3D9GetLatestVertexProfile} retrieves the latest vertex shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.
Starting in Cg 2.2, if the environment variable `CGD3D9_LATEST_VERTEX_PROFILE` is set in the application’s environment to a string that `cgGetProfile` translates to a valid profile (meaning not `CG_PROFILE_UNKNOWN`), then it is this `CGprofile` value that will be returned by `cgD3D9GetLatestVertexProfile`.

**EXAMPLES**

```c
CGprofile bestVertexProfile = cgD3D9GetLatestVertexProfile();
```

**ERRORS**

None.

**HISTORY**

`cgD3D9GetLatestVertexProfile` was introduced in Cg 1.1.

**SEE ALSO**

`cgD3D9GetLatestPixelProfile`

## 6.8 cgD3D9GetManageTextureParameters

**NAME**

`cgD3D9GetManageTextureParameters` - get the manage texture parameters flag from a context

**SYNOPSIS**

```c
#include <Cg/cgD3D9.h>

CGbool cgD3D9GetManageTextureParameters( CGcontext context );
```

**PARAMETERS**

- context
  
The context from which the automatic texture management setting will be retrieved.

**RETURN VALUES**

Returns the manage texture management flag from `context`.

**DESCRIPTION**

`cgD3D9GetManageTextureParameters` returns the manage texture management flag from context. See `cgD3D9SetManageTextureParameters` for more information.

**EXAMPLES**

```c
CGbool manage = cgD3D9GetManageTextureParameters( pCtx );
if ( manage )
    doSomething();
```

**ERRORS**

None.

**HISTORY**

`cgD3D9GetManageTextureParameters` was introduced in Cg 1.5.
SEE ALSO

cgD3D9SetManageTextureParameters

6.9  cgD3D9GetOptimalOptions

NAME

cgD3D9GetOptimalOptions - get the best set of compiler options for a profile

SYNOPSIS

#include <Cg/cgD3D9.h>

char const ** cgD3D9GetOptimalOptions( CGprofile profile );

PARAMETERS

profile

The profile whose optimal arguments are requested.

RETURN VALUES

Returns a null-terminated array of strings representing the optimal set of compiler options for profile.

Returns NULL if no D3D device is currently set.

DESCRIPTION

cgD3D9GetOptimalOptions returns the best set of compiler options for a given profile. This is an expanded interface function because it needs to know about the D3D device to determine the most optimal options.

The elements of the returned array are meant to be used as part of the args parameter to cgCreateProgram or cgCreateProgramFromFile.

The returned string does not need to be destroyed by the application. However, the contents could change if the function is called again for the same profile but a different D3D device.

EXAMPLES

const char * vertOptions[] = { myCustomArgs,
   cgD3D9GetOptimalOptions(vertProfile),
   NULL };

// create the vertex shader
CGprogram myVS = cgCreateProgramFromFile( context,
   CG_SOURCE,
   "vshader.cg",
   vertProfile,
   "VertexShader",
   vertOptions);

ERRORS

None.

HISTORY

cgD3D9GetOptimalOptions was introduced in Cg 1.1.
SEE ALSO

cgD3D9GetLatestVertexProfile, cgD3D9GetLatestPixelProfile, cgCreateProgram, cgCreateProgramFromFile

6.10 cgD3D9GetTextureParameter

NAME
cgD3D9GetTextureParameter - get the value of a texture parameter

SYNOPSIS

#include <Cg/cgD3D9.h>

IDirect3DBaseTexture9 * cgD3D9GetTextureParameter( CGparameter param );

PARAMETERS

param
The texture parameter for which the D3D texture object will be retrieved.

RETURN VALUES

Returns a pointer to the D3D texture to which param was set. Return NULL if param has not been set.

DESCRIPTION

cgD3D9GetTextureParameter returns the D3D texture pointer to which a texture parameter was set using cgD3D9SetTextureParameter. If the parameter has not been set, the NULL will be returned.

EXAMPLES

// param is a texture parameter defined elsewhere...

HRESULT hr = cgD3D9SetTexture( param, cgD3D9GetTextureParameter( param ) );

ERRORS

None.

HISTORY

cgD3D9GetTextureParameter was introduced in Cg 1.5.

SEE ALSO

cgD3D9SetTextureParameter

6.11 cgD3D9GetVertexDeclaration

NAME

cgD3D9GetVertexDeclaration - get the default vertex declaration stream

6.10 cgD3D9GetTextureParameter
SYNOPSIS
#include <Cg/cgD3D9.h>

CGbool cgD3D9GetVertexDeclaration( CGprogram program,
                                    D3DVERTEXELEMENT9 decl[MAXD3DECLLENGTH] );

PARAMETERS
    program
        The program from which to retrieve the vertex declaration.
    decl
        A D3DVERTEXELEMENT9 array that will be filled with the D3D9 vertex declaration.

RETURN VALUES
Returns CG_TRUE on success.
Returns CG_FALSE otherwise.

DESCRIPTION
cgD3D9GetVertexDeclaration retrieves the default vertex declaration stream for a program. The declaration always
uses a tightly packed single stream. The stream is always terminated with D3DECL_END(), so this can be used to
determine the actual length of the returned declaration.

The default vertex declaration is always a single stream. There will be one D3DVERTEXELEMENT9 element for
each varying input parameter.

If you want to use a custom vertex declaration, you can test that declaration for compatibility by calling
cgD3D9ValidateVertexDeclaration.

EXAMPLES
For example:

    void main( in float4 pos : POSITION,
                in float4 dif : COLOR0,
                in float4 tex : TEXCOORD0,
                out float4 hpos : POSITION );

would have this default vertex declaration:

const D3DVERTEXELEMENT9 decl[] = {
    { 0, 0, D3DECLTYPE_FLOAT4, D3DECLMETHOD_DEFAULT, D3DECLUSAGE_POSITION, 0 },
    { 0, 16, D3DECLTYPE_FLOAT4, D3DECLMETHOD_DEFAULT, D3DECLUSAGE_COLOR, 0 },
    { 0, 32, D3DECLTYPE_FLOAT4, D3DECLMETHOD_DEFAULT, D3DECLUSAGE_TEXCOORD, 0 },
    D3DECL_END()
};

ERRORS
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY
cgD3D9GetVertexDeclaration was introduced in Cg 1.1.
SEE ALSO

cgD3D9ValidateVertexDeclaration

6.12 cgD3D9IsParameterShadowingEnabled

NAME
cgD3D9IsParameterShadowingEnabled - determine if parameter shadowing is enabled

SYNOPSIS

#include <Cg/cgD3D9.h>

CGbool cgD3D9IsParameterShadowingEnabled( CGprogram program );

PARAMETERS

program

The program to check for parameter shadowing.

RETURN VALUES

Returns CG_TRUE if parameter shadowing is enabled for program.

Returns CG_FALSE otherwise.

DESCRIPTION

cgD3D9IsParameterShadowingEnabled determines if parameter shadowing is enabled for program.

EXAMPLES

// program is a CGprogram initialized elsewhere
... 
CGbool isShadowing = cgD3D9IsParameterShadowingEnabled(program);

ERRORS

None.

HISTORY

cgD3D9IsParameterShadowingEnabled was introduced in Cg 1.1.

SEE ALSO

cgD3D9EnableParameterShadowing, cgD3D9LoadProgram

6.13 cgD3D9IsProfileSupported

NAME

cgD3D9IsProfileSupported - determine if a profile is supported by cgD3D9
SYNOPSIS

#include <Cg/cgD3D9.h>

CGbool cgD3D9IsProfileSupported( CGprofile profile );

PARAMETERS

profile

The profile which will be checked for support.

RETURN VALUES

Returns CG_TRUE if \texttt{profile} is supported by the cgD3D9 library.

Returns CG_FALSE otherwise.

However if \texttt{cgD3D9SetDevice} has not been called to register a IDirect3DDevice9 device yet, this routine returns CG_TRUE for all valid D3D9 profiles.

DESCRIPTION

cgD3D9IsProfileSupported returns CG_TRUE if the profile indicated by profile is supported by the cgD3D9 library.

EXAMPLES

// assuming the program requires Shader Model 3.0 ...

if (!cgD3D9IsProfileSupported(CG_PROFILE_VS_3_0) || !cgD3D9IsProfileSupported(CG_PROFILE_PS_3_0)) {
    fprintf(stderr, "Sorry, required profiles not supported on this system.\n");
    exit(1);
}

ERRORS

None.

HISTORY

cgD3D9IsProfileSupported was introduced in Cg 1.5.

SEE ALSO

cgD3D9GetLatestPixelProfile, cgD3D9GetLatestVertexProfile

6.14 cgD3D9IsProgramLoaded

NAME

cgD3D9IsProgramLoaded - determine if a program has been loaded

SYNOPSIS

#include <Cg/cgD3D9.h>

CGbool cgD3D9IsProgramLoaded( CGprogram program );
PARAMETERS

program

The program which will be checked.

RETURN VALUES

Returns CG_TRUE if program has been loaded using cgD3D9LoadProgram.

Returns CG_FALSE otherwise.

DESCRIPTION

cgD3D9IsProgramLoaded determines if a program has been loaded using cgD3D9LoadProgram.

EXAMPLES

// program is a CGprogram initialized elsewhere
...
CGbool isLoaded = cgD3D9IsProgramLoaded(program);

ERRORS

None.

HISTORY

cgD3D9IsProgramLoaded was introduced in Cg 1.1.

SEE ALSO

cgD3D9LoadProgram

6.15 cgD3D9LoadProgram

NAME

cgD3D9LoadProgram - create a D3D shader and enable the expanded interface routines

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9LoadProgram( CGprogram program,
                           CGbool paramShadowing,
                           DWORD assemFlags );

PARAMETERS

program

A program whose compiled output is used to create the D3D shader.

paramShadowing

Indicates if parameter shadowing is desired for program.

assemFlags

The flags to pass to D3DXAssembleShader. See the D3D documentation for a list of valid flags.
RETURN VALUES

Returns D3D_OK if the function succeeds or program has already been loaded.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9LoadProgram creates a D3D shader for a program and enables use of expanded interface routines for that program.

cgD3D9LoadProgram assembles the compiled Cg output for program using D3DXAssembleShader and then creates a D3D shader using IDirect3DDevice9::CreateVertexShader or IDirect3DDevice9::CreatePixelShader depending on the program's profile.

Parameter shadowing is enabled or disabled for the program with paramShadowing. This behavior can be changed after creating the program by calling cgD3D9EnableParameterShadowing.

The D3D shader handle is not returned. If the shader handle is desired by the application, the expanded interface should not be used for that program.

EXAMPLES

// vertexProg is a CG program using a vertex profile
// pixelProg is a CG program using a pixel profile
...
HRESULT hr1 = cgD3D9LoadProgram(vertexProg, TRUE, D3DXASM_DEBUG);
HRESULT hr2 = cgD3D9LoadProgram(pixelProg, TRUE, 0);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_INVALIDPROFILE is returned if program's profile is not a supported D3D profile.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

HISTORY

cgD3D9LoadProgram was introduced in Cg 1.1.

SEE ALSO

cgD3D9EnableParameterShadowing, cgD3D9ValidateVertexDeclaration, cgD3D9SetDevice

6.16 cgD3D9RegisterStates

NAME

cgD3D9RegisterStates - registers graphics pass states for CgFX files

SYNOPSIS

#include <Cg/cgD3D9.h>

void cgD3D9RegisterStates( CGContext context );
PARAMETERS
context
The context in which to register the states.

RETURN VALUES
None.

DESCRIPTION
cgD3D9RegisterStates registers a set of states for passes in techniques in CgFX effect files. These states correspond to the set of D3D states that is relevant and/or useful to be set in passes in effect files. See the Cg User’s Guide for complete documentation of the states that are made available after calling cgD3D9RegisterStates.

EXAMPLES
// register D3D9 states for this context
CGcontext = cgCreateContext();
cgD3D9RegisterStates(context);

ERRORS
CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY
cgD3D9RegisterStates was introduced in Cg 1.5.
Starting with Cg 2.2, when cgD3D9RegisterStates creates program states it calls cgSetStateLatestProfile to register the latest profile for the appropriate program domain. The latest profile value is determined via cgD3D9GetLatestVertexProfile or cgD3D9GetLatestPixelProfile

SEE ALSO
cgCreateState, cgSetStateLatestProfile, cgSetPassState, cgResetPassState, cgCallStateValidateCallback, cgGLRegisterStates, cgD3D10RegisterStates

6.17 cgD3D9ResourceToDeclUsage

NAME
cgD3D9ResourceToDeclUsage - get the D3DDECLUSAGE member associated with a resource

SYNOPSIS
#include <Cg/cgD3D9.h>

BYTE cgD3D9ResourceToDeclUsage( CGresource resource );

PARAMETERS
resource
Enumerated type indicating the resource to convert to a D3DDECLUSAGE.
RETURN VALUES

Returns the D3DDECLUSAGE member associated with resource. This is generally the CGresource name with the index stripped off.

Returns CGD3D9_INVALID_USAGE if the resource is not a vertex shader input resource.

DESCRIPTION

cgD3D9ResourceToDeclUsage converts a CGresource enumerated type to a member of the D3DDECLUSAGE enum. The returned type is not an explicit member of the enum to match the associated member of the D3DVERTEXELEMENT9 struct, and also to allow for an error return condition.

The returned value can be used as the Usage member of the D3DVERTEXELEMENT9 struct to create a vertex declaration for a shader. See the D3D9 documentation for the full details on declaring vertex declarations in D3D9.

EXAMPLES

D3DVERTEXELEMENT9 elt =
{
    0, 0,
    D3DDECLTYPE_FLOAT3,
    D3DDECLMETHOD_DEFAULT,
    cgD3D9ResourceToDeclUsage(CG_TEXCOORD3),
    cgD3D9GetParameterResourceIndex(CG_TEXCOORD3)
};

ERRORS

None.

HISTORY

cgD3D9ResourceToDeclUsage was introduced in Cg 1.1.

SEE ALSO

cgD3D9GetVertexDeclaration, cgD3D9ValidateVertexDeclaration

6.18 cgD3D9SetDevice

NAME

cgD3D9SetDevice - set the D3D device

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetDevice( IDirect3DDevice9 * device );

PARAMETERS

device

Pointer to an IDirect3DDevice9 interface that the expanded interface will use for any D3D-specific routine it may call. This parameter can be NULL to free all D3D resources used by the expanded interface and remove its reference to the D3D device.
RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetDevice informs the expanded interface of the new D3D device. This will destroy any D3D resources for programs previously loaded with cgD3D9LoadProgram and use the new D3D device to recreate them. The expanded interface will increment the reference count to the D3D device, so this function must eventually be called with NULL to release that reference so D3D can be properly shut down.

If device is NULL, all D3D resources for programs previously loaded with cgD3D9LoadProgram are destroyed. However, these programs are still considered managed by the expanded interface, so if a new D3D device is set later these programs will be recreated using the new D3D device.

If a new device is being set, all D3D resources for programs previously loaded with cgD3D9LoadProgram are rebuilt using the new device. All shadowed parameters for these programs are maintained across D3D device changes except texture parameters. Since textures in D3D are bound to a particular D3D device, these resources cannot be saved across device changes. When these textures are recreated for the new D3D device, they must be re-bound to the sampler parameter.

Note that calling cgD3D9SetDevice(NULL) does not destroy any core runtime resources (CGprograms, CGparameters, etc.) used by the expanded interface. These must be destroyed seperately using cgDestroyProgram and cgDestroyContext.

EXAMPLES

// pDev is an IDirect3DDevice9 interface initialized elsewhere
...
cgD3D9SetDevice(pDev);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

HISTORY

cgD3D9SetDevice was introduced in Cg 1.1.

SEE ALSO

cgD3D9GetDevice, cgDestroyProgram, cgDestroyContext, cgD3D9LoadProgram

6.19 cgD3D9SetManageTextureParameters

NAME

cgD3D9SetManageTextureParameters - set the manage texture parameters flag for a context

SYNOPSIS

#include <Cg/cgD3D9.h>

void cgD3D9SetManageTextureParameters( CGcontext context,
                                       CGbool flag );
PARAMETERS

context
The context in which the automatic texture management behavior will be changed.

flag
A boolean switch which controls automatic texture management by the runtime.

RETURN VALUES

None.

DESCRIPTION

By default, cgD3D9 does not manage any texture state in D3D. It is up to the user to enable and disable textures using D3D. This behavior is the default to avoid conflicts with texture state on geometry that’s rendered with the fixed function pipeline or without cgD3D9.

If automatic texture management is desired, cgD3D9SetManageTextureParameters may be called with flag set to CG_TRUE before cgD3D9BindProgram is called. Whenever cgD3D9BindProgram is called, the cgD3D9 runtime will make all the appropriate texture parameter calls on the application’s behalf.

Calling cgD3D9SetManageTextureParameters with flag set to CG_FALSE will disable automatic texture management.

NOTE: When cgD3D9SetManageTextureParameters is set to CG_TRUE, applications should not make texture state change calls to D3D after calling cgD3D9BindProgram, unless the application is trying to override some parts of cgD3D9’s texture management.

EXAMPLES

// Enable automatic texture management
cgD3D9SetManageTextureParameters( pCtx, CG_TRUE );

ERRORS

None.

HISTORY

cgD3D9SetManageTextureParameters was introduced in Cg 1.5.

SEE ALSO

cgD3D9GetManageTextureParameters, cgD3D9BindProgram

6.20 cgD3D9SetSamplerState

NAME

cgD3D9SetSamplerState - set the state associated with a sampler parameter

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetSamplerState( CGparameter param,
D3DSAMPLERSTATETYPE type,
DWORD value );

PARAMETERS

param
The sampler parameter whose state is to be set.
type
The D3D sampler state to set.
value
A value appropriate for type. See the D3D documentation for appropriate values for each valid type.

RETURN VALUES

Returns D3D_OK if the function succeeds.
Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetSamplerState sets the state associated with a sampler parameter.

EXAMPLES

// param is a CGparameter handle of type sampler
...
// Set this sampler for tri-linear filtering
cgD3D9SetSamplerState(param, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
cgD3D9SetSamplerState(param, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
cgD3D9SetSamplerState(param, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_INVALIDPROFILE is returned if params's profile is not a supported D3D profile.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTSAMPLER is returned if param is not a sampler.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetSamplerState was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetTexture, cgD3D9SetTextureWrapMode
6.21 cgD3D9SetTextureParameter

NAME

cgD3D9SetTextureParameter - sets the value of a texture parameter

SYNOPSIS

#include <Cg/cgD3D9.h>

void cgD3D9SetTextureParameter( CGparameter param,
                                IDirect3DBaseTexture9 * texture );

PARAMETERS

    param
    The texture parameter that will be set.

    texture
    An D3D texture to which the parameter will be set.

RETURN VALUES

None.

DESCRIPTION

cgD3D9SetTextureParameter sets the value of a texture parameter to a given D3D9 texture object.
cgD3D9SetTextureParameter is to be used for setting texture parameters in a CgFX effect instead of
cgD3D9SetTexture.

EXAMPLES

IDirect3DTexture9 *myTexture;
// Assume myTexture is loaded here...

// param is an effect sampler parameter
cgD3D9SetTextureParameter( param, myTexture );

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter handle.

HISTORY

cgD3D9SetTextureParameter was introduced in Cg 1.5.

SEE ALSO

cgD3D9GetTextureParameter, cgD3D9SetManageTextureParameters

6.22 cgD3D9SetTexture

NAME

cgD3D9SetTexture - set the texture for a sampler parameter
SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetTexture( CGparameter param,
                          IDirect3DBaseTexture9 * texture );

PARAMETERS

  param
  The sampler parameter whose values are to be set.

  texture
  Pointer to an IDirect3DBaseTexture9, the texture to set for param.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetTexture sets the texture for a sampler parameter.

When parameter shadowing is enabled, the D3D runtime will maintain a reference (via AddRef) to texture, so care
must be taken to set the parameter back to NULL when the texture is no longer needed. Otherwise the reference count
will not reach zero and the texture’s resources will not get destroyed. When destroying the program that the parameter
is associated with, all references to these textures are automatically removed.

EXAMPLES

    // param is a CGparameter handle of type sampler
    // tex is an IDirect3DTexture9* initialized elsewhere
    ...
    cgD3D9SetTexture(param, tex);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_INVALIDPROFILE is returned if param's profile is not a supported D3D profile.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded
interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTSAMPLER is returned if param is not a sampler.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetTexture was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetSamplerState, cgD3D9SetTextureWrapMode
6.23 cgD3D9SetTextureWrapMode

NAME

cgD3D9SetTextureWrapMode - set the texture wrap mode for a sampler parameter

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetTextureWrapMode( CGparameter param,
                DWORD value );

PARAMETERS

param
    The sampler parameter whose wrap mode is to be set.
value
    The texture wrap mode. value can be zero (0) or a combination of D3DWRAP_U, D3DWRAP_V,
    and D3DWRAP_W. See the D3D documentation for an explanation of texture wrap modes
    (D3DRS_WRAP0-7).

RETURN VALUES

Returns D3D_OK if the function succeeds.
Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetTextureWrapMode sets the texture wrap mode associated with a sampler parameter.

EXAMPLES

// param is a CGparameter handle of type sampler
...
// Set this sampler for wrapping in 2D
cgD3D9SetTextureWrapMode(param, D3DWRAP_U | D3DWRAP_V);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_INVALIDPROFILE is returned if params’s profile is not a supported D3D profile.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded
interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTSAMPLER is returned if param is not a sampler.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetTextureWrapMode was introduced in Cg 1.1.
SEE ALSO

cgD3D9SetTexture, cgD3D9SetSamplerState

6.24 cgD3D9SetUniformArray

NAME
cgD3D9SetUniformArray - set the elements of an array of uniform parameters

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetUniformArray( CGparameter param,
  DWORD offset,
  DWORD numItems,
  const void * values );

PARAMETERS

  param
  The parameter whose array elements are to be set. It must be a uniform parameter that is not a sampler.
  offset
  The offset at which to start setting array elements.
  numItems
  The number of array elements to set.
  values
  An array of floats, the elements in the array to set for param. The amount of data required depends
  on the type of parameter, but is always specified as an array of one or more floating point values.
  The type is void* so a compatible user-defined structure can be passed in without type-casting. Use
  cgD3D9TypeToSize to determine how many values are required for a particular type. This size multiplied
  by numItems is the number of values this function expects.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetUniformArray sets the elements for an array of uniform parameters. All values should be of type float.
There is assumed to be enough values to set all specified elements of the array.

EXAMPLES

// param is a CGparameter handle of type float3
// arrayParam is a CGparameter handle of type float2x2[3]
...
// initialize the data for each parameter
D3DXVECTOR3 paramData(1,2,3);
float arrayData[2][2][2] =

6.24. cgD3D9SetUniformArray
{ 
  0,1,
  2,3,
  4,5,
  6,7
};
... 
// non-arrays can be set, but only when offset=0 and numItems=1.
cgD3D9SetUniformArray(param, paramData, 0, 1);
// set the 2nd and 3rd elements of the array
cgD3D9SetUniform(arrayParam, arrayData, 1, 2);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_NULLVALUE is returned if values is NULL.

CGD3D9ERR_OUTOFRANGE is returned if offset plus numItems is out of the range of param.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetUniformArray was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetUniform, cgD3D9SetUniformMatrix, cgD3D9SetUniformMatrixArray, cgD3D9TypeToSize

6.25 cgD3D9SetUniformMatrixArray

NAME

cgD3D9SetUniformMatrixArray - set the elements for an array of uniform matrix parameters

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetUniformMatrixArray( CGparameter param,
                                      DWORD offset,
                                      DWORD numItems,
                                      const D3DMATRIX * matrices );

PARAMETERS
	param
  The parameter whose array elements are to be set. It must be a uniform matrix parameter.

  offset
The offset at which to start setting array elements.

numItems

The number of array elements to set.

matrices

An array of matrices to set for param. The upper-left portion of each matrix is extracted to fit the size of the input parameter. numItems matrices are expected to be passed to the function.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9SetUniformMatrixArray sets the elements for an array of uniform matrix parameters.

EXAMPLES

```
// matrixParam is a CGparameter handle of type float3x2
// arrayParam is a CGparameter handle of type float4x4[4]
...
// initilize the data for each parameter
D3DXMATRIX matTexTransform(
  0.5f, 0, 0, 0,
  0, 0.5f, 0, 0,
  0.5f, 0.5f, 0, 0,
  0, 0, 0, 0);
D3DXMATRIX matRot[2];
D3DXMatrixRotationAxis(&matRot[0], &D3DXVECTOR3(0,0,1), D3DX_PI*0.5f);
D3DXMatrixRotationAxis(&matRot[1], &D3DXVECTOR3(0,1,0), D3DX_PI*0.5f);
...
// only use the upper-left portion.
// non-arrays can be set, but only when offset=0 and numItems=1.
cgD3D9SetUniformArray(matrixParam, &matTexTransform, 0, 1);
// set the 3rd and 4th elements of the array
cgD3D9SetUniformArray(arrayParam, matRot, 2, 2);
```

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTMATRIX is returned if param is not a matrix.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_NULLVALUE is returned if matrices is NULL.

CGD3D9ERR_OUTOFRANGE is returned if offset plus numItems is out of the range of param.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.
**HISTORY**

cgD3D9SetUniformMatrixArray was introduced in Cg 1.1.

**SEE ALSO**

cgD3D9SetUniform, cgD3D9SetUniformArray, cgD3D9SetUniformMatrix, cgD3D9TypeToSize

### 6.26 cgD3D9SetUniformMatrix

**NAME**

cgD3D9SetUniformMatrix - set the values of a uniform matrix parameter

**SYNOPSIS**

```c
#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetUniformMatrix( CGparameter param,
    const D3DMATRIX * matrix );
```

**PARAMETERS**

- **param**
  - The parameter whose values are to be set. It must be a uniform matrix parameter.

- **matrix**
  - The matrix to set for the parameter. The upper-left portion of the matrix is extracted to fit the size of `param`.

**RETURN VALUES**

Returns **D3D_OK** if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

**DESCRIPTION**

cgD3D9SetUniformMatrix sets the values of a uniform matrix parameter.

**EXAMPLES**

```c
// matrixParam is a CGparameter handle of type float3x2
// arrayParam is a CGparameter handle of type float4x4[2]
...
// initialize the data for each parameter
D3DXMATRIX matTexTransform(  
    0.5f, 0, 0, 0,  
    0, 0.5f, 0, 0,  
    0.5f, 0.5f, 0, 0,  
    0, 0, 0, 0
);
D3DXMATRIX matRot[2];
D3DXMatrixRotationAxis(&matRot[0], &D3DXVECTOR3(0,0,1), D3DX_PI*0.5f);
D3DXMatrixRotationAxis(&matRot[1], &D3DXVECTOR3(0,1,0), D3DX_PI*0.5f);
...
// only use the upper-left portion
cgD3D9SetUniform(matrixParam, &matTexTransform);
```
Cg 3.1 Reference Manual, Release 3.1

// you can use arrays, but you must set the entire array
cgD3D9SetUniform(arrayParam, matRot);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTMATRIX is returned if param is not a matrix.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetUniformMatrix was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetUniform, cgD3D9SetUniformMatrixArray, cgD3D9TypeToSize

6.27 cgD3D9SetUniform

NAME

cgD3D9SetUniform - set the value of a uniform parameter

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9SetUniform( CGparameter param,
                           const void * values );

PARAMETERS

  param

  The parameter whose values are to be set. param must be a uniform parameter that is not a sampler.

  values

  The values to which to set param. The amount of data required depends on the type of parameter, but
  is always specified as an array of one or more floating point values. The type is void* so a compatible
  user-defined structure can be passed in without type-casting. Use cgD3D9TypeToSize to determine how
  many values are required for a particular type.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.
DESCRIPTION

cgD3D9SetUniform sets the value for a uniform parameter. All values should be of type float. There is assumed to be enough values to set all elements of the parameter.

EXAMPLES

// param is a CGparameter handle of type float3
// matrixParam is a CGparameter handle of type float2x3
// arrayParam is a CGparameter handle of type float2x2[3]
...
// initialize the data for each parameter
D3DXVECTOR3 paramData(1,2,3);
float matrixData[2][3] =
{
0,1,2,
3,4,5
};
float arrayData[3][2][2] =
{
0,1,
2,3,
4,5,
6,7,
8,9,
0,1
};
...
// set the parameters
cgD3D9SetUniform(param, paramData);
cgD3D9SetUniform(matrixParam, matrixData);
// you can use arrays, but you must set the entire array
cgD3D9SetUniform(arrayParam, arrayData);

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NOTUNIFORM is returned if param is not a uniform parameter.

CGD3D9ERR_INVALIDPARAM is returned if the parameter fails to set for any other reason.

HISTORY

cgD3D9SetUniform was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetUniformArray, cgD3D9SetUniformMatrix, cgD3D9SetUniformMatrixArray, cgD3D9TypeToSize
6.28 cgD3D9TranslateCGerror

NAME
cgD3D9TranslateCGerror - convert a Cg runtime error into a string

SYNOPSIS
#include <Cg/cgD3D9.h>
const char* cgD3D9TranslateCGerror( CGerror error );

PARAMETERS
error
The error code to translate. Can be a core runtime error or a D3D runtime error.

RETURN VALUES
Returns a pointer to a string describing error.

DESCRIPTION
cgD3D9TranslateCGerror converts a Cg runtime error into a string. This routine should be called instead of the core runtime routine cgGetErrorString because it will also translate errors that the Cg D3D runtime generates.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCallback.

EXAMPLES
char buf[512];
CGerror error = cgGetLastError();
if (error != CG_NO_ERROR)
{
    sprintf(buf, "An error occurred. Error description: '%s'\n", cgD3D9TranslateCGerror(error));
    OutputDebugString(buf);
}

ERRORS
None.

HISTORY
cgD3D9TranslateCGerror was introduced in Cg 1.1.

SEE ALSO
cgGetErrorString, cgSetErrorCallback

6.29 cgD3D9TranslateHRESULT

NAME
cgD3D9TranslateHRESULT - convert an HRESULT into a string
SYNOPSIS

#include <Cg/cgD3D9.h>

const char * cgD3D9TranslateHRESULT( HRESULT hr );

PARAMETERS

hr

The HRESULT to translate. Can be a generic HRESULT or a D3D runtime error.

RETURN VALUES

Returns a pointer to a string describing the error.

DESCRIPTION

cgD3D9TranslateHRESULT converts an HRESULT into a string. This routine should be called instead of 
DXGetErrorDescription9 because it will also translate errors that the Cg D3D runtime generates.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCallback.

EXAMPLES

char buf[512];
HRESULT hres = cgD3D9GetLastError();
if (FAILED(hres))
    {
        sprintf(buf, "A D3D error occurred. Error description: '%s'\n",
            cgD3D9TranslateHRESULT(hres));
        OutputDebugString(buf);
    }

ERRORS

None.

HISTORY

cgD3D9TranslateHRESULT was introduced in Cg 1.1.

SEE ALSO

cgD3D9TranslateCGerror, cgGetString, cgSetErrorCallback

6.30 cgD3D9TypeToSize

NAME

cgD3D9TypeToSize - get the size of a CGtype enumerated type

SYNOPSIS

#include <Cg/cgD3D9.h>

DWORD cgD3D9TypeToSize( CGtype type );
PARAMETERS

type

Member of the CGtype enumerated type whose size is to be returned.

RETURN VALUES

Returns the size of type in terms of consecutive floating point values.
Returns 0 if the type does not have an inherent size. Sampler types fall into this category.

DESCRIPTION

cgD3D9TypeToSize retrieves the size of a CGtype enumerated type in terms of consecutive floating point values.
If the type does not have an inherent size, the return value is 0. Sampler types fall into this category.

EXAMPLES

// param is a CGparameter initialized earlier
...
DWORD size = cgD3D9TypeToSize(cgGetParameterType(param));

// (sanity check that parameters have the expected size)
...
assert(cgD3D9TypeToSize(cgGetParameterType(vsModelView)) == 16);
assert(cgD3D9TypeToSize(cgGetParameterType(psColor)) == 4);

ERRORS

None.

HISTORY

cgD3D9TypeToSize was introduced in Cg 1.1.

SEE ALSO

cgD3D9ResourceToDeclUsage, cgD3D9GetVertexDeclaration, cgD3D9ValidateVertexDeclaration

6.31 cgD3D9UnbindProgram

NAME

cgD3D9UnbindProgram - de-activate a program with D3D

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9UnbindProgram( CGprogram program );

PARAMETERS

program

The program to de-activate with D3D.
RETURN VALUES

Returns **D3D_OK** if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9UnbindProgram de-activates a program with D3D. The program is de-activated using *IDirect3DDevice9::SetVertexShader* or *IDirect3DDevice9::SetPixelShader* with a NULL argument depending on the program’s profile type.

EXAMPLES

```cpp
// vertexProg and pixelProg are CG programs initialized elsewhere
// pDev is an IDirect3DDevice9 interface initialized elsewhere
...
HRESULT hr = cgD3D9BindProgram(vertexProg);
HRESULT hr2 = cgD3D9BindProgram(pixelProg);
// Draw a quad using the vertex and pixel shader
// A vertex and index buffer are set up elsewhere.
HRESULT hr3 = pDev->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, 4, 0, 2);

cgD3D9UnbindProgram(vertexProg);
cgD3D9UnbindProgram(pixelProg);
```

ERRORS

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

HISTORY

cgD3D9UnbindProgram was introduced in Cg 3.0.

SEE ALSO

cgD3D9LoadProgram, cgD3D9BindProgram

6.32  cgD3D9UnloadAllPrograms

NAME

cgD3D9UnloadAllPrograms - unload all D3D programs

SYNOPSIS

```cpp
#include <Cg/cgD3D9.h>

void cgD3D9UnloadAllPrograms( void );
```

PARAMETERS

None.
RETURN VALUES
None.

DESCRIPTION

cgD3D9UnloadAllPrograms unloads all of the currently loaded D3D programs.
See cgD3D9UnloadProgram for details on what the runtime does when unloading a program.

EXAMPLES

// unload all D3D programs
cgD3D9UnloadAllPrograms();

ERRORS
None.

HISTORY

cgD3D9UnloadAllPrograms was introduced in Cg 1.5.

SEE ALSO

cgD3D9UnloadProgram

6.33 cgD3D9UnloadProgram

NAME

cgD3D9UnloadProgram - destroy D3D shader and disable use of expanded interface routines

SYNOPSIS

#include <Cg/cgD3D9.h>

HRESULT cgD3D9UnloadProgram( CGprogram program );

PARAMETERS

program

The program for which to disable expanded interface management. The CGprogram handle is still valid
after this call.

RETURN VALUES

Returns D3D_OK if the function succeeds.
Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D9UnloadProgram destroys the D3D shader for a program and disables use of expanded interface routines for
that program.

This call does not destroy the CGprogram itself. It only destroys the resources used by the expanded interface, such
as the D3D shader object and any shadowed parameters. Use the core runtime function cgDestroyProgram to free the
CGprogram itself. Also note that freeing a CGprogram using the core runtime implicitly calls this routine to avoid resource leaks.

This call is only necessary if specific lifetime control of expanded interface resources outside the lifetime of their associated CGprogram is desired. For instance, if the expanded interface is no longer used, but the CGprogram handle will still be used.

**EXAMPLES**

```c
// prog is a CGprogram initialized elsewhere
...
HRESULT hres = cgD3D9UnloadProgram(prog);
```

**ERRORS**

cgD3D9Failed is generated if a D3D function returns an error.

CGD3D9ERR_NOTLOADED is returned if program was not loaded with the cgD3D9LoadProgram.

CGD3D9ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D9SetDevice.

**HISTORY**

cgD3D9UnloadProgram was introduced in Cg 1.1.

**SEE ALSO**

cgD3D9UnloadAllPrograms, cgDestroyProgram

### 6.34 cgD3D9ValidateVertexDeclaration

**NAME**

cgD3D9ValidateVertexDeclaration - validate a custom D3D9 vertex declaration stream

**SYNOPSIS**

```c
#include <Cg/cgD3D9.h>

CGbool cgD3D9ValidateVertexDeclaration( CGprogram program,
                        const D3DVERTEXELEMENT9 * decl );
```

**PARAMETERS**

- **program**
  - The program to test for compatibility.

- **decl**
  - The D3D9 custom vertex declaration stream to test for compatibility. It must be terminated by D3DDECL_END().

**RETURN VALUES**

Returns CG_TRUE if the vertex stream is compatible.

Returns CG_FALSE otherwise.
DESCRIPTION

cgD3D9ValidateVertexDeclaration tests a custom D3D9 vertex declaration stream for compatibility with the inputs expected by a program.

For a vertex stream to be compatible with a program’s expected inputs it must have a D3DVERTEXELEMENT9 element for each varying input parameter that the program uses.

EXAMPLES

// Decl is a custom vertex declaration already setup
CGbool ret = cgD3D9ValidateVertexDeclaration(program, Decl);
if (ret == CG_FALSE)
    printf( "Vertex declaration not compatible with "
            "the program’s varying parameters.\n" );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if program is not a valid program handle.

HISTORY

cgD3D9ValidateVertexDeclaration was introduced in Cg 1.1.

SEE ALSO

cgD3D9ResourceToDeclUsage
7.1 cgD3D10BindProgram

NAME

cgD3D10BindProgram - activate a program with D3D

SYNOPSIS

#include <Cg/cgD3D10.h>

HRESULT cgD3D10BindProgram( CGprogram program );

PARAMETERS

    program

The program to activate with D3D.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D10BindProgram activates a program with D3D. The program is activated using
ID3D10Device::SetVertexShader or ID3D10Device::SetPixelShader depending on the program’s profile type.

D3D allows only one vertex shader and one pixel shader to be active at any given time, so activating a program of a
given type implicitly deactivates any other program of a that type.

EXAMPLES

// vertexProg and pixelProg are CGprograms initialized elsewhere
// pDev is an ID3D10Device interface initialized elsewhere
...
HRESULT hr = cgD3D10BindProgram(vertexProg);
HRESULT hr2 = cgD3D10BindProgram(pixelProg);
// Draw a quad using the vertex and pixel shader
// A vertex and index buffer are set up elsewhere.
HRESULT hr3 = pDev->DrawIndexedPrimitve(D3DPT_TRIANGLELIST, 0, 4, 0, 2);
ERRORS

`cgD3D10Failed` is generated if a D3D function returns an error.

`CGD3D10ERR_NOTLOADED` is returned if *program* was not loaded with the `cgD3D10LoadProgram`.

`CGD3D10ERR_NODEVICE` is returned if a required D3D device is `NULL`. This usually occurs when an expanded interface routine is called but a D3D device has not been set with `cgD3D10SetDevice`.

HISTORY

`cgD3D10BindProgram` was introduced in Cg 2.1.

SEE ALSO

`cgD3D10LoadProgram, cgD3D10SetTextureParameter`

### 7.2 cgD3D10CreateBuffer

**NAME**

`cgD3D10CreateBuffer` - create a D3D10 buffer object

**SYNOPSIS**

```c
#include <Cg/cgD3D10.h>

CGbuffer cgD3D10CreateBuffer( CGcontext context,
                               int size,
                               const void *data,
                               D3D10_USAGE bufferUsage );
```

**PARAMETERS**

- `context`
  - The context to which the new buffer will be added.
- `size`
  - The length in bytes of the buffer to create.
- `data`
  - The initial data to be copied into the buffer. NULL will fill the buffer with zero.
- `bufferUsage`
  - A D3D10 usage flag as specified by `D3D10_USAGE`.

**RETURN VALUES**

Returns a `CGbuffer` handle on success.

Returns `NULL` if any error occurs.

**DESCRIPTION**

`cgD3D10CreateBuffer` creates a D3D10 buffer object.
EXAMPLES

Cgbuffer myBuffer = cgD3D10CreateBuffer( myCgContext, sizeof( float ) * 16, myData, D3D10_USAGE_DEFAULT );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_MEMORY_ALLOC_ERROR is generated if a buffer couldn’t be created.

HISTORY
cgD3D10CreateBuffer was introduced in Cg 3.1.

SEE ALSO
cgCreateBuffer, cgD3D11CreateBuffer, cgGLCreateBuffer, cgD3D10CreateBufferFromObject, cgD3D10GetBufferObject

7.3 cgD3D10CreateBufferFromObject

NAME
cgD3D10CreateBufferFromObject · create a Cg buffer from a D3D10 buffer object

SYNOPSIS

#include <Cg/cgD3D10.h>

Cgbuffer cgD3D10CreateBufferFromObject( CGcontext context, ID3D10Buffer * obj, CGbool manageObject );

PARAMETERS

context

The context to which the new buffer will be added.

obj

A D3D10 buffer object created by the application.

manageObject

A boolean switch which controls whether obj will be deleted by the runtime when the Cgbuffer object returned by cgD3D10CreateBufferFromObject is destroyed.

RETURN VALUES

Returns a Cgbuffer handle on success.
Returns NULL if any error occurs.

DESCRIPTION

cgD3D10CreateBufferFromObject creates a Cg buffer from a preexisting D3D10 buffer object. This D3D10 object will be deleted by the runtime when the Cg buffer is destroyed if manageObject is CG_TRUE. Otherwise the application is responsible for deleting the D3D10 object obj.
EXAMPLES

CGbuffer myBuffer = cgD3D10CreateBufferFromObject( myCgContext, d3d10BufferObj, CG_TRUE );

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D10CreateBufferFromObject was introduced in Cg 3.1.

SEE ALSO

cgCreateBuffer, cgDestroyBuffer, cgD3D10GetBufferByIndex

7.4 cgD3D10GetBufferByIndex

NAME

cgD3D10GetBufferByIndex - Returns a pointer to an ID3D10Buffer interface by the constant buffer index.

SYNOPSIS

#include <Cg/cgD3D10.h>

ID3D10Buffer * cgD3D10GetBufferByIndex( CGprogram Program,
                                       UINT Index );

PARAMETERS

  Program
    The Cg program containing the buffer.
  Index
    A zero-based index.

RETURN VALUES

Returns a pointer to an ID3D10Buffer interface containing the constant buffer.

DESCRIPTION

cgD3D10GetBufferByIndex returns a pointer to an ID3D10Buffer interface containing the constant buffer for manual manipulation. If the user manually changes the constant values in this way, the constant values contained in the corresponding CGbuffer (if exists) will be stale.

EXAMPLES

ID3D10Buffer * myConstantBuffer = cgD3D10GetBufferByIndex( myCgProgram, 0 );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR if Program is not a valid Cg program.

HISTORY

cgD3D10GetBufferByIndex was introduced in Cg 2.1.
SEE ALSO

cgCreateBuffer

7.5 cgD3D10GetBufferObject

NAME
cgD3D10GetBufferObject - get the D3D10 buffer object for a buffer

SYNOPSIS

#include <Cg/cgD3D10.h>

ID3D10Buffer * cgD3D10GetBufferObject( CGbuffer buffer );

PARAMETERS

buffer

The buffer for which the associated D3D10 buffer object will be retrieved.

RETURN VALUES

Returns the D3D10 buffer object associated with buffer.

Returns NULL if any error occurs.

DESCRIPTION

cgD3D10GetBufferObject returns the D3D10 buffer object associated with a buffer.

EXAMPLES

ID3D10Buffer * id = cgD3D10GetBufferObject( myBuffer );

ERRORS

CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

HISTORY

cgD3D10GetBufferObject was introduced in Cg 3.1.

SEE ALSO

cgCreateBuffer, cgD3D10CreateBuffer, cgD3D10CreateBufferFromObject

7.6 cgD3D10GetCompiledProgram

NAME

cgD3D10GetCompiledProgram - Gets the compiled shader as a ID3D10Blob returned from Direct3D after cgD3D10LoadProgram is called.
SYNOPSIS

#include <Cg/cgD3D10.h>

ID3D10Blob * cgD3D10GetCompiledProgram( CGprogram program );

PARAMETERS

program

The program handle after a call to cgD3D10LoadProgram has been made.

RETURN VALUES

Returns a pointer to a ID3D10Blob object containing the compiled shader code.

Returns NULL if the program was not loaded.

DESCRIPTION

cgD3D10GetCompiledProgram allows the user to get back the compiled shader from Direct3D once cgD3D10LoadProgram has been called.

EXAMPLES

CGprogram myCgProgram = cgCreateProgram( ... ); cgD3D10LoadProgram( myCgProgram, 0 );

ID3D10Blob * obj = cgD3D10GetCompiledProgram( myCgProgram );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if the program is invalid.

HISTORY

cgD3D10GetCompiledProgram was introduced in Cg 2.1.

SEE ALSO

cgD3D10LoadProgram

7.7 cgD3D10GetDevice

NAME

cgD3D10GetDevice - retrieves the current D3D10 device associated with a context

SYNOPSIS

#include <Cg/cgD3D10.h>

ID3D10Device * cgD3D10GetDevice( CGcontext context );

PARAMETERS

context

The context from which to get the current device.
RETURN VALUES

Returns the current D3D10 device associated with context.

DESCRIPTION

cgD3D10GetDevice retrieves the current D3D10 device associated with context. Note that the returned device pointer may be NULL.

EXAMPLES

ID3D10Device* curDevice = cgD3D10GetDevice(ctx);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D10GetDevice was introduced in Cg 2.1.

SEE ALSO

cgD3D10SetDevice

7.8 cgD3D10GetIASignatureByPass

NAME

cgD3D10GetIASignatureByPass - Gets the compiled vertex shader signature as a ID3D10Blob from a pass of a validated technique.

SYNOPSIS

#include <Cg/cgD3D10.h>

ID3D10Blob * cgD3D10GetIASignatureByPass( CGpass pass );

PARAMETERS

pass

The pass handle after validation of a CgFX technique.

RETURN VALUES

Returns a pointer to a ID3D10Blob object containing the vertex shader signature.

Returns NULL if the program was not loaded.

DESCRIPTION

cgD3D10GetIASignatureByPass allows the user to get back the vertex shader signature of a pass of a validated CgFX technique.

EXAMPLES
myCgEffect = cgCreateEffectFromFile( myCgContext, "effect.cgfx", NULL );
myCgTechnique = cgGetFirstTechnique( myCgEffect );

if( cgValidateTechnique( myCgTechnique ) != CG_FALSE )
{
  const D3D10_INPUT_ELEMENT_DESC layout[] =
  {
    { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D10_INPUT_PER_VERTEX_DATA, 0 },
  };
  CGpass myPass = cgGetFirstPass( myCgTechnique );
  ID3D10Blob * pVSBuf = cgD3D10GetIASignatureByPass( myPass );
  hr = pDevice->CreateInputLayout( layout, 1, pVSBuf->GetBufferPointer(),
  pVSBuf->GetBufferSize(), &g_pVertexLayout );
}

ERRORS
CG_INVALID_PASS_HANDLE_ERROR is generated if the pass is invalid.

HISTORY
cgD3D10GetIASignatureByPass was introduced in Cg 2.1.

SEE ALSO
cgD3D10GetCompiledProgram

7.9 cgD3D10GetLastError

NAME
cgD3D10GetLastError - get the last D3D error that occurred

SYNOPSIS
#include <Cg/cgD3D10.h>
HRESULT cgD3D10GetLastError( void );

PARAMETERS
None.

RETURN VALUES
Returns the last D3D error that occurred during an expanded interface function call.
Returns D3D_OK if no D3D error has occurred since the last call to cgD3D10GetLastError.

DESCRIPTION
cgD3D10GetLastError retrieves the last D3D error that occurred during an expanded interface function call. The last error is always cleared immediately after the call.
EXAMPLES

HRESULT lastError = cgD3D10GetLastError();

ERRORS

None.

HISTORY

cgD3D10GetLastError was introduced in Cg 2.1.

SEE ALSO

cgD3D10TranslateHRESULT

7.10 cgD3D10GetLatestGeometryProfile

NAME

cgD3D10GetLatestGeometryProfile - get the latest supported geometry shader version

SYNOPSIS

#include <Cg/cgD3D10.h>

CGprofile cgD3D10GetLatestGeometryProfile( void );

PARAMETERS

None.

RETURN VALUES

Returns the latest geometry shader version supported by the current D3D device.

Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION

cgD3D10GetLatestGeometryProfile retrieves the latest geometry shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 2.2, if the environment variable CGD3D10_LATEST_GEOMETRY_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGprofile value that will be returned by cgD3D10GetLatestGeometryProfile.

EXAMPLES

CGprofile bestGeometryProfile = cgD3D10GetLatestGeometryProfile();

ERRORS

None.

HISTORY

cgD3D10GetLatestGeometryProfile was introduced in Cg 2.1.
SEE ALSO

cgD3D10GetLatestPixelProfile, cgD3D10GetLatestVertexProfile

7.11 cgD3D10GetLatestPixelProfile

NAME
cgD3D10GetLatestPixelProfile - get the latest supported pixel shader version

SYNOPSIS

#include <Cg/cgD3D10.h>

CGprofile cgD3D10GetLatestPixelProfile( void );

PARAMETERS

None.

RETURN VALUES

Returns the latest pixel shader version supported by the current D3D device.

Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION

cgD3D10GetLatestPixelProfile retrieves the latest pixel shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 2.2, if the environment variable CGD3D10_LATEST_PIXEL_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGprofile value that will be returned by cgD3D10GetLatestPixelProfile.

EXAMPLES

CGprofile bestPixelProfile = cgD3D10GetLatestPixelProfile();

ERRORS

None.

HISTORY

cgD3D10GetLatestPixelProfile was introduced in Cg 2.1.

SEE ALSO

cgD3D10GetLatestGeometryProfile, cgD3D10GetLatestVertexProfile

7.12 cgD3D10GetLatestVertexProfile

NAME

cgD3D10GetLatestVertexProfile - get the latest supported vertex shader version
SYNOPSIS

#include <Cg/cgD3D10.h>

CGprofile cgD3D10GetLatestVertexProfile ( void );

PARAMETERS

None.

RETURN VALUES

Returns the latest vertex shader version supported by the current D3D device.

Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION

cgD3D10GetLatestVertexProfile retrieves the latest vertex shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 2.2, if the environment variable CGD3D10_LATEST_VERTEX_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGProfile value that will be returned by cgD3D10GetLatestVertexProfile.

EXAMPLES

CGprofile bestVertexProfile = cgD3D10GetLatestVertexProfile();

ERRORS

None.

HISTORY

cgD3D10GetLatestVertexProfile was introduced in Cg 2.1.

SEE ALSO

cgD3D10GetLatestGeometryProfile, cgD3D10GetLatestPixelProfile

7.13 cgD3D10GetManageTextureParameters

NAME

cgD3D10GetManageTextureParameters - get the manage texture parameters flag from a context

SYNOPSIS

#include <Cg/cgD3D10.h>

CGbool cgD3D10GetManageTextureParameters ( CGcontext context );

PARAMETERS

context

The context from which the automatic texture management setting will be retrieved.
RETURN VALUES

Returns the manage texture management flag from context.

DESCRIPTION

cgD3D10GetManageTextureParameters returns the manage texture management flag from context. See cgD3D10SetManageTextureParameters for more information.

EXAMPLES

CGbool manage = cgD3D10GetManageTextureParameters( pCtx );
if( manage )
    doSomething();

ERRORS

None.

HISTORY

cgD3D10GetManageTextureParameters was introduced in Cg 2.1.

SEE ALSO

cgD3D10SetManageTextureParameters

7.14 cgD3D10GetOptimalOptions

NAME

cgD3D10GetOptimalOptions - get the best set of compiler options for a profile

SYNOPSIS

#include <Cg/cgD3D10.h>

char const ** cgD3D10GetOptimalOptions( CGprofile profile );

PARAMETERS

profile

The profile whose optimal arguments are requested.

RETURN VALUES

Returns a null-terminated array of strings representing the optimal set of compiler options for profile.

Returns NULL if no D3D device is currently set.

DESCRIPTION

cgD3D10GetOptimalOptions returns the best set of compiler options for a given profile. This is an expanded interface function because it needs to know about the D3D device to determine the most optimal options.

The elements of the returned array are meant to be used as part of the args parameter to cgCreateProgram or cgCreateProgramFromFile.
The returned string does not need to be destroyed by the application. However, the contents could change if the function is called again for the same profile but a different D3D device.

**EXAMPLES**

```c
const char * vertOptions[] = { myCustomArgs,
                               cgD3D10GetOptimalOptions(vertProfile),
                               NULL }

// create the vertex shader
CGprogram myVS = cgCreateProgramFromFile( context,
                                          CG_SOURCE,
                                          "vshader.cg",
                                          vertProfile,
                                          "VertexShader",
                                          vertOptions);
```

**ERRORS**

None.

**HISTORY**

cgD3D10GetOptimalOptions was introduced in Cg 2.1.

**SEE ALSO**

cgD3D10GetLatestVertexProfile, cgD3D10GetLatestPixelProfile, cgCreateProgram, cgCreateProgramFromFile

### 7.15 cgD3D10GetProgramErrors

**NAME**

cgD3D10GetProgramErrors - Gets a list of errors returned from Direct3D if the program did not load.

**SYNOPSIS**

```c
#include <Cg/cgD3D10.h>

ID3D10Blob * cgD3D10GetProgramErrors( CGprogram program );
```

**PARAMETERS**

- program

  The program handle after a call to cgD3D10LoadProgram has been made.

**RETURN VALUES**

Returns a pointer to a ID3D10Blob object containing a list of errors if the program did not load.

Returns NULL if the program was loaded.

**DESCRIPTION**

cgD3D10GetProgramErrors allows the user to get back the compiled shader from Direct3D once cgD3D10LoadProgram has been called.
EXAMPLES

CG program myCgProgram = cgCreateProgram( ... ); cgD3D10LoadProgram( myCgProgram, 0 );
ID3D10Blob * err = cgD3D10GetProgramErrors( myCgProgram );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if the program is invalid;

HISTORY

cgD3D10GetProgramErrors was introduced in Cg 2.1.

SEE ALSO

cgD3D10LoadProgram

7.16 cgD3D10IsProfileSupported

NAME

cgD3D10IsProfileSupported - determine if a profile is supported by cgD3D10

SYNOPSIS

#include <Cg/cgD3D10.h>

CGbool cgD3D10IsProfileSupported( CGprofile profile );

PARAMETERS

profile

The profile which will be checked for support.

RETURN VALUES

Returns CG_TRUE if profile is supported by the cgD3D10 library.

Returns CG_FALSE otherwise.

However if cgD3D10SetDevice has not been called to register a ID3D10Device device yet, this routine returns
CG_TRUE for all valid D3D10 profiles.

DESCRIPTION

cgD3D10IsProfileSupported returns CG_TRUE if the profile indicated by profile is supported by the cgD3D10 library.

EXAMPLES

// assuming the program requires Shader Model 3.0 ...

if ( (!cgD3D10IsProfileSupported(CG_PROFILE_VS_3_0)) ||
   (!cgD3D10IsProfileSupported(CG_PROFILE_PS_3_0)) ) {
   fprintf(stderr, "Sorry, required profiles not supported on this system.\n")
   exit(1);
}
7.17 cgD3D10IsProgramLoaded

NAME

cgD3D10IsProgramLoaded - determine if a program has been loaded

SYNOPSIS

#include <Cg/cgD3D10.h>

CGbool cgD3D10IsProgramLoaded( CGprogram program );

PARAMETERS

program

The program which will be checked.

RETURN VALUES

Returns CG_TRUE if program has been loaded using cgD3D10LoadProgram.

Returns CG_FALSE otherwise.

DESCRIPTION

cgD3D10IsProgramLoaded determines if a program has been loaded using cgD3D10LoadProgram.

EXAMPLES

// program is a CGprogram initialized elsewhere
...
CGbool isLoaded = cgD3D10IsProgramLoaded(program);

ERRORS

None.

HISTORY

cgD3D10IsProgramLoaded was introduced in Cg 2.1.

SEE ALSO

cgD3D10LoadProgram
7.18 cgD3D10LoadProgram

NAME
cgD3D10LoadProgram - create a D3D shader and enable the expanded interface routines

SYNOPSIS

#include <Cg/cgD3D10.h>

HRESULT cgD3D10LoadProgram( CGprogram program,  
                               UINT flags );

PARAMETERS

program
   A program whose compiled output is used to create the D3D shader.

flags
   The flags to pass to D3DXAssembleShader. See the D3D documentation for a list of valid flags.

RETURN VALUES

Returns D3D_OK if the function succeeds or program has already been loaded.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D10LoadProgram creates a D3D shader for a program and enables use of expanded interface routines for that program.

cgD3D10LoadProgram assembles the compiled Cg output for program using D3DXAssembleShader and then creates a D3D shader using ID3D10Device::CreateVertexShader or ID3D10Device::CreatePixelShader depending on the program’s profile.

The D3D shader handle is not returned. If the shader handle is desired by the application, the expanded interface should not be used for that program.

EXAMPLES

// vertexProg is a CGprogram using a vertex profile
// pixelProg is a CGprogram using a pixel profile

...  
HRESULT hr1 = cgD3D10LoadProgram(vertexProg, TRUE, D3DXASM_DEBUG);  
HRESULT hr2 = cgD3D10LoadProgram(pixelProg, TRUE, 0);

ERRORS

cgD3D10Failed is generated if a D3D function returns an error.

CGD3D10ERR_INVALIDPROFILE is returned if program’s profile is not a supported D3D profile.

CGD3D10ERR_NODEVICE is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D10SetDevice.

HISTORY

cgD3D10LoadProgram was introduced in Cg 2.1.
7.19 cgD3D10RegisterStates

NAME

cgD3D10RegisterStates - registers graphics pass states for CgFX files

SYNOPSIS

#include <Cg/cgD3D10.h>

void cgD3D10RegisterStates( CGcontext context );

PARAMETERS

context

The context in which to register the states.

RETURN VALUES

None.

DESCRIPTION

cgD3D10RegisterStates registers a set of states for passes in techniques in CgFX effect files. These states correspond to the set of D3D states that is relevant and/or useful to be set in passes in effect files. See the Cg User’s Guide for complete documentation of the states that are made available after calling cgD3D10RegisterStates.

EXAMPLES

// register D3D10 states for this context

CGcontext context = cgCreateContext();
cgD3D10RegisterStates(context);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D10RegisterStates was introduced in Cg 2.1.

Starting with Cg 2.2, cgD3D10RegisterStates calls cgSetStateLatestProfile for program states it creates and registers the latest profile returned by cgD3D10GetLatestVertexProfile, cgD3D10GetLatestGeometryProfile or cgD3D10GetLatestPixelProfile for the appropriate program domain.

SEE ALSO

cgCreateState, cgSetStateLatestProfile, cgSetPassState, cgResetPassState, cgCallStateValidateCallback, cgGLRegisterStates, cgD3D9RegisterStates
7.20 cgD3D10SetDevice

NAME
cgD3D10SetDevice - set the D3D device

SYNOPSIS

```
#include <Cg/cgD3D10.h>

HRESULT cgD3D10SetDevice( CGcontext context,
                          ID3D10Device * device );
```

PARAMETERS

- **context**
  - The context in which to set the current device.

- **device**
  - Pointer to an ID3D10Device interface that the expanded interface will use for any D3D-specific routine it may call. This parameter can be NULL to free all D3D resources used by the expanded interface and remove its reference to the D3D device.

RETURN VALUES

Returns D3D_OK if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION

cgD3D10SetDevice informs the expanded interface of the new D3D device. This will destroy any D3D resources for programs previously loaded with cgD3D10LoadProgram and use the new D3D device to recreate them. The expanded interface will increment the reference count to the D3D device, so this function must eventually be called with NULL to release that reference so D3D can be properly shut down.

If **device** is NULL, all D3D resources for programs previously loaded with cgD3D10LoadProgram are destroyed. However, these programs are still considered managed by the expanded interface, so if a new D3D device is set later these programs will be recreated using the new D3D device.

If a new device is being set, all D3D resources for programs previously loaded with cgD3D10LoadProgram are rebuilt using the new device. All shadowed parameters for these programs are maintained across D3D device changes except texture parameters. Since textures in D3D are bound to a particular D3D device, these resources cannot be saved across device changes. When these textures are recreated for the new D3D device, they must be re-bound to the sampler parameter.

Note that calling cgD3D10SetDevice(NULL) does not destroy any core runtime resources (CGprograms, CGparameters, etc.) used by the expanded interface. These must be destroyed separately using cgDestroyProgram and cgDestroyContext.

EXAMPLES

```
// pDev is an ID3D10Device interface initialized elsewhere
int main()
{
    cgD3D10SetDevice(pDev);
}
```

ERRORS

cgD3D10Failed is generated if a D3D function returns an error.
HISTORY
cgd3d10SetDevice was introduced in Cg 2.1.

SEE ALSO
cgd3d10GetDevice, cgDestroyProgram, cgDestroyContext, cgd3d10LoadProgram

7.21 cgD3D10SetManageTextureParameters

NAME
cgD3D10SetManageTextureParameters - set the manage texture parameters flag for a context

SYNOPSIS
#include <Cg/cgD3D10.h>

void cgD3D10SetManageTextureParameters( CGcontext context,
                                      CGbool flag );

PARAMETERS
  context
    The context in which the automatic texture management behavior will be changed.
  flag
    A boolean switch which controls automatic texture management by the runtime.

RETURN VALUES
None.

DESCRIPTION
By default, cgD3D10 does not manage any texture state in D3D. It is up to the user to enable and disable textures using D3D. This behavior is the default to avoid conflicts with texture state on geometry that’s rendered with the fixed function pipeline or without cgD3D10.

If automatic texture management is desired, cgD3D10SetManageTextureParameters may be called with flag set to CG_TRUE before cgD3D10BindProgram is called. Whenever cgD3D10BindProgram is called, the cgD3D10 runtime will make all the appropriate texture parameter calls on the application’s behalf.

Calling cgD3D10SetManageTextureParameters with flag set to CG_FALSE will disable automatic texture management.

NOTE: When cgD3D10SetManageTextureParameters is set to CG_TRUE, applications should not make texture state change calls to D3D after calling cgD3D10BindProgram, unless the application is trying to override some parts of cgD3D10’s texture management.

EXAMPLES
// Enable automatic texture management
cgd3D10SetManageTextureParameters( pCtx, CG_TRUE );

ERRORS
None.
7.22 cgD3D10SetSamplerStateParameter

NAME

cgD3D10SetSamplerStateParameter - Sets a sampler state object to a Cg sampler parameter.

SYNOPSIS

```c
#include <Cg/cgD3D10.h>

void cgD3D10SetSamplerStateParameter( CGparameter param,
                                        ID3D10SamplerState * samplerState );
```

PARAMETERS

- param
  - The sampler parameter whose state is to be set.
- samplerState
  - The D3D sampler state to set.

RETURN VALUES

None.

DESCRIPTION

`cgD3D10SetSamplerStateParameter` sets the sampler state associated with a sampler parameter. NULL means default sampler state.

EXAMPLES

```c
ID3D10SamplerState * sstate = NULL; Device->CreateSamplerState( &samplerDesc, &sstate );
cgD3D10SetSamplerStateParameter( myCgSamplerParam, sstate );
```

ERRORS

`CG_INVALID_PARAMETER_ERROR` is generated if `param` is invalid.

HISTORY

`cgD3D10SetSamplerStateParameter` was introduced in Cg 2.1.

SEE ALSO

`cgD3D10GetTextureSamplerStateParameter`
7.23 cgD3D10SetTextureParameter

NAME

cgD3D10SetTextureParameter - sets the value of a texture parameter

SYNOPSIS

#include <Cg/cgD3D10.h>

void cgD3D10SetTextureParameter( CGparameter param,
                                      ID3D10Resource * texture );

PARAMETERS

param
  The texture parameter that will be set.
texture
  An D3D texture to which the parameter will be set.

RETURN VALUES

None.

DESCRIPTION

cgD3D10SetTextureParameter sets the value of a texture parameter to a given D3D10 texture object.

EXAMPLES

ID3D10Resource *myTexture;
// Assume myTexture is loaded here...

// .. and param is an effect sampler parameter
cgD3D10SetTextureParameter( param, myTexture );

ERRORS

CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter handle.

HISTORY

cgD3D10SetTextureParameter was introduced in Cg 2.1.

SEE ALSO

cgD3D10SetManageTextureParameters

7.24 cgD3D10SetTextureSamplerStateParameter

NAME

cgD3D10SetTextureSamplerStateParameter - Sets a texture resource and sampler state object to a Cg sampler parameter.
SYNOPSIS

```c
#include <Cg/cg.h>

void cgD3D10SetTextureSamplerStateParameter( CGparameter param,
                                          ID3D10Resource * texture,
                                          ID3D10SamplerState * samplerState );
```

PARAMETERS

- **param**
  - The sampler parameter whose texture and state is to be set.
- **texture**
  - The texture resource object being set.
- **samplerState**
  - The sampler state object being set.

RETURN VALUES

None.

DESCRIPTION

`cgD3D10SetTextureSamplerStateParameter` accomplishes the same thing as calling both `cgD3D10SetTextureParameter` and `cgD3D10SetSamplerStateParameter` together.

EXAMPLES

```c
ID3D10Resource * myTexture; ID3D10SamplerState * mySamplerState;
Device->CreateTexture2D( &desc, &initalData, &myTexture ); Device->CreateSamplerState( &desc, &mySamplerState );
cgD3D10SetTextureSamplerStateParameter( myCgSamplerParam, myTexture, mySamplerState );
```

ERRORS

**CG_INVALID_PARAMETER_ERROR** is generated if `param` is invalid.

HISTORY

`cgD3D10SetTextureSamplerStateParameter` was introduced in Cg 2.1.

SEE ALSO

`cgD3D10SetTextureParameter`, `cgD3D10SetSamplerStateParameter`

7.25 **cgD3D10TranslateCGerror**

NAME

`cgD3D10TranslateCGerror` - convert a Cg runtime error into a string
SYNOPSIS

#include <Cg/cgD3D10.h>

const char * cgD3D10TranslateCGerror( CGerror error );

PARAMETERS

error

The error code to translate. Can be a core runtime error or a D3D runtime error.

RETURN VALUES

Returns a pointer to a string describing error.

DESCRIPTION

cgD3D10TranslateCGerror converts a Cg runtime error into a string. This routine should be called instead of the core runtime routine cgGetErrorString because it will also translate errors that the Cg D3D runtime generates.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCallback.

EXAMPLES

char buf[512];
CGerror error = cgGetLastError();
if (error != CG_NO_ERROR)
{
    sprintf(buf, "An error occurred. Error description: '%s'\n",
            cgD3D10TranslateCGerror(error));
    OutputDebugString(buf);
}

ERRORS

None.

HISTORY

cgD3D10TranslateCGerror was introduced in Cg 2.1.

SEE ALSO

cgGetErrorString, cgSetErrorCallback

7.26 cgD3D10TranslateHRESULT

NAME

cgD3D10TranslateHRESULT - convert an HRESULT into a string

SYNOPSIS

#include <Cg/cgD3D10.h>

const char * cgD3D10TranslateHRESULT( HRESULT hr );
PARAMETERS

hr

The HRESULT to translate. Can be a generic HRESULT or a D3D runtime error.

RETURN VALUES

Returns a pointer to a string describing the error.

DESCRIPTION

cgD3D10TranslateHRESULT converts an HRESULT into a string. This routine should be called instead of 
DXGetErrorDescription10 because it will also translate errors that the Cg D3D runtime generates.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCall-
back.

EXAMPLES

char buf[512];
HRESULT hres = cgD3D10GetLastError();
if (FAILED(hres))
{
    sprintf(buf, "A D3D error occurred. Error description: '%s'

OutputDebugString(buf);
}

ERRORS

None.

HISTORY

cgD3D10TranslateHRESULT was introduced in Cg 2.1.

SEE ALSO

cgD3D10TranslateCGerror, cgGetErrorString, cgSetErrorCallback

7.27 cgD3D10TypeToSize

NAME

cgD3D10TypeToSize - get the size of a CGtype enumerated type

SYNOPSIS

#include <Cg/cgD3D10.h>

DWORD cgD3D10TypeToSize( CGtype type );

PARAMETERS

    type

    Member of the CGtype enumerated type whose size is to be returned.
RETURN VALUES

Returns the size of \texttt{type} in terms of consecutive floating point values.
Returns \texttt{0} if the type does not have an inherent size. Sampler types fall into this category.

DESCRIPTION

cgD3D10TypeToSize retrieves the size of a \texttt{CGtype} enumerated type in terms of consecutive floating point values.
If the type does not have an inherent size, the return value is \texttt{0}. Sampler types fall into this category.

EXAMPLES

// param is a CGparameter initialized earlier
...
DWORD size = cgD3D10TypeToSize(cgGetParameterType(param));

// (sanity check that parameters have the expected size)
...
assert(cgD3D10TypeToSize(cgGetParameterType(vsModelView)) == 16);
assert(cgD3D10TypeToSize(cgGetParameterType(psColor)) == 4);

ERRORS

None.

HISTORY

cgD3D10TypeToSize was introduced in Cg 2.1.

SEE ALSO

cgD3D10SetDevice

7.28 cgD3D10UnbindProgram

NAME

cgD3D10UnbindProgram - Unbinds a D3D10 program.

SYNOPSIS

#include <Cg/cgD3D10.h>

\texttt{void cgD3D10UnbindProgram( CGprogram Program );}

PARAMETERS

Program

A program whose profile is used to unbind the program from the API.

RETURN VALUES

None.

DESCRIPTION

cgD3D10UnbindProgram Unbinds a D3D10 program from the profile of the given program ‘Program.’
EXAMPLES

// CGprogram vertexProg;
//...
cgD3D10UnbindProgram( vertexProg );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if Program is not a valid program.

HISTORY

cgD3D10UnbindProgram was introduced in Cg 3.0.

SEE ALSO

cgD3D10BindProgram

7.29 cgD3D10UnloadProgram

NAME

cgD3D10UnloadProgram - Unloads a D3D shader from the runtime data structures

SYNOPSIS

#include <Cg/cgD3D10.h>

void cgD3D10UnloadProgram( CGprogram Program );

PARAMETERS

Program

A program whose compiled output is used to create the D3D shader.

RETURN VALUES

None.

DESCRIPTION

cgD3D10UnloadProgram Unloads a D3D shader from the runtime data structures.

EXAMPLES

// vertexProg is a CGprogram using a vertex profile
...
cgD3D10UnloadProgram( vertexProg );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if Program is not a valid program.

HISTORY

cgD3D10UnloadProgram was introduced in Cg 2.1.
SEE ALSO

cgD3D10LoadProgram
8.1 cgD3D11BindProgram

NAME
cgD3D11BindProgram - activate a program with D3D

SYNOPSIS
#include <Cg/cgD3D11.h>

HRESULT cgD3D11BindProgram( CGprogram program );

PARAMETERS
program
The program to activate with D3D.

RETURN VALUES
Returns D3D_OK if the function succeeds.
Returns the D3D failure code if the function fails due to a D3D call.

DESCRIPTION
cgD3D11BindProgram activates a program with D3D. The program is activated using
ID3D11DeviceContext::VSSetShader or ID3D11DeviceContext::PSSetShader depending on the program’s
profile type.
D3D allows only one vertex shader and one pixel shader to be active at any given time, so activating a program of a
given type implicitly deactivates any other program of a that type.

EXAMPLES

// vertexProg and pixelProg are CGprograms initialized elsewhere
// pDevContext is an ID3D11DeviceContext interface initialized elsewhere

HRESULT hr = cgD3D11BindProgram(vertexProg);
HRESULT hr2 = cgD3D11BindProgram(pixelProg);
ERRORS

E_FAIL is returned if program was not loaded with the cgD3D11LoadProgram.

E_FAIL is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with cgD3D11SetDevice.

HISTORY

cgD3D11BindProgram was introduced in Cg 3.0.

SEE ALSO

cgD3D11LoadProgram, cgD3D11SetTextureParameter

8.2 cgD3D11CreateBuffer

NAME

cgD3D11CreateBuffer - create a D3D11 buffer object

SYNOPSIS

#include <Cg/cgD3D11.h>

CGbuffer cgD3D11CreateBuffer( CGcontext context,
                            int size,
                            const void *data,
                            D3D11_USAGE bufferUsage );

PARAMETERS

    context
    The context to which the new buffer will be added.

    size
    The length in bytes of the buffer to create.

    data
    The initial data to be copied into the buffer. NULL will fill the buffer with zero.

    bufferUsage
    A D3D11 usage flag as specified by D3D11_USAGE.

RETURN VALUES

    Returns a CGbuffer handle on success.
    Returns NULL if any error occurs.

DESCRIPTION

cgD3D11CreateBuffer creates a D3D11 buffer object.

EXAMPLES

CGbuffer myBuffer = cgD3D11CreateBuffer( myCgContext, sizeof( float ) * 16,
                                         myData, D3D11_USAGE_DEFAULT );
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.
CG_MEMORY_ALLOC_ERROR is generated if a buffer couldn’t be created.

HISTORY

cgD3D11CreateBuffer was introduced in Cg 3.1.

SEE ALSO

cgCreateBuffer, cgD3D10CreateBuffer, cgGLCreateBuffer, cgD3D11CreateBufferFromObject,
cgD3D11GetBufferObject

8.3 cgD3D11CreateBufferFromObject

NAME

cgD3D11CreateBufferFromObject - create a Cg buffer from a D3D11 buffer object

SYNOPSIS

#include <Cg/cgD3D11.h>

CGbuffer cgD3D11CreateBufferFromObject( CGcontext context,
                                        ID3D11Buffer * obj,
                                        CGbool manageObject );

PARAMETERS

context
    The context to which the new buffer will be added.

obj
    A D3D11 buffer object created by the application.

manageObject
    A boolean switch which controls whether obj will be deleted by the runtime when the CGbuffer object
    returned by cgD3D11CreateBufferFromObject is destroyed.

RETURN VALUES

Returns a CGbuffer handle on success.
Returns NULL if any error occurs.

DESCRIPTION

cgD3D11CreateBufferFromObject creates a Cg buffer from a preexisting D3D11 buffer object. This D3D11 object
will be deleted by the runtime when the Cg buffer is destroyed if manageObject is CG_TRUE. Otherwise the
application is responsible for deleting the D3D11 object obj.

EXAMPLES

CGbuffer myBuffer = cgD3D11CreateBufferFromObject( myCgContext, d3d11BufferObj, CG_TRUE );
ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D11CreateBufferFromObject was introduced in Cg 3.1.

SEE ALSO

cgCreateBuffer, cgDestroyBuffer, cgD3D11GetBufferByIndex

8.4 cgD3D11GetBufferByIndex

NAME

cgD3D11GetBufferByIndex - Returns a pointer to an ID3D11Buffer interface by the constant buffer index.

SYNOPSIS

#include <Cg/cgD3D11.h>

ID3D11Buffer * cgD3D11GetBufferByIndex( CGprogram Program,
                                        UINT Index );

PARAMETERS

Program

The Cg program containing the buffer.

Index

A zero-based index.

RETURN VALUES

Returns a pointer to an ID3D11Buffer interface containing the constant buffer.

DESCRIPTION

cgD3D11GetBufferByIndex returns a pointer to an ID3D11Buffer interface containing the constant buffer for manual manipulation. If the user manually changes the constant values in this way, the constant values contained in the corresponding CGbuffer (if exists) will be stale.

EXAMPLES

ID3D11Buffer * myConstantBuffer = cgD3D11GetBufferByIndex( myCgProgram, 0 );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR if Program is not a valid Cg program.

HISTORY

cgD3D11GetBufferByIndex was introduced in Cg 3.0.

SEE ALSO

cgCreateBuffer
8.5 cgD3D11GetBufferObject

NAME
cgD3D11GetBufferObject - get the D3D11 buffer object for a buffer

SYNOPSIS
#include <Cg/cgD3D11.h>

ID3D11Buffer * cgD3D11GetBufferObject( CGbuffer buffer );

PARAMETERS
buffer
The buffer for which the associated D3D11 buffer object will be retrieved.

RETURN VALUES
Returns the D3D11 buffer object associated with buffer.
Returns NULL if any error occurs.

DESCRIPTION
cgD3D11GetBufferObject returns the D3D11 buffer object associated with a buffer.

EXAMPLES
ID3D11Buffer * id = cgD3D11GetBufferObject( myBuffer );

ERRORS
CG_INVALID_BUFFER_HANDLE_ERROR is generated if buffer is not a valid buffer.

HISTORY
cgD3D11GetBufferObject was introduced in Cg 3.1.

SEE ALSO
cgCreateBuffer, cgD3D11CreateBuffer, cgD3D11CreateBufferFromObject

8.6 cgD3D11GetCompiledProgram

NAME
cgD3D11GetCompiledProgram - Gets the compiled shader as a ID3DBlob returned from Direct3D after cgD3D11LoadProgram is called.

SYNOPSIS
#include <Cg/cgD3D11.h>

ID3DBlob * cgD3D11GetCompiledProgram( CGprogram program );
PARAMETERS

program

The program handle after a call to cgD3D11LoadProgram has been made.

RETURN VALUES

Returns a pointer to a ID3DBlob object containing the compiled shader code.

DESCRIPTION

cgD3D11GetCompiledProgram allows the user to get back the compiled shader from Direct3D once cgD3D11LoadProgram has been called.

EXAMPLES

CGprogram myCgProgram = cgCreateProgram( ... ); cgD3D11LoadProgram( myCgProgram, 0 );
ID3DBlob * obj = cgD3D11GetCompiledProgram( myCgProgram );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if the program is invalid.

HISTORY

cgD3D11GetCompiledProgram was introduced in Cg 3.0.

SEE ALSO

cgD3D11LoadProgram

8.7 cgD3D11GetDevice

NAME

cgD3D11GetDevice - retrieves the current D3D11 device associated with a context

SYNOPSIS

#include <Cg/cgD3D11.h>

ID3D11Device * cgD3D11GetDevice( CGcontext context );

PARAMETERS

context

The context from which to get the current device.

RETURN VALUES

Returns the current D3D11 device associated with context.

DESCRIPTION

cgD3D11GetDevice retrieves the current D3D11 device associated with context. Note that the returned device pointer may be NULL.
EXAMPLES

ID3D11Device* curDevice = cgD3D11GetDevice(ctx);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D11GetDevice was introduced in Cg 3.0.

SEE ALSO

cgD3D11SetDevice

8.8 cgD3D11GetIASignatureByPass

NAME

cgD3D11GetIASignatureByPass - Gets the compiled vertex shader signature as a ID3DBlob from a pass of a validated technique.

SYNOPSIS

#include <Cg/cgD3D11.h>

ID3DBlob * cgD3D11GetIASignatureByPass( CGpass pass );

PARAMETERS

pass

The pass handle after validation of a CgFX technique.

RETURN VALUES

Returns a pointer to a ID3DBlob object containing the vertex shader signature.

Returns NULL if the program was not loaded.

DESCRIPTION

cgD3D11GetIASignatureByPass allows the user to get back the vertex shader signature of a pass of a validated CgFX technique.

EXAMPLES

myCgEffect = cgCreateEffectFromFile( myCgContext, "effect.cgfx", NULL );
myCgTechnique = cgGetFirstTechnique( myCgEffect );

if( cgValidateTechnique( myCgTechnique ) != CG_FALSE )
{
    const D3D11_INPUT_ELEMENT_DESC layout[] =
    {
        { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D11_INPUT_PER_VERTEX_DATA, 0 },
    };

    CGpass myPass = cgGetFirstPass( myCgTechnique );
ID3DBlob * pVSBuf = cgD3D11GetIASignatureByPass( myPass );

hr = pDevice->CreateInputLayout( layout, 1, pVSBuf->GetBufferPointer(),
    pVSBuf->GetBufferSize(), &g_pVertexLayout );
}

ERRORS

CG_INVALID_PASS_HANDLE_ERROR is generated if the pass is invalid.

HISTORY

cgD3D11GetIASignatureByPass was introduced in Cg 3.0.

SEE ALSO

cgD3D11GetCompiledProgram

8.9 cgD3D11GetLastError

NAME

cgD3D11GetLastError - get the last D3D error that occurred

SYNOPSIS

#include <Cg/cgD3D11.h>

HRESULT cgD3D11GetLastError( void );

PARAMETERS

None.

RETURN VALUES

Returns the last D3D error that occurred during an expanded interface function call.

Returns D3D_OK if no D3D error has occurred since the last call to cgD3D11GetLastError.

DESCRIPTION

cgD3D11GetLastError retrieves the last D3D error that occurred during an expanded interface function call. The last error is always cleared immediately after the call.

EXAMPLES

HRESULT lastError = cgD3D11GetLastError();

ERRORS

None.

HISTORY

cgD3D11GetLastError was introduced in Cg 3.0.
SEE ALSO

cgD3D11TranslateHRESULT

8.10 cgD3D11GetLatestDomainProfile

NAME
cgD3D11GetLatestDomainProfile - get the latest supported domain shader version

SYNOPSIS

#include <Cg/cgD3D11.h>

CGprofile cgD3D11GetLatestDomainProfile( void );

PARAMETERS
None.

RETURN VALUES
Returns the latest domain shader version supported by the current D3D device.
Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION
cgD3D11GetLatestDomainProfile retrieves the latest domain shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 3.0, if the environment variable CGD3D11_LATEST_DOMAIN_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGprofile value that will be returned by cgD3D11GetLatestDomainProfile.

EXAMPLES

CGprofile bestDomainProfile = cgD3D11GetLatestDomainProfile();

ERRORS
None.

HISTORY
cgD3D11GetLatestDomainProfile was introduced in Cg 3.0.

SEE ALSO
8.11 cgD3D11GetLatestGeometryProfile

NAME
cgD3D11GetLatestGeometryProfile - get the latest supported geometry shader version

SYNOPSIS
#include <Cg/cgD3D11.h>
CGprofile cgD3D11GetLatestGeometryProfile( void );

PARAMETERS
None.

RETURN VALUES
Returns the latest geometry shader version supported by the current D3D device.
Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION
cgD3D11GetLatestGeometryProfile retrieves the latest geometry shader version that the current D3D device sup-
ports. This is an expanded interface function because it needs to know about the D3D device to determine the most
current version supported.

Starting in Cg 3.0, if the environment variable CGD3D11_LATEST_GEOMETRY_PROFILE is set in the application’s
environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it
is this CGprofile value that will be returned by cgD3D11GetLatestGeometryProfile.

EXAMPLES
CGprofile bestGeometryProfile = cgD3D11GetLatestGeometryProfile();

ERRORS
None.

HISTORY
cgD3D11GetLatestGeometryProfile was introduced in Cg 3.0.

SEE ALSO
cgD3D11GetLatestDomainProfile, cgD3D11GetLatestHullProfile, cgD3D11GetLatestPixelProfile,
cgD3D11GetLatestVertexProfile

8.12 cgD3D11GetLatestHullProfile

NAME
cgD3D11GetLatestHullProfile - get the latest supported hull shader version
SYNOPSIS

#include <Cg/cgD3D11.h>

CGprofile cgD3D11GetLatestHullProfile( void );

PARAMETERS

None.

RETURN VALUES

Returns the latest hull shader version supported by the current D3D device.

Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION

cgD3D11GetLatestHullProfile retrieves the latest hull shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 3.0, if the environment variable CGD3D11_LATEST_HULL_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGprofile value that will be returned by cgD3D11GetLatestHullProfile.

EXAMPLES

CGprofile bestHullProfile = cgD3D11GetLatestHullProfile();

ERRORS

None.

HISTORY

cgD3D11GetLatestHullProfile was introduced in Cg 3.0.

SEE ALSO


8.13 cgD3D11GetLatestPixelProfile

NAME

cgD3D11GetLatestPixelProfile - get the latest supported pixel shader version

SYNOPSIS

#include <Cg/cgD3D11.h>

CGprofile cgD3D11GetLatestPixelProfile( void );

PARAMETERS

None.
RETURN VALUES

Returns the latest pixel shader version supported by the current D3D device. Returns CG_PROFILE_UNKNOWN if no D3D device is currently set.

DESCRIPTION

cgD3D11GetLatestPixelProfile retrieves the latest pixel shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 3.0, if the environment variable CGD3D11_LATEST PIXEL_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE UNKNOWN), then it is this CGprofile value that will be returned by cgD3D11GetLatestPixelProfile.

EXAMPLES

CGprofile bestPixelProfile = cgD3D11GetLatestPixelProfile();

ERRORS

None.

HISTORY

cgD3D11GetLatestPixelProfile was introduced in Cg 3.0.

SEE ALSO

cgD3D11GetLatestDomainProfile, cgD3D11GetLatestGeometryProfile, cgD3D11GetLatestHullProfile, cgD3D11GetLatestVertexProfile

8.14 cgD3D11GetLatestVertexProfile

NAME

cgD3D11GetLatestVertexProfile - get the latest supported vertex shader version

SYNOPSIS

#include <Cg/cgD3D11.h>

CGprofile cgD3D11GetLatestVertexProfile( void );

PARAMETERS

None.

RETURN VALUES

Returns the latest vertex shader version supported by the current D3D device. Returns CG_PROFILE UNKNOWN if no D3D device is currently set.
DESCRIPTION

cgD3D11GetLatestVertexProfile retrieves the latest vertex shader version that the current D3D device supports. This is an expanded interface function because it needs to know about the D3D device to determine the most current version supported.

Starting in Cg 3.0, if the environment variable CGD3D11_LATEST_VERTEX_PROFILE is set in the application’s environment to a string that cgGetProfile translates to a valid profile (meaning not CG_PROFILE_UNKNOWN), then it is this CGProfile value that will be returned by cgD3D11GetLatestVertexProfile.

EXAMPLES

CGProfile bestVertexProfile = cgD3D11GetLatestVertexProfile();

ERRORS

None.

HISTORY

cgD3D11GetLatestVertexProfile was introduced in Cg 3.0.

SEE ALSO


8.15 cgD3D11GetManageTextureParameters

NAME

cgD3D11GetManageTextureParameters - get the manage texture parameters flag from a context

SYNOPSIS

#include <Cg/cgD3D11.h>

CGbool cgD3D11GetManageTextureParameters( CGcontext context );

PARAMETERS

context

The context from which the automatic texture management setting will be retrieved.

RETURN VALUES

Returns the manage texture management flag from context.

DESCRIPTION

cgD3D11GetManageTextureParameters returns the manage texture management flag from context. See cgD3D11SetManageTextureParameters for more information.

EXAMPLES

CGbool manage = cgD3D11GetManageTextureParameters( pCtx );

if( manage )
    doSomething();
ERRORS
None.

HISTORY

cgD3D11GetManageTextureParameters was introduced in Cg 3.0.

SEE ALSO

cgD3D11SetManageTextureParameters

8.16 cgD3D11GetOptimalOptions

NAME

cgD3D11GetOptimalOptions - get the best set of compiler options for a profile

SYNOPSIS

#include <Cg/cgD3D11.h>

char const ** cgD3D11GetOptimalOptions( CGprofile profile );

PARAMETERS

profile

The profile whose optimal arguments are requested.

RETURN VALUES

Returns a null-terminated array of strings representing the optimal set of compiler options for profile.

Returns NULL if no D3D device is currently set.

DESCRIPTION

cgD3D11GetOptimalOptions returns the best set of compiler options for a given profile. This is an expanded interface function because it needs to know about the D3D device to determine the most optimal options.

The elements of the returned array are meant to be used as part of the args parameter to cgCreateProgram or cgCreateProgramFromFile.

The returned string does not need to be destroyed by the application. However, the contents could change if the function is called again for the same profile but a different D3D device.

EXAMPLES

const char* vertOptions[] = { myCustomArgs,
   cgD3D11GetOptimalOptions(vertProfile),
   NULL };

// create the vertex shader
CGprogram myVS = cgCreateProgramFromFile( context,
   CG_SOURCE,
   "vshader.cg",
   vertProfile,
   "VertexShader",
   vertOptions);
ERRORS
None.

HISTORY

cgD3D11GetOptimalOptions was introduced in Cg 3.0.

SEE ALSO

cgD3D11GetLatestVertexProfile, cgD3D11GetLatestPixelProfile, cgCreateProgram, cgCreateProgramFromFile

8.17 cgD3D11GetProgramErrors

NAME
cgD3D11GetProgramErrors - Gets a list of errors returned from Direct3D if the program did not load.

SYNOPSIS

#include <Cg/cgD3D11.h>

ID3D10Blob * cgD3D11GetProgramErrors( CGprogram program );

PARAMETERS

program
The program handle after a call to cgD3D11LoadProgram has been made.

RETURN VALUES

Returns a pointer to a ID3DBlob object containing a list of errors if the program did not load.
Returns NULL if the program was loaded.

DESCRIPTION

cgD3D11GetProgramErrors allows the user to get back the compiled shader from Direct3D once

cgD3D11LoadProgram has been called.

EXAMPLES

CGprogram myCgProgram = cgCreateProgram( ... ); cgD3D11LoadProgram( myCgProgram, 0 );

ID3DBlob * err = cgD3D11GetProgramErrors( myCgProgram );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if the program is invalid;

HISTORY

cgD3D11GetProgramErrors was introduced in Cg 3.0.

SEE ALSO


cgD3D11LoadProgram
8.18 cgD3D11IsProfileSupported

NAME
cgD3D11IsProfileSupported - determine if a profile is supported by cgD3D11

SYNOPSIS
#include <Cg/cgD3D11.h>

CGbool cgD3D11IsProfileSupported( CGprofile profile );

PARAMETERS

profile
The profile which will be checked for support.

RETURN VALUES

Returns CG_TRUE if profile is supported by the cgD3D10 library.
Returns CG_FALSE otherwise.

However if cgD3D11SetDevice has not been called to register a ID3D11Device device yet, this routine returns CG_TRUE for all valid D3D11 profiles.

DESCRIPTION
cgD3D11IsProfileSupported returns CG_TRUE if the profile indicated by profile is supported by the cgD3D11 library.

EXAMPLES

// assuming the program requires Shader Model 5.0 ...

if (!cgD3D11IsProfileSupported(CG_PROFILE_VS_5_0)) || (!cgD3D11IsProfileSupported(CG_PROFILE_PS_5_0)) {
    fprintf(stderr, "Sorry, required profiles not supported on this system.\n");
    exit(1);
}

ERRORS

None.

HISTORY

cgD3D11IsProfileSupported was introduced in Cg 3.0.

SEE ALSO

cgD3D11GetLatestPixelProfile, cgD3D11GetLatestVertexProfile

8.19 cgD3D11IsProgramLoaded

NAME
cgD3D11IsProgramLoaded - determine if a program has been loaded
SYNOPSIS
#include <Cg/cgD3D11.h>

CGbool cgD3D11IsProgramLoaded( CGprogram program );

PARAMETERS
  program
    The program which will be checked.

RETURN VALUES
Returns CG_TRUE if program has been loaded using cgD3D11LoadProgram.
Returns CG_FALSE otherwise.

DESCRIPTION
cgD3D11IsProgramLoaded determines if a program has been loaded using cgD3D11LoadProgram.

EXAMPLES
// program is a CGprogram initialized elsewhere
...  
CGbool isLoaded = cgD3D11IsProgramLoaded(program);

ERRORS
None.

HISTORY
cgD3D11IsProgramLoaded was introduced in Cg 3.0.

SEE ALSO
cgD3D11LoadProgram

8.20 cgD3D11LoadProgram

NAME
cgD3D11LoadProgram - create a D3D shader and enable the expanded interface routines

SYNOPSIS
#include <Cg/cgD3D11.h>

HRESULT cgD3D11LoadProgram( CGprogram program,
    UINT flags );

PARAMETERS
  program
    A program whose compiled output is used to create the D3D shader.
  flags
The flags to pass to **D3DCompile**. See the D3D documentation for a list of valid flags.

**RETURN VALUES**

Returns **D3D_OK** if the function succeeds or **program** has already been loaded.

Returns the D3D failure code if the function fails due to a D3D call.

**DESCRIPTION**

**cgD3D11LoadProgram** creates a D3D shader for a program and enables use of expanded interface routines for that program.

**cgD3D11LoadProgram** compiles the compiled Cg output for **program** using **D3DCompile** and then creates a D3D shader using **ID3D11Device::CreateVertexShader** or **ID3D11Device::CreatePixelShader** depending on the program’s profile.

The D3D shader handle is not returned. If the shader handle is desired by the application, the expanded interface should not be used for that program.

**EXAMPLES**

```c
// vertexProg is a CGprogram using a vertex profile
// pixelProg is a CGprogram using a pixel profile
...
HRESULT hr1 = cgD3D11LoadProgram(vertexProg, 0);
HRESULT hr2 = cgD3D11LoadProgram(pixelProg, 0);
```

**ERRORS**

**E_FAIL** is generated if a D3D function returns an error.

**E_FAIL** is returned if **program**’s profile is not a supported D3D profile.

**E_FAIL** is returned if a required D3D device is NULL. This usually occurs when an expanded interface routine is called but a D3D device has not been set with **cgD3D11SetDevice**.

**HISTORY**

**cgD3D11LoadProgram** was introduced in Cg 3.0.

**SEE ALSO**

**cgD3D11SetDevice**

### 8.21 cgD3D11RegisterStates

**NAME**

**cgD3D11RegisterStates** - registers graphics pass states for CgFX files

**SYNOPSIS**

```c
#include <Cg/cgD3D11.h>

void cgD3D11RegisterStates( CGcontext context );
```
PARAMETERS

context

The context in which to register the states.

RETURN VALUES

None.

DESCRIPTION

cgD3D11RegisterStates registers a set of states for passes in techniques in CgFX effect files. These states correspond to the set of D3D states that is relevant and/or useful to be set in passes in effect files. See the Cg User’s Guide for complete documentation of the states that are made available after calling cgD3D11RegisterStates.

EXAMPLES

// register D3D11 states for this context
CGcontext context = cgCreateContext();
cgD3D11RegisterStates(context);

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if context is not a valid context.

HISTORY

cgD3D11RegisterStates was introduced in Cg 3.0.

Starting with Cg 3.0, cgD3D11RegisterStates calls cgSetStateLatestProfile for program states it creates and registers the latest profile returned by cgD3D11GetLatestVertexProfile, cgD3D11GetLatestGeometryProfile or cgD3D11GetLatestPixelProfile for the appropriate program domain.

SEE ALSO

cgCreateState, cgSetStateLatestProfile, cgSetPassState, cgResetPassState, cgCallStateValidateCallback, cgGLRegisterStates, cgD3D9RegisterStates

8.22 cgD3D11SetDevice

NAME

cgD3D11SetDevice - set the D3D device

SYNOPSIS

#include <Cg/cgD3D11.h>

HRESULT cgD3D11SetDevice( CGcontext context,
                          ID3D11Device * device );

PARAMETERS

context

The context in which to set the current device.

device
Pointer to an **ID3D11Device** interface that the expanded interface will use for any D3D-specific routine it may call. This parameter can be **NULL** to free all D3D resources used by the expanded interface and remove its reference to the D3D device.

**RETURN VALUES**

Returns **D3D_OK** if the function succeeds.

Returns the D3D failure code if the function fails due to a D3D call.

**DESCRIPTION**

**cgD3D11SetDevice** informs the expanded interface of the new D3D device. This will destroy any D3D resources for programs previously loaded with **cgD3D11LoadProgram** and use the new D3D device to recreate them. The expanded interface will increment the reference count to the D3D device, so this function must eventually be called with **NULL** to release that reference so D3D can be properly shut down.

If **device** is **NULL**, all D3D resources for programs previously loaded with **cgD3D11LoadProgram** are destroyed. However, these programs are still considered managed by the expanded interface, so if a new D3D device is set later these programs will be recreated using the new D3D device.

If a new device is being set, all D3D resources for programs previously loaded with **cgD3D11LoadProgram** are rebuilt using the new device. All shadowed parameters for these programs are maintained across D3D device changes except texture parameters. Since textures in D3D are bound to a particular D3D device, these resources cannot be saved across device changes. When these textures are recreated for the new D3D device, they must be re-bound to the sampler parameter.

Note that calling **cgD3D11SetDevice(NULL)** does not destroy any core runtime resources (**CGprograms**, **CGparameters**, etc.) used by the expanded interface. These must be destroyed seperately using **cgDestroyProgram** and **cgDestroyContext**.

**EXAMPLES**

```
// pDev is an ID3D11Device interface initialized elsewhere
...
cgD3D11SetDevice(pDev);
```

**ERRORS**

**E_FAIL** is returned if a D3D function returns an error.

**HISTORY**

**cgD3D11SetDevice** was introduced in Cg 3.0.

**SEE ALSO**

**cgD3D11GetDevice**, **cgDestroyProgram**, **cgDestroyContext**, **cgD3D11LoadProgram**

**8.23 cgD3D11SetManageTextureParameters**

**NAME**

**cgD3D11SetManageTextureParameters** - set the manage texture parameters flag for a context
SYNOPSIS

#include <Cg/cgD3D11.h>

void cgD3D11SetManageTextureParameters( CGcontext context,
                                      CGbool flag );

PARAMETERS

context
The context in which the automatic texture management behavior will be changed.

flag
A boolean switch which controls automatic texture management by the runtime.

RETURN VALUES

None.

DESCRIPTION

By default, cgD3D11 does not manage any texture state in D3D. It is up to the user to enable and disable textures using D3D. This behavior is the default to avoid conflicts with texture state on geometry that’s rendered with the fixed function pipeline or without cgD3D11.

If automatic texture management is desired, cgD3D11SetManageTextureParameters may be called with flag set to CG_TRUE before cgD3D11BindProgram is called. Whenever cgD3D11BindProgram is called, the cgD3D11 runtime will make all the appropriate texture parameter calls on the application’s behalf.

Calling cgD3D11SetManageTextureParameters with flag set to CG_FALSE will disable automatic texture management.

NOTE: When cgD3D11SetManageTextureParameters is set to CG_TRUE, applications should not make texture state change calls to D3D after calling cgD3D11BindProgram, unless the application is trying to override some parts of cgD3D11’s texture management.

EXAMPLES

// Enable automatic texture management
cgD3D11SetManageTextureParameters( pCtx, CG_TRUE );

ERRORS

None.

HISTORY

cgD3D11SetManageTextureParameters was introduced in Cg 3.0.

SEE ALSO

cgD3D11GetManageTextureParameters, cgD3D11BindProgram

8.24 cgD3D11SetSamplerStateParameter

NAME

cgD3D11SetSamplerStateParameter - Sets a sampler state object to a Cg sampler parameter.
SYNOPSIS

#include <Cg/cgD3D11.h>

void cgD3D11SetSamplerStateParameter( CGparameter param,
                                       ID3D11SamplerState * samplerState );

PARAMETERS

param

The sampler parameter whose state is to be set.

samplerState

The D3D sampler state to set.

RETURN VALUES

None.

DESCRIPTION

cgD3D11SetSamplerStateParameter sets the sampler state associated with a sampler parameter.
NULL means default sampler state.

EXAMPLES

ID3D11SamplerState * sstate = NULL; Device->CreateSamplerState( &samplerDesc, &sstate );
cgD3D11SetSamplerStateParameter( myCgSamplerParam, sstate );

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if param is invalid.

HISTORY

cgD3D11SetSamplerStateParameter was introduced in Cg 3.0.

SEE ALSO

cgD3D11SetTextureSamplerStateParameter

8.25 cgD3D11SetTextureParameter

NAME

cgD3D11SetTextureParameter - sets the value of a texture parameter

SYNOPSIS

#include <Cg/cgD3D11.h>

void cgD3D11SetTextureParameter( CGparameter param,
                                 ID3D11Resource * texture );
PARAMETERS

param
The texture parameter that will be set.
texture
An D3D texture to which the parameter will be set.

RETURN VALUES
None.

DESCRIPTION
cgD3D11SetTextureParameter sets the value of a texture parameter to a given D3D11 texture object.

EXAMPLES

ID3D11Resource *myTexture;
// Assume myTexture is loaded here...
// .. and param is an effect sampler parameter
cgD3D11SetTextureParameter( param, myTexture );

ERRORS
CG_INVALID_PARAM_HANDLE_ERROR is generated if param is not a valid parameter handle.

HISTORY
cgD3D11SetTextureParameter was introduced in Cg 3.0.

SEE ALSO
cgD3D11SetManageTextureParameters

8.26 cgD3D11SetTextureSamplerStateParameter

NAME

cgD3D11SetTextureSamplerStateParameter - Sets a texture resource and sampler state object to a Cg sampler parameter.

SYNOPSIS

#include <Cg/cgD3D11.h>

void cgD3D11SetTextureSamplerStateParameter( CGparameter param,
                                          ID3D11Resource * texture,
                                          ID3D11SamplerState * samplerState );

PARAMETERS

param
The sampler parameter whose texture and state is to be set.
texture
The texture resource object being set.

samplerState

The sampler state object being set.

RETURN VALUES

None.

DESCRIPTION

cgD3D11SetTextureSamplerStateParameter accomplishes the same thing as calling both cgD3D11SetTextureParameter and cgD3D11SetSamplerStateParameter together.

EXAMPLES

ID3D11Resource * myTexture; ID3D11SamplerState * mySamplerState;
Device->CreateTexture2D( &desc, &initalData, &myTexture ); Device->CreateSamplerState( &desc, &mySamplerState );
cgD3D11SetTextureSamplerStateParameter( myCgSamplerParam, myTexture, mySamplerState );

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if param is invalid.

HISTORY

cgD3D11SetTextureSamplerStateParameter was introduced in Cg 3.0.

SEE ALSO

cgD3D11SetTextureParameter, cgD3D11SetSamplerStateParameter

8.27 cgD3D11TranslateCGerror

NAME

cgD3D11TranslateCGerror - convert a Cg runtime error into a string

SYNOPSIS

#include <Cg/cgD3D11.h>

const char * cgD3D11TranslateCGerror( CGerror error );

PARAMETERS

error

The error code to translate. Can be a core runtime error or a D3D runtime error.

RETURN VALUES

Returns a pointer to a string describing error.
DESCRIPTION

cgD3D11TranslateCGerror converts a Cg runtime error into a string. This routine should be called instead of the core runtime routine cgGetErrorString because it will also translate errors that the Cg D3D runtime generates.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCallback.

EXAMPLES

```c
char buf[512];
CGerror error = cgGetLastError();
if (error != CG_NO_ERROR)
{
    sprintf(buf, "An error occurred. Error description: '%s'\n",
            cgD3D11TranslateCGerror(error));
    OutputDebugString(buf);
}
```

ERRORS

None.

HISTORY

cgD3D11TranslateCGerror was introduced in Cg 3.0.

SEE ALSO

cgGetErrorString, cgSetErrorCallback

8.28 cgD3D11TranslateHRESULT

NAME

cgD3D11TranslateHRESULT - convert an HRESULT into a string

SYNOPSIS

```c
#include <Cg/cgD3D11.h>

const char * cgD3D11TranslateHRESULT( HRESULT hr );
```

PARAMETERS

hr

The HRESULT to translate. Can be a generic HRESULT or a D3D runtime error.

RETURN VALUES

Returns a pointer to a string describing the error.

DESCRIPTION

cgD3D11TranslateHRESULT converts an HRESULT into a string.

This routine will typically be called in debugging situations such as inside an error callback set using cgSetErrorCallback.
EXAMPLES

```c
char buf[512];
HRESULT hres = cgGetLastError();
if (FAILED(hres))
{
    sprintf(buf, "A D3D error occurred. Error description: '%s'\n",
            cgD3D11TranslateHRESULT(hres));
    OutputDebugString(buf);
}
```

ERRORS

None.

HISTORY

cgD3D11TranslateHRESULT was introduced in Cg 3.0.

SEE ALSO

cgD3D11TranslateCGerror, cgGetErrorString, cgSetErrorCallback

8.29 cgD3D11TypeToSize

NAME

cgD3D11TypeToSize - get the size of a CGtype enumerated type

SYNOPSIS

```c
#include <Cg/cgD3D11.h>

DWORD cgD3D11TypeToSize( CGtype type );
```

PARAMETERS

- `type`:
  Member of the `CGtype` enumerated type whose size is to be returned.

RETURN VALUES

- Returns the size of `type` in terms of consecutive floating point values.
- Returns 0 if the type does not have an inherent size. Sampler types fall into this category.

DESCRIPTION

- `cgD3D11TypeToSize` retrieves the size of a `CGtype` enumerated type in terms of consecutive floating point values.
- If the type does not have an inherent size, the return value is 0. Sampler types fall into this category.

EXAMPLES

```c
// param is a CGparameter initialized earlier
...
DWORD size = cgD3D11TypeToSize(cgGetParameterType(param));
// (sanity check that parameters have the expected size)
```
... 
assert(cgD3D11TypeToSize(cgGetParameterType(vsModelView)) == 16);
assert(cgD3D11TypeToSize(cgGetParameterType(psColor)) == 4);

ERRORS
None.

HISTORY
cgD3D11TypeToSize was introduced in Cg 3.0.

SEE ALSO
cgD3D11SetDevice

8.30  cgD3D11UnbindProgram

NAME
cgD3D11UnbindProgram - Unbinds a D3D11 program.

SYNOPSIS
#include <Cg/cgD3D11.h>

void cgD3D11UnbindProgram( CGprogram Program );

PARAMETERS
     Program
       A program whose profile is used to unbind the program from the API.

RETURN VALUES
None.

DESCRIPTION
cgD3D11UnbindProgram Unbinds a D3D11 program from the profile of the given program ‘Program.’

EXAMPLES
// CGprogram vertexProg;
//...
cgD3D11UnbindProgram( vertexProg );

ERRORS
CG_INVALID_PROGRAM_HANDLE_ERROR is generated if Program is not a valid program.

HISTORY
cgD3D11UnbindProgram was introduced in Cg 3.0.

SEE ALSO
cgD3D11BindProgram
8.31 cgD3D11UnloadProgram

NAME
cgD3D11UnloadProgram - Unloads a D3D shader from the runtime data structures

SYNOPSIS
#include <Cg/cgD3D11.h>

void cgD3D11UnloadProgram( CGprogram Program );

PARAMETERS

   Program

       A program whose compiled output is used to create the D3D shader.

RETURN VALUES
None.

DESCRIPTION
cgD3D11UnloadProgram Unloads a D3D shader from the runtime data structures.

EXAMPLES

// vertexProg is a CGprogram using a vertex profile
...
cgD3D11UnloadProgram( vertexProg );

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if Program is not a valid program.

HISTORY
cgD3D11UnloadProgram was introduced in Cg 3.0.

SEE ALSO

cgD3D11LoadProgram
9.1 abs

NAME
abs - returns absolute value of scalars and vectors.

SYNOPSIS

float abs(float a);
float1 abs(float1 a);
float2 abs(float2 a);
float3 abs(float3 a);
float4 abs(float4 a);

half abs(half a);
half1 abs(half1 a);
half2 abs(half2 a);
half3 abs(half3 a);
half4 abs(half4 a);

fixed abs(fixed a);
fixed1 abs(fixed1 a);
fixed2 abs(fixed2 a);
fixed3 abs(fixed3 a);
fixed4 abs(fixed4 a);

PARAMETERS

a
Vector or scalar of which to determine the absolute value.

DESCRIPTION

Returns the absolute value of a scalar or vector.

For vectors, the returned vector contains the absolute value of each element of the input vector.

REFERENCE IMPLEMENTATION

abs for a float scalar could be implemented like this.

float abs(float a)
{
}
return max(-a, a);
}

PROFILE SUPPORT

abs is supported in all profiles.
Support in the fp20 is limited.
Consider abs to be free or extremely inexpensive.

SEE ALSO

max

9.2 acos

NAME

acos - returns arccosine of scalars and vectors.

SYNOPSIS

float acos(float a);
float1 acos(float1 a);
float2 acos(float2 a);
float3 acos(float3 a);
float4 acos(float4 a);

half acos(half a);
half1 acos(half1 a);
half2 acos(half2 a);
half3 acos(half3 a);
half4 acos(half4 a);

fixed acos(fixed a);
fixed1 acos(fixed1 a);
fixed2 acos(fixed2 a);
fixed3 acos(fixed3 a);
fixed4 acos(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the arccosine.

DESCRIPTION

Returns the arccosine of a in the range [0, pi], expecting a to be in the range [-1, +1].
For vectors, the returned vector contains the arccosine of each element of the input vector.

REFERENCE IMPLEMENTATION

acos for a float scalar could be implemented like this.
float acos(float x) {
    float negate = float(x < 0);
    x = abs(x);
    float ret = -0.0187293;
    ret = ret * x;
    ret = ret + 0.0742610;
    ret = ret * x;
    ret = ret - 0.2121144;
    ret = ret * x;
    ret = ret + 1.5707288;
    ret = ret * sqrt(1.0-x);
    ret = ret - 2 * negate * ret;
    return negate * 3.14159265358979 + ret;
}

PROFILE SUPPORT

acos is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO

abs, asin, cos, sqrt

9.3 all

NAME

all - returns true if a boolean scalar or all components of a boolean vector are true.

SYNOPSIS

bool all(bool a);
bool all(bool1 a);
bool all(bool2 a);
bool all(bool3 a);
bool all(bool4 a);

PARAMETERS

a

Boolean vector or scalar of which to determine if any component is true.

DESCRIPTION

Returns true if a boolean scalar or all components of a boolean vector are true.

REFERENCE IMPLEMENTATION

all for a bool4 vector could be implemented like this.
bool all(bool4 a)
{
    return a.x && a.y && a.z && a.w;
}

PROFILE SUPPORT

all is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO

any

9.4 any

NAME

any - returns true if a boolean scalar or any component of a boolean vector is true.

SYNOPSIS

bool any(bool a);
bool any(bool1 a);
bool any(bool2 a);
bool any(bool3 a);
bool any(bool4 a);

PARAMETERS

a

    Boolean vector or scalar of which to determine if any component is true.

DESCRIPTION

Returns true if a boolean scalar or any component of a boolean vector is true.

REFERENCE IMPLEMENTATION

any for a bool4 vector could be implemented like this.

bool any(bool4 a)
{
    return a.x || a.y || a.z || a.w;
}

PROFILE SUPPORT

any is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO

all
9.5 asin

NAME

asin - returns arcsine of scalars and vectors.

SYNOPSIS

float asin(float a);
float1 asin(float1 a);
float2 asin(float2 a);
float3 asin(float3 a);
float4 asin(float4 a);

half asin(half a);
half1 asin(half1 a);
half2 asin(half2 a);
half3 asin(half3 a);
half4 asin(half4 a);

fixed asin(fixed a);
fixed1 asin(fixed1 a);
fixed2 asin(fixed2 a);
fixed3 asin(fixed3 a);
fixed4 asin(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the arcsine.

DESCRIPTION

Returns the arcsine of a in the range [-pi/2,+pi/2], expecting a to be in the range [-1,+1].

For vectors, the returned vector contains the arcsine of each element of the input vector.

REFERENCE IMPLEMENTATION

asin for a float scalar could be implemented like this.

// Handbook of Mathematical Functions
// M. Abramowitz and I.A. Stegun, Ed.

float asin(float x) {
    float negate = float(x < 0);
    x = abs(x);
    float ret = -0.0187293;
    ret *= x;
    ret += 0.0742610;
    ret *= x;
    ret -= 0.2121144;
    ret *= x;
    ret += 1.5707288;
    ret = 3.14159265358979*0.5 - sqrt(1.0 - x)*ret;
    return ret - 2 * negate * ret;
}
PROFILE SUPPORT

asin is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO

abs, acos, sin, sqrt

9.6 atan2

NAME

atan2 - returns arctangent of scalars and vectors.

SYNOPSIS

float atan2(float y, float x);
float1 atan2(float1 y, float1 x);
float2 atan2(float2 y, float2 x);
float3 atan2(float3 y, float3 x);
float4 atan2(float4 y, float4 x);

half atan2(half y, half x);
half1 atan2(half1 y, half1 x);
half2 atan2(half2 y, half2 x);
half3 atan2(half3 y, half3 x);
half4 atan2(half4 y, half4 x);

fixed atan2(fixed y, fixed x);
fixed1 atan2(fixed1 y, fixed1 x);
fixed2 atan2(fixed2 y, fixed2 x);
fixed3 atan2(fixed3 y, fixed3 x);
fixed4 atan2(fixed4 y, fixed4 x);

PARAMETERS

y
Vector or scalar for numerator of ratio of which to determine the arctangent.

x
Vector or scalar of denominator of ratio of which to determine the arctangent.

DESCRIPTION

atan2 calculates the arctangent of y/x. atan2 is well defined for every point other than the origin, even if x equals 0 and y does not equal 0.

For vectors, the returned vector contains the arctangent of each element of the input vector.

REFERENCE IMPLEMENTATION

atan2 for a float2 scalar could be implemented as an approximation like this.

float2 atan2(float2 y, float2 x)
{
    float2 t0, t1, t2, t3, t4;
t3 = abs(x);
t1 = abs(y);
t0 = max(t3, t1);
t1 = min(t3, t1);
t3 = float(1) / t0;
t3 = t1 * t3;

t4 = t3 * t3;
t0 = -float(0.013480470);
t0 = t0 * t4 + float(0.057477314);
t0 = t0 * t4 - float(0.121239071);
t0 = t0 * t4 + float(0.195635925);
t0 = t0 * t4 - float(0.332994597);
t0 = t0 * t4 + float(0.999995630);
t3 = t0 * t3;

t3 = (abs(y) > abs(x)) ? float(1.570796327) - t3 : t3;
t3 = (x < 0) ? float(3.141592654) - t3 : t3;
t3 = (y < 0) ? -t3 : t3;

} return t3;

PROFILE SUPPORT

atan2 is supported in all profiles but fp20.

SEE ALSO

abs, acos, asin, atan, sqrt, tan

9.7 atan

NAME

atan - returns arctangent of scalars and vectors.

SYNOPSIS

float atan(float a);
float1 atan(float1 a);
float2 atan(float2 a);
float3 atan(float3 a);
float4 atan(float4 a);
half atan(half a);
half1 atan(half1 a);
half2 atan(half2 a);
half3 atan(half3 a);
half4 atan(half4 a);

fixed atan(fixed a);
fixed1 atan(fixed1 a);
fixed2 atan(fixed2 a);
fixed3 atan(fixed3 a);
fixed4 atan(fixed4 a);
PARAMETERS

a

Vector or scalar of which to determine the arctangent.

DESCRIPTION

Returns the arctangent of x in the range of -\pi/2 to \pi/2 radians.
For vectors, the returned vector contains the arctangent of each element of the input vector.

REFERENCE IMPLEMENTATION

\texttt{atan} for a \texttt{float} scalar could be implemented like this.

\begin{verbatim}
float atan(float x) {
    return atan2(x, float(1));
}
\end{verbatim}

\texttt{atan2} is typically implemented as an approximation.

PROFILE SUPPORT

\texttt{atan} is supported in all profiles but \texttt{fp20}.

SEE ALSO

abs, acos, asin, atan2, sqrt, tan

9.8 bitCount

NAME

\texttt{bitCount} - return the number of bits set in a bitfield.

SYNOPSIS

\begin{verbatim}
int bitCount(int x)
int2 bitCount(int2 x)
int3 bitCount(int3 x)
int4 bitCount(int4 x)

int bitCount(uint x)
int2 bitCount(uint2 x)
int3 bitCount(uint3 x)
int4 bitCount(uint4 x)
\end{verbatim}

PARAMETERS

x

Bitfield to count bits in.

DESCRIPTION

Returns the count of set bits (value of 1) in the bitfield \texttt{x}.
REFERENCE IMPLEMENTATION

The `bitCount` function for an `int` bitfield can be implemented like this:

```c
int bitCount(int x)
{
    int i;
    int res = 0;
    for(i = 0; i < 32; i++) {
        mask = 1 << i;
        if (x & mask)
            res ++;
    }
    return res;
}
```

PROFILE SUPPORT

The `bitCount` function is supported in `gp5` profiles.

SEE ALSO

- `GL_ARB_gpu_shader5`
- `bitfieldExtract`, `bitfieldInsert`, `bitfieldReverse`, `findLSB` and `findMSB`

9.9 bitfieldExtract

NAME

`bitfieldExtract` - return an extracted range of bits from a bitfield.

SYNOPSIS

```c
int  bitfieldExtract(int  a, int  b, int  c)
int2 bitfieldExtract(int2 a, int  b, int  c)
int3 bitfieldExtract(int3 a, int  b, int  c)
int4 bitfieldExtract(int4 a, int  b, int  c)
uint bitfieldExtract(uint  a, int  b, int  c)
uint2 bitfieldExtract(uint2 a, int  b, int  c)
uint3 bitfieldExtract(uint3 a, int  b, int  c)
uint4 bitfieldExtract(uint4 a, int  b, int  c)
```

PARAMETERS

- `a` - Bitfield to extract bits from.
- `b` - Bit offset number. Bit offsets start at 0.
- `c` - Number of bits to extract.
DESCRIPTION

Returns bits from offset \( b \) of length \( c \) in the bitfield \( a \).

REFERENCE IMPLEMENTATION

\texttt{bitfieldExtract} for an \texttt{int} bitfield can be implemented like this:

```c
int bitfieldExtract(int a, int b, int c)
{
    int mask = ~(0xffffffff << c);
    if (b > 0)
        return (a >> (b - 1)) & mask;
    else
        return a & mask;
}
```

PROFILE SUPPORT

\texttt{bitfieldExtract} is supported in \texttt{gp5} profiles.

SEE ALSO

\texttt{GL_ARB_gpu_shader5}

\texttt{bitfieldInsert}, \texttt{bitfieldReverse}, \texttt{bitCount}, \texttt{findLSB} and \texttt{findMSB}

9.10 \textbf{bitfieldInsert}

NAME

\texttt{bitfieldInsert} - returns an extracted range of bits from a bitfield.

SYNOPSIS

```c
int bitfieldInsert(int a, int b, int c, int d)
int2 bitfieldInsert(int2 a, int b, int c, int d)
int3 bitfieldInsert(int3 a, int b, int c, int d)
int4 bitfieldInsert(int4 a, int b, int c, int d)

uint bitfieldInsert(uint a, uint b, int c, int d)
uint2 bitfieldInsert(uint2 a, uint b, int c, int d)
uint3 bitfieldInsert(uint3 a, uint b, int c, int d)
uint4 bitfieldInsert(uint4 a, uint b, int c, int d)
```

PARAMETERS

- \( a \)
  - Bitfield to insert bits into.
- \( b \)
  - Bit pattern to insert.
- \( c \)
  - Bit offset number.
- \( d \)
Number of bits to insert.

DESCRIPTION
Returns the result of inserting bits \( B \) at offset \( C \) of length \( D \) in the bitfield \( A \).

REFERENCE IMPLEMENTATION

\texttt{bitfieldInsert} for an \texttt{int} bitfield can be implemented like this:

\begin{verbatim}
int bitfieldInsert(int a, int b, int c, int d)
{
    uint mask = ~(0xffffffff << d) << c;
    mask ^= mask;
    a &= mask;
    return a | (b << c);
}
\end{verbatim}

PROFILE SUPPORT

\texttt{bitfieldInsert} is supported in \texttt{gp5} profiles.

SEE ALSO

\texttt{GL_ARB_gpu_shader5}

\texttt{bitfieldExtract, bitfieldReverse, bitCount, findLSB} and \texttt{findMSB}

### 9.11 bitfieldReverse

NAME

\texttt{bitfieldReverse} - return the reversed bitfield.

SYNOPSIS

\begin{verbatim}
int  bitfieldReverse(int  x)
int2 bitfieldReverse(int2 x)
int3 bitfieldReverse(int3 x)
int4 bitfieldReverse(int4 x)

uint bitfieldReverse(uint  x)
uint2 bitfieldReverse(uint2 x)
uint3 bitfieldReverse(uint3 x)
uint4 bitfieldReverse(uint4 x)
\end{verbatim}

PARAMETERS

\texttt{x}

Bitfield to reverse.

DESCRIPTION
Returns the reverse of the bitfield \( x \).
REFERENCE IMPLEMENTATION

`bitfieldReverse` for an `int` bitfield can be implemented like this:

```c
int bitfieldReverse(int x) {
    int res = 0;
    int i, shift, mask;

    for (i = 0; i < 32; i++) {
        mask = 1 << i;
        shift = 32 - 2*i - 1;
        mask &= x;
        mask = (shift > 0) ? mask << shift : mask >> -shift;
        res |= mask;
    }

    return res;
}
```

PROFILE SUPPORT

`bitfieldReverse` is supported in `gp5` and `hls11` profiles.

SEE ALSO

`GL_ARB_gpu_shader5`

`bitfieldInsert`, `bitfieldExtract`, `bitCount`, `findLSB` and `findMSB`

9.12 ceil

NAME

`ceil` - returns smallest integer not less than a scalar or each vector component.

SYNOPSIS

```c
float  ceil(float  a);
float1 ceil(float1 a);
float2 ceil(float2 a);
float3 ceil(float3 a);
float4 ceil(float4 a);

half   ceil(half  a);
half1  ceil(half1 a);
half2  ceil(half2 a);
half3  ceil(half3 a);
half4  ceil(half4 a);

fixed  ceil(fixed  a);
fixed1 ceil(fixed1 a);
fixed2 ceil(fixed2 a);
fixed3 ceil(fixed3 a);
fixed4 ceil(fixed4 a);
```
PARAMETERS

a

Vector or scalar of which to determine the ceiling.

DESCRIPTION

Returns the ceiling or smallest integer not less than a scalar or each vector component.

REFERENCE IMPLEMENTATION

ceil for a float scalar could be implemented like this.

```c
float ceil(float v)
{
    return -floor(-v);
}
```

PROFILE SUPPORT

ceil is supported in all profiles except fp20.

SEE ALSO

floor

9.13 clamp

NAME

clamp - returns smallest integer not less than a scalar or each vector component.

SYNOPSIS

```c
float clamp(float x, float a, float b);
float1 clamp(float1 x, float1 a, float1 b);
float2 clamp(float2 x, float2 a, float2 b);
float3 clamp(float3 x, float3 a, float3 b);
float4 clamp(float4 x, float4 a, float4 b);

half clamp(half x, half a, half b);
half1 clamp(half1 x, half1 a, half1 b);
half2 clamp(half2 x, half2 a, half2 b);
half3 clamp(half3 x, half3 a, half3 b);
half4 clamp(half4 x, half4 a, half4 b);

fixed clamp(fixed x, fixed a, fixed b);
fixed1 clamp(fixed1 x, fixed1 a, fixed1 b);
fixed2 clamp(fixed2 x, fixed2 a, fixed2 b);
fixed3 clamp(fixed3 x, fixed3 a, fixed3 b);
fixed4 clamp(fixed4 x, fixed4 a, fixed4 b);

float1 clamp(float1 x, float a, float b);
float2 clamp(float2 x, float a, float b);
float3 clamp(float3 x, float a, float b);
float4 clamp(float4 x, float a, float b);

half1 clamp(half1 x, half a, half b);
```

9.13. clamp
half2 clamp(half2 x, half a, half b);
half3 clamp(half3 x, half a, half b);
half4 clamp(half4 x, half a, half b);

fixed1 clamp(fixed1 x, fixed a, fixed b);
fixed2 clamp(fixed2 x, fixed a, fixed b);
fixed3 clamp(fixed3 x, fixed a, fixed b);
fixed4 clamp(fixed4 x, fixed a, fixed b);

PARAMETERS

x
  Vector or scalar to clamp.
a
  Vector or scalar for bottom of clamp range.
b
  Vector or scalar for top of clamp range.

DESCRIPTION

Returns \( x \) clamped to the range \([a, b]\) as follows:

1. Returns \( a \) if \( x \) is less than \( a \); else
2. Returns \( b \) if \( x \) is greater than \( b \); else
3. Returns \( x \) otherwise.

For vectors, the returned vector contains the clamped result of each element of the vector \( x \) clamped using the respective element of vectors \( a \) and \( b \).

REFERENCE IMPLEMENTATION

clamp for float scalars could be implemented like this.

```c
float clamp(float x, float a, float b)
{
    return max(a, min(b, x));
}
```

PROFILE SUPPORT

clamp is supported in all profiles except fp20.

SEE ALSO

max, min, saturate

9.14 clip

NAME

clip - conditionally kill a pixel before output
SYNOPSIS

```c
void clip(float4 x);
void clip(float3 x);
void clip(float2 x);
void clip(float1 x);
void clip(float x);
```

PARAMETERS

`x`

Vector/scalar condition to clip on

DESCRIPTION

kills the current pixel output if any component of the given vector, or the given scalar, is negative

REFERENCE IMPLEMENTATION

clip is equivalent to

```c
void clip(float4 x)
{
    if (any(x < 0))
        discard;
}
```

PROFILE SUPPORT

clip is supported in all pixel/fragment profiles.

SEE ALSO

9.15 cosh

NAME

cosh - returns hyperbolic cosine of scalars and vectors.

SYNOPSIS

```c
float cosh(float a);
float1 cosh(float1 a);
float2 cosh(float2 a);
float3 cosh(float3 a);
float4 cosh(float4 a);

half cosh(half a);
half1 cosh(half1 a);
half2 cosh(half2 a);
half3 cosh(half3 a);
half4 cosh(half4 a);

fixed cosh(fixed a);
fixed1 cosh(fixed1 a);
fixed2 cosh(fixed2 a);
fixed3 cosh(fixed3 a);
fixed4 cosh(fixed4 a);
```
PARAMETERS

a

Vector or scalar of which to determine the hyperbolic cosine.

DESCRIPTION

Returns the hyperbolic cosine of a.

For vectors, the returned vector contains the hyperbolic cosine of each element of the input vector.

REFERENCE IMPLEMENTATION

`cosh` for a scalar `float` could be implemented like this.

```c
float cosh(float x)
{
    return 0.5 * (exp(x) + exp(-x));
}
```

PROFILE SUPPORT

`cosh` is supported in all profiles except `fp20`.

SEE ALSO

`acos`, `cos`, `exp`, `sinh`, `tanh`

9.16 cos

NAME

`cos` - returns cosine of scalars and vectors.

SYNOPSIS

```c
float cos(float a);
float1 cos(float1 a);
float2 cos(float2 a);
float3 cos(float3 a);
float4 cos(float4 a);

half cos(half a);
half1 cos(half1 a);
half2 cos(half2 a);
half3 cos(half3 a);
half4 cos(half4 a);

fixed cos(fixed a);
fixed1 cos(fixed1 a);
fixed2 cos(fixed2 a);
fixed3 cos(fixed3 a);
fixed4 cos(fixed4 a);
```

PARAMETERS

a

Vector or scalar of which to determine the cosine.
DESCRIPTION

Returns the cosine of \( a \) in radians. The return value is in the range \([-1,+1]\).

For vectors, the returned vector contains the cosine of each element of the input vector.

REFERENCE IMPLEMENTATION

\( \cos \) is best implemented as a native cosine instruction, however \( \cos \) for a \texttt{float} scalar could be implemented by an approximation like this.

\[
\begin{align*}
\cos(\texttt{float} \ a) & \\
& \{
/* \text{C simulation gives a max absolute error of less than } 1.8e-7 */
\textbf{const} \texttt{float4} c0 = \texttt{float4} (\ 0.0, \ 0.5, \\
& \quad \ 1.0, \ 0.0 \ );
\textbf{const} \texttt{float4} c1 = \texttt{float4} (\ 0.25, \ -9.0, \\
& \quad \ 0.75, \ 0.159154943091 \ );
\textbf{const} \texttt{float4} c2 = \texttt{float4} (\ 24.9808039603, \ -24.9808039603, \\
& \quad \ -60.1458091736, \ 60.1458091736 \ );
\textbf{const} \texttt{float4} c3 = \texttt{float4} (\ 85.4537887573, \ -85.4537887573, \\
& \quad \ -64.9393539429, \ 64.9393539429 \ );
\textbf{const} \texttt{float4} c4 = \texttt{float4} (\ 19.7392082214, \ -19.7392082214, \\
& \quad \ -1.0, \ 1.0 \ );
/* r0.x = \cos(a) */
\texttt{float3} r0, r1, r2;
\texttt{r1.x} = \texttt{c1.w} * \ a; \quad & // \text{normalize input}
\texttt{r1.y} = \texttt{frac(} \ r1.x \ \texttt{)}; \quad & // \text{and extract fraction}
\texttt{r2.x} = \texttt{(float)} (\ \texttt{r1.y} < \texttt{c1.x} \ ); \quad & // \text{range check: 0.0 to 0.25}
\texttt{r2.yz} = \texttt{(float2)} (\ \texttt{r1.yy} >= \texttt{c1.yz} \ ); \quad & // \text{range check: 0.75 to 1.0}
\texttt{r2.y} = \texttt{dot(} \ r2, \texttt{c4.zwz} \ ); \quad & // \text{range check: 0.25 to 0.75}
\texttt{r0} = \texttt{c0.xyz} - \texttt{r1.yyy}; \quad & // \text{range centering}
\texttt{r0} = \texttt{r0} * \texttt{r0}; \quad &
\texttt{r1} = \texttt{c2.xyx} * \texttt{r0} + \texttt{c2.zwz}; \quad & // \text{start power series}
\texttt{r1} = \texttt{r1} * \texttt{r0} + \texttt{c3.xyx}; \quad &
\texttt{r1} = \texttt{r1} * \texttt{r0} + \texttt{c3.zwz}; \quad &
\texttt{r1} = \texttt{r1} * \texttt{r0} + \texttt{c4.xyx}; \quad &
\texttt{r1} = \texttt{r1} * \texttt{r0} + \texttt{c4.zwz}; \quad &
\texttt{r0.x} = \texttt{dot(} \ r1, \texttt{-r2} \ ); \quad & // \text{range extract}
\texttt{return} \texttt{r0.x};
\end{align*}
\]

PROFILE SUPPORT

\( \cos \) is fully supported in all profiles unless otherwise specified.

\( \cos \) is supported via an approximation (shown above) in the \texttt{vs\_1\_1}, \texttt{vp20}, and \texttt{arbvp1} profiles.

\( \cos \) is unsupported in the \texttt{fp20}, \texttt{ps\_1\_1}, \texttt{ps\_1\_2}, and \texttt{ps\_1\_3} profiles.

SEE ALSO

\( \texttt{acos}, \texttt{dot}, \texttt{frac}, \texttt{sin}, \texttt{sincos}, \texttt{tan} \)
9.17 cross

NAME
cross - returns the cross product of two three-component vectors

SYNOPSIS
float3 cross(float3 a, float3 b);
half3 cross(half3 a, half3 b);
fixed3 cross(fixed3 a, fixed3 b);

PARAMETERS
a
Three-component vector.
b
Three-component vector.

DESCRIPTION
Returns the cross product of three-component vectors a and b. The result is a three-component vector.

REFERENCE IMPLEMENTATION
cross for float3 vectors could be implemented this way:
float3 cross(float3 a, float3 b)
{
    return a.yzx * b.zxy - a.zxy * b.yzx;
}

PROFILE SUPPORT
cross is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO
dot

9.18 ddx

NAME
ddx - returns approximate partial derivative with respect to window-space X

SYNOPSIS
float ddx(float a);
float1 ddx(float1 a);
float2 ddx(float2 a);
float3 ddx(float3 a);
float4 ddx(float4 a);
half ddx(half a);
half1 ddx(half1 a);
half2 ddx(half2 a);
half3 ddx(half3 a);
half4 ddx(half4 a);
fixed ddx(fixed a);
fixed1 ddx(fixed1 a);
fixed2 ddx(fixed2 a);
fixed3 ddx(fixed3 a);
fixed4 ddx(fixed4 a);

PARAMETERS
a
Vector or scalar of which to approximate the partial derivative with respect to window-space X.

DESCRIPTION
Returns approximate partial derivative of a with respect to the window-space (horizontal) x coordinate.
For vectors, the returned vector contains the approximate partial derivative of each element of the input vector.
This function is only available in fragment program profiles (but not all of them).
The specific way the partial derivative is computed is implementation-dependent. Typically fragments are rasterized in 2x2 arrangements of fragments (called quad-fragments) and the partial derivatives of a variable is computed by differencing with the adjacent horizontal fragment in the quad-fragment.
The partial derivative computation may incorrect when ddx is used in control flow paths where not all the fragments within a quad-fragment have branched the same way.
The partial derivative computation may be less exact (wobbly) when the variable is computed based on varying parameters interpolated with centroid interpolation.

REFERENCE IMPLEMENTATION
ddx is not expressible in Cg code.

PROFILE SUPPORT
ddx is supported only in fragment profiles. Vertex and geometry profiles lack the concept of window space.
ddx is unsupported in the fp20, ps_1_1, ps_1_2, ps_1_3, and arbfp1 profiles.

SEE ALSO
ddy, fp30, fp40, fwidth, gp4fp

9.19 ddy

NAME
ddy - returns approximate partial derivative with respect to window-space Y
SYNOPSIS

float  ddy(float  a);
float1 ddy(float1 a);
float2 ddy(float2 a);
float3 ddy(float3 a);
float4 ddy(float4 a);

half  ddy(half  a);
half1 ddy(half1 a);
half2 ddy(half2 a);
half3 ddy(half3 a);
half4 ddy(half4 a);

fixed ddy(fixed  a);
fixed1 ddy(fixed1 a);
fixed2 ddy(fixed2 a);
fixed3 ddy(fixed3 a);
fixed4 ddy(fixed4 a);

PARAMETERS

a

Vector or scalar of which to approximate the partial derivative with respect to window-space Y.

DESCRIPTION

Returns approximate partial derivative of a with respect to the window-space (vertical) y coordinate.

For vectors, the returned vector contains the approximate partial derivative of each element of the input vector.

This function is only available in fragment program profiles (but not all of them).

The specific way the partial derivative is computed is implementation-dependent. Typically fragments are rasterized in 2x2 arrangements of fragments (called quad-fragments) and the partial derivatives of a variable is computed by differencing with the adjacent horizontal fragment in the quad-fragment.

The partial derivative computation may incorrect when ddy is used in control flow paths where not all the fragments within a quad-fragment have branched the same way.

The partial derivative computation may be less exact (wobbly) when the variable is computed based on varying parameters interpolated with centroid interpolation.

REFERENCE IMPLEMENTATION

ddy is not expressible in Cg code.

PROFILE SUPPORT

ddy is supported only in fragment profiles. Vertex and geometry profiles lack the concept of window space.

ddy is unsupported in the fp20, ps_1_1, ps_1_2, ps_1_3, and arbfp1 profiles.

SEE ALSO

ddx, fp30, fp40, fwidth, gp4fp
9.20 degrees

NAME

degrees - converts values of scalars and vectors from radians to degrees

SYNOPSIS

float degrees(float a);
float1 degrees(float1 a);
float2 degrees(float2 a);
float3 degrees(float3 a);
float4 degrees(float4 a);

half degrees(half a);
half1 degrees(half1 a);
half2 degrees(half2 a);
half3 degrees(half3 a);
half4 degrees(half4 a);

fixed degrees(fixed a);
fixed1 degrees(fixed1 a);
fixed2 degrees(fixed2 a);
fixed3 degrees(fixed3 a);
fixed4 degrees(fixed4 a);

PARAMETERS

a

Vector or scalar of which to convert from radians to degrees.

DESCRIPTION

Returns the scalar or vector converted from radians to degrees.

For vectors, the returned vector contains each element of the input vector converted from radians to degrees.

REFERENCE IMPLEMENTATION

degrees for a float scalar could be implemented like this.

float degrees(float a)
{
    return 57.29577951 * a;
}

PROFILE SUPPORT

degrees is supported in all profiles except fp20.

SEE ALSO

cos, radians, sin, tan
9.21 determinant

NAME

determinant - return the scalar determinant of a square matrix

SYNOPSIS

float determinant(float1x1 A);
float determinant(float2x2 A);
float determinant(float3x3 A);
float determinant(float4x4 A);

PARAMETERS

A

Square matrix of which to compute the determinant.

DESCRIPTION

Returns the determinant of the square matrix \( A \).

REFERENCE IMPLEMENTATION

The various \texttt{determinant} functions can be implemented like this:

\begin{verbatim}
float determinant(float1x1 A)
{
    return A._m00;
}

float determinant(float2x2 A)
{
    return A._m00*A._m11 - A._m01*A._m10;
}

float determinant(float3x3 A)
{
    return dot(A._m00_m01_m02,
               A._m11_m12_m10 * A._m22_m23_m20 - A._m12_m10_m11 * A._m21_m22_m20);
}

float determinant(float4x4 A) 
{
    return dot(float4(1,-1,1,-1) * A._m00_m01_m02_m03,
               A._m11_m12_m13_m10*( A._m22_m23_m20_m21 + A._m33_m30_m31_m32
                                 - A._m23_m20_m21_m22 + A._m32_m30_m31_m32)
               + A._m12_m13_m10_m11*( A._m23_m20_m21_m22 + A._m31_m32_m33_m30
                                  - A._m21_m22_m23_m20 + A._m33_m30_m31_m32)
               + A._m13_m10_m11_m12*( A._m21_m22_m23_m20 + A._m32_m33_m30_m31
                                 - A._m22_m23_m20_m21 + A._m31_m32_m33_m30));
}
\end{verbatim}

PROFILE SUPPORT

determinant is supported in all profiles except \texttt{fp20}, \texttt{vs_1_1}, \texttt{ps_1_1}, \texttt{ps_1_2} and \texttt{ps_1_3}. However profiles such as \texttt{ps_2_0} without native floating-point will have problems computing the larger determinants and may have ranges issues computing even small determinants.
SEE ALSO
mul, transpose

9.22 distance

NAME
distance - return the Euclidean distance between two points

SYNOPSIS

float distance(float pt1, float pt2);
float distance(float1 pt1, float1 pt2);
float distance(float2 pt1, float2 pt2);
float distance(float3 pt1, float3 pt2);
float distance(float4 pt1, float4 pt2);
half distance(half pt1, half pt2);
half distance(half1 pt1, half1 pt2);
half distance(half2 pt1, half2 pt2);
half distance(half3 pt1, half3 pt2);
half distance(half4 pt1, half4 pt2);

fixed distance(fixed pt1, fixed pt2);
fixed distance(fixed1 pt1, fixed1 pt2);
fixed distance(fixed2 pt1, fixed2 pt2);
fixed distance(fixed3 pt1, fixed3 pt2);
fixed distance(fixed4 pt1, fixed4 pt2);

PARAMETERS

pt1
First point.

pt2
Second point.

DESCRIPTION

Returns the Euclidean distance from a first point pt1 to a second point pt2.

REFERENCE IMPLEMENTATION
distance for a float3 vector could be implemented like this.

float distance(float3 pt1, float3 pt2)
{
  float3 v = vt2 - pt1;
  return sqrt(dot(v,v));
}

PROFILE SUPPORT
distance is supported in all profiles except fp20.
SEE ALSO

dot, length, normalize, sqrt

### 9.23 dot

**NAME**

dot - returns the scalar dot product of two vectors

**SYNOPSIS**

```c
float dot(float a, float b);
float dot(float1 a, float1 b);
float dot(float2 a, float2 b);
float dot(float3 a, float3 b);
float dot(float4 a, float4 b);

half dot(half a, half b);
half dot(half1 a, half1 b);
half dot(half2 a, half2 b);
half dot(half3 a, half3 b);
half dot(half4 a, half4 b);

fixed dot(fixed a, fixed b);
fixed dot(fixed1 a, fixed1 b);
fixed dot(fixed2 a, fixed2 b);
fixed dot(fixed3 a, fixed3 b);
fixed dot(fixed4 a, fixed4 b);
```

**PARAMETERS**

- `a`
  
  First vector.

- `b`
  
  Second vector.

**DESCRIPTION**

Returns the scalar dot product of two same-typed vectors `a` and `b`.

**REFERENCE IMPLEMENTATION**

dot for *float4* vectors could be implemented this way:

```c
float dot(float4 a, float4 b)
{
    return a.x*b.x + a.y*b.y + a.z*b.z + a.w*b.w;
}
```

**PROFILE SUPPORT**

dot is supported in all profiles.

The *fixed3* dot product is very efficient in the *fp20* and *fp30* profiles.

The *float3* and *float4* dot products are very efficient in the *vp20*, *vp30*, *vp40*, *arbvp1*, *fp30*, *fp40*, and *arbfp1* profiles.
The float2 dot product is very efficient in the fp40 profile. In optimal circumstances, two two-component dot products can sometimes be performed at the four-component and three-component dot product rate.

SEE ALSO
cross, lit, mul

9.24 exp2

NAME

exp2 - returns the base-2 exponential of scalars and vectors

SYNOPSIS

float exp2(float a);
float1 exp2(float1 a);
float2 exp2(float2 a);
float3 exp2(float3 a);
float4 exp2(float4 a);

half exp2(half a);
half1 exp2(half1 a);
half2 exp2(half2 a);
half3 exp2(half3 a);
half4 exp2(half4 a);

fixed exp2(fixed a);
fixed1 exp2(fixed1 a);
fixed2 exp2(fixed2 a);
fixed3 exp2(fixed3 a);
fixed4 exp2(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the base-2 exponential.

DESCRIPTION

Returns the base-2 exponential \( a \).

For vectors, the returned vector contains the base-2 exponential of each element of the input vector.

REFERENCE IMPLEMENTATION

float3 exp2(float3 a)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++) {
        rv[i] = exp2(a[i]); // this is the ANSI C standard library exp2()
    }
    return rv;
}

exp2 is typically implemented with a native base-2 exponential instruction.
PROFILE SUPPORT

exp2 is fully supported in all profiles unless otherwise specified.
Support in the fp20 is limited to constant compile-time evaluation.

SEE ALSO

exp, log, pow

9.25 exp

NAME

exp - returns the base-e exponential of scalars and vectors

SYNOPSIS

float exp(float a);
float1 exp(float1 a);
float2 exp(float2 a);
float3 exp(float3 a);
float4 exp(float4 a);

half exp(half a);
half1 exp(half1 a);
half2 exp(half2 a);
half3 exp(half3 a);
half4 exp(half4 a);

fixed exp(fixed a);
fixed1 exp(fixed1 a);
fixed2 exp(fixed2 a);
fixed3 exp(fixed3 a);
fixed4 exp(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the base-e exponential. The value e is approximately 2.71828182845904523536.

DESCRIPTION

Returns the base-e exponential a.

For vectors, the returned vector contains the base-e exponential of each element of the input vector.

REFERENCE IMPLEMENTATION

float3 exp(float3 a)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++)
        rv[i] = exp(a[i]); // this is the ANSI C standard library exp()
exp is typically implemented with either a native base-2 exponentional instruction or pow.

PROFILE SUPPORT

exp is fully supported in all profiles unless otherwise specified.
Support in the fp20 is limited to constant compile-time evaluation.

SEE ALSO

exp2, log, pow

9.26 faceforward

NAME

faceforward - returns a normal as-is if a vertex’s eye-space position vector points in the opposite direction of a geometric normal, otherwise return the negated version of the normal

SYNOPSIS

float faceforward(float N, float I, float Ng);
float1 faceforward(float1 N, float1 I, float1 Ng);
float2 faceforward(float2 N, float2 I, float2 Ng);
float3 faceforward(float3 N, float3 I, float3 Ng);
float4 faceforward(float4 N, float4 I, float4 Ng);

half faceforward(half N, half I, half Ng);
half1 faceforward(half1 N, half1 I, half1 Ng);
half2 faceforward(half2 N, half2 I, half2 Ng);
half3 faceforward(half3 N, half3 I, half3 Ng);
half4 faceforward(half4 N, half4 I, half4 Ng);

fixed faceforward(fixed N, fixed I, fixed Ng);
fixed1 faceforward(fixed1 N, fixed1 I, fixed1 Ng);
fixed2 faceforward(fixed2 N, fixed2 I, fixed2 Ng);
fixed3 faceforward(fixed3 N, fixed3 I, fixed3 Ng);
fixed4 faceforward(fixed4 N, fixed4 I, fixed4 Ng);

PARAMETERS

N
    Perturbed normal vector.
I
    Incidence vector (typically a direction vector from the eye to a vertex).
Ng
    Geometric normal vector (for some facet the perturbed normal belongs).
DESCRIPTION

Returns a (peturbed) normal as-is if a vertex’s eye-space position vector points in the opposite direction of a geometric normal, otherwise return the negated version of the (peturbed) normal

Mathematically, if the dot product of $I$ and $Ng$ is negative, $N$ is returned unchanged; otherwise $-N$ is returned.

This function is inspired by a RenderMan function of the same name though the RenderMan version has only two parameters.

REFERENCE IMPLEMENTATION

`faceforward` for `float3` vectors could be implemented this way:

```cpp
float3 faceforward( float3 N, float3 I, float Ng )
{
    return dot(I, Ng) < 0 ? N : -N;
}
```

PROFILE SUPPORT

`refract` is supported in all profiles.

SEE ALSO

`dot`, `reflect`, `refract`, `normalize`

9.27 findLSB

NAME

`findLSB` - return the number of the least significant set bit.

SYNOPSIS

```cpp
int   findLSB(int  x)
int2  findLSB(int2 x)
int3  findLSB(int3 x)
int4  findLSB(int4 x)

int   findLSB(uint x)
int2  findLSB(uint2 x)
int3  findLSB(uint3 x)
int4  findLSB(uint4 x)
```

PARAMETERS

$x$

Bitfield to find LSB in.

DESCRIPTION

Returns the bit number of the least significant bit (value of 1) in the bitfield $x$. If no bits have the value 1 then -1 is returned.
REFERENCE IMPLEMENTATION

**findLSB** for an **int** bitfield can be implemented like this:

```c
int findLSB(int x)
{
    int i;
    int mask;
    int res = -1;
    for(i = 0; i < 32; i++) {
        mask = 1 << i;
        if (x & mask) {
            res = i;
            break;
        }
    }
    return res;
}
```

PROFILE SUPPORT

**findLSB** is supported in **gp5** and **hlsl11** profiles.

SEE ALSO

GL_ARB_gpu_shader5

*bitfieldExtract, bitfieldInsert, bitfieldReverse, bitCount* and **findMSB**

### 9.28 findMSB

**NAME**

**findMSB** - return the number of the most significant bit.

**SYNOPSIS**

```c
int findMSB(int x)
int2 findMSB(int2 x)
int3 findMSB(int3 x)
int4 findMSB(int4 x)

int findMSB(uint x)
int2 findMSB(uint2 x)
int3 findMSB(uint3 x)
int4 findMSB(uint4 x)
```

**PARAMETERS**

- **x**

  Bitfield to find MSB in.

**DESCRIPTION**

Returns the bit number of the most significant bit (value of 1) in the bitfield **x**. If the number is negative then the position of the first zero bit is returned. If no bits have the value 1 (or 0 in the negative case) then -1 is returned.
REFERENCE IMPLEMENTATION

findMSB for an int bitfield can be implemented like this:

```c
int findMSB(int x)
{
    int i;
    int mask;
    int res = -1;
    if (x < 0) x ^= x;
    for (i = 0; i < 32; i++) {
        mask = 0x80000000 >> i;
        if (x & mask) {
            res = 31 - i;
            break;
        }
    }
    return res;
}
```

PROFILE SUPPORT

findMSB is supported in gp5 and hls11 profiles.

SEE ALSO

GL_ARB_gpu_shader5

bitfieldExtract, bitfieldInsert, bitfieldReverse, bitCount and findLSB

9.29 floatToIntBits

NAME

floatToIntBits - returns the 32-bit integer representation of an IEEE 754 floating-point scalar or vector

SYNOPSIS

```c
int floatToIntBits(float x);
int1 floatToIntBits(float1 x);
int2 floatToIntBits(float2 x);
int3 floatToIntBits(float3 x);
int4 floatToIntBits(float4 x);
```

PARAMETERS

x

Floating-point vector or scalar to cast to a scalar int or vector of ints.

DESCRIPTION

Returns a representation of the specified floating-point scalar value or vector values according to the IEEE 754 floating-point “single format” bit layout.

Not-A-Number (NaN) floating-point values are canonicalized to the integer value 0x7fc00000 regardless of the specific NaN encoding. The sign bit of the NaN is discarded.

This function is based on Java’s java.lang.Float method of the same name. See:
**floatToIntBits** requires instructions to be generated to canonicalize NaN values so **floatToIntBits** is typically more expensive than **floatToRawIntBits**.

**REFERENCE IMPLEMENTATION**

**floatToIntBits** operates consistent with the following ANSI C code:

```c
int floatToIntBits(float x)
{
    union {
        float f;  // assuming 32-bit IEEE 754 single-precision
        int i;   // assuming 32-bit 2’s complement int
    } u;

    if (isnan(x)) {
        return 0x7fc00000;
    } else {
        u.f = x;
        return u.i;
    }
}
```

**PROFILE SUPPORT**

**floatToIntBits** is supported by the gp4vp, gp4gp, and gp4vp profiles.

**floatToIntBits** is not supported by pre-G80 profiles.

**SEE ALSO**

ceil, floatToRawIntBits, floor, intBitsToFloat, round, trunc

### 9.30 floatToRawIntBits

**NAME**

**floatToRawIntBits** - returns the raw 32-bit integer representation of an IEEE 754 floating-point scalar or vector

**SYNOPSIS**

```c
int floatToRawIntBits(float x);
int1 floatToRawIntBits(float1 x);
int2 floatToRawIntBits(float2 x);
int3 floatToRawIntBits(float3 x);
int4 floatToRawIntBits(float4 x);
```

**PARAMETERS**

- **x**
  
  Floating-point vector or scalar to raw cast to a scalar int or vector of ints.

**DESCRIPTION**

Returns a representation of the specified floating-point scalar value or vector values according to the IEEE 754 floating-point “single format” bit layout, preserving Not-a-Number (NaN) values.

This function is based on Java’s java.lang.Float method of the same name. See:
The Cg compiler can typically optimize `floatToRawIntBits` so it has no instruction cost.

**REFERENCE IMPLEMENTATION**

`floatToRawIntBits` operates consistent with the following ANSI C code:

```c
int floatToRawIntBits(float x) {
    union {
        float f; // assuming 32-bit IEEE 754 single-precision
        int i;  // assuming 32-bit 2’s complement int
    } u;
    u.f = x;
    return u.i;
}
```

**PROFILE SUPPORT**

`floatToRawIntBits` is supported by the `gp4vp`, `gp4gp`, and `gp4vp` profiles.

`floatToRawIntBits` is *not* supported by pre-G80 profiles.

**SEE ALSO**

`ceil`, `floatToIntBits`, `floor`, `intBitsToFloat`, `round`, `trunc`

### 9.31 floor

**NAME**

`floor` - returns largest integer not greater than a scalar or each vector component.

**SYNOPSIS**

```c
float  floor(float  a);
float1 floor(float1 a);
float2 floor(float2 a);
float3 floor(float3 a);
float4 floor(float4 a);
```

```c
half  floor(half  a);
half1 floor(half1 a);
half2 floor(half2 a);
half3 floor(half3 a);
half4 floor(half4 a);
```

```c
fixed floor(fixed  a);
fixed1 floor(fixed1 a);
fixed2 floor(fixed2 a);
fixed3 floor(fixed3 a);
fixed4 floor(fixed4 a);
```

**PARAMETERS**

- `a`
Vector or scalar of which to determine the floor.

**DESCRIPTION**

Returns the floor or largest integer not greater than a scalar or each vector component.

**REFERENCE IMPLEMENTATION**

floor for a float3 vector could be implemented like this.

```c
float3 floor(float3 v)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++) {
        rv[i] = v[i] - frac(v[i]);
    }
    return rv;
}
```

**PROFILE SUPPORT**

floor is supported in all profiles except fp20.

**SEE ALSO**

ceil, round

### 9.32 fmod

**NAME**

fmod - returns the remainder of \( x/y \) with the same sign as \( x \)

**SYNOPSIS**

```c
float fmod(float x, float y);
float1 fmod(float1 x, float1 y);
float2 fmod(float2 x, float2 y);
float3 fmod(float3 x, float3 y);
float4 fmod(float4 x, float4 y);

half  fmod(half x, half y);
half1 fmod(half1 x, half1 y);
half2 fmod(half2 x, half2 y);
half3 fmod(half3 x, half3 y);
half4 fmod(half4 x, half4 y);

fixed fmod(fixed x, fixed y);
fixed1 fmod(fixed1 x, fixed1 y);
fixed2 fmod(fixed2 x, fixed2 y);
fixed3 fmod(fixed3 x, fixed3 y);
fixed4 fmod(fixed4 x, fixed4 y);
```

**PARAMETERS**

\( x \)
Vector or scalar numerator

\( y \)

Vector or scalar denominator

**DESCRIPTION**

`fmod` returns the remainder of \( x \) divided by \( y \) with the same sign as \( x \). If \( y \) is zero, the result is implementation-defined because of division by zero.

For vectors, the returned vector contains the signed remainder of each element of the input vector.

**REFERENCE IMPLEMENTATION**

`fmod` for an `float2` vector could be implemented as:

```cpp
float2 fmod(float2 a, float2 b)
{
    float2 c = frac(abs(a/b)) * abs(b);
    return (a < 0) ? -c : c; /* if ( a < 0 ) c = 0-c */
}
```

**PROFILE SUPPORT**

`fmod` is supported in all profiles but `fp20`.

**SEE ALSO**

`abs, frac`

### 9.33 frac

**NAME**

`frac` - returns the fractional portion of a scalar or each vector component.

**SYNOPSIS**

```cpp
float frac(float a);
float1 frac(float1 a);
float2 frac(float2 a);
float3 frac(float3 a);
float4 frac(float4 a);

half frac(half a);
half1 frac(half1 a);
half2 frac(half2 a);
half3 frac(half3 a);
half4 frac(half4 a);

fixed frac(fixed a);
fixed1 frac(fixed1 a);
fixed2 frac(fixed2 a);
fixed3 frac(fixed3 a);
fixed4 frac(fixed4 a);
```
PARAMETERS

\[ a \]

Vector or scalar of which to return its fractional portion.

DESCRIPTION

Returns the fractional portion of a scalar or each vector component.

REFERENCE IMPLEMENTATION

frac for a float scalar could be implemented like this.

```cpp
float frac(float v)
{
    return v - floor(v);
}
```

PROFILE SUPPORT

frac is supported in all profiles except fp20.

SEE ALSO

ceil, floor, round, trunc

9.34 frexp

NAME

frexp - splits scalars and vectors into normalized fraction and a power of 2

SYNOPSIS

```cpp
float frexp(float x, out float e);
float1 frexp(float1 x, out float1 e);
float2 frexp(float2 x, out float2 e);
float3 frexp(float3 x, out float3 e);
float4 frexp(float4 x, out float4 e);

half frexp(half x, out half e);
half1 frexp(half1 x, out half1 e);
half2 frexp(half2 x, out half2 e);
half3 frexp(half3 x, out half3 e);
half4 frexp(half4 x, out half4 e);

fixed frexp(fixed x, out fixed e);
fixed1 frexp(fixed1 x, out fixed1 e);
fixed2 frexp(fixed2 x, out fixed2 e);
fixed3 frexp(fixed3 x, out fixed3 e);
fixed4 frexp(fixed4 x, out fixed4 e);
```

PARAMETERS

\[ x \]

Vector or scalar of which to split.

\[ e \]
Vector or scalar where the exponent of \( x \) is output.

**DESCRIPTION**

This function decomposes \( x \) into two parts: a mantissa between 0.5 and 1 (returned by the function) and an exponent output as \( e \).

If the value \( x \) is zero, both parts of the result are zero.

For vectors, the returned vector contains the mantissa of each element of the input vector and the output vector contains the exponent of each element of the input vector.

**REFERENCE IMPLEMENTATION**

The example below is not legal Cg because it uses the & address-of operator not supported by Cg in order to call the ANSI C frexp routine.

```c
float3 frexp(float3 x, out float3 e)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++) {
        float eout;
        rv[i] = frexp(a[i], &eout); // this is the ANSI C standard library frexp()
        e[i] = eout;
    }
    return rv;
}
```

**PROFILE SUPPORT**

*frexp* is fully supported in all profiles unless otherwise specified.

Support in the *fp20* is limited to constant compile-time evaluation.

**SEE ALSO**

*exp2, log, pow*

### 9.35 fwidth

**NAME**

*fwidth* - returns sum of approximate window-space partial derivatives magnitudes

**SYNOPSIS**

```c
float fwidth(float a);
float1 fwidth(float1 a);
float2 fwidth(float2 a);
float3 fwidth(float3 a);
float4 fwidth(float4 a);

half fwidth(half a);
half1 fwidth(half1 a);
half2 fwidth(half2 a);
half3 fwidth(half3 a);
```
half4 fwidth(half4 a);
fixed fwidth(fixed a);
fixed1 fwidth(fixed1 a);
fixed2 fwidth(fixed2 a);
fixed3 fwidth(fixed3 a);
fixed4 fwidth(fixed4 a);

PARAMETERS

a

Vector or scalar of which to sum its approximate window-space partial derivative magnitudes, with respect to window-space X and Y.

DESCRIPTION

Returns sum of the absolute values of each approximate partial derivative of a with respect to both the window-space (horizontal) x and (vertical) y coordinate.

For vectors, the returned vector contains the sum of partial derivative magnitudes of each element of the input vector.

This function can be used to approximate the fragment width (hence the name “fwidth”) for level-of-detail computations dependent on change in window-space.

This function is only available in fragment program profiles (but not all of them).

The specific way the partial derivative is computed is implementation-dependent. Typically fragments are rasterized in 2x2 arrangements of fragments (called quad-fragments) and the partial derivatives of a variable is computed by differencing with the adjacent horizontal fragment in the quad-fragment.

The partial derivative computation may incorrect when fwidth is used in control flow paths where not all the fragments within a quad-fragment have branched the same way.

The partial derivative computation may be less exact (wobbly) when the variable is computed based on varying parameters interpolated with centroid interpolation.

REFERENCE IMPLEMENTATION

fmod for float3 vectors could be implemented this way:

float3 fwidth(float3 a)
{
    return abs(ddx(a)) + abs(ddy(a));
}

PROFILE SUPPORT

fwidth is supported only in fragment profiles. Vertex and geometry profiles lack the concept of window space.

fwidth is unsupported in the fp20, ps_1_1, ps_1_2, ps_1_3, and arbfp1 profiles.

SEE ALSO

ddx, ddy, fp30, fp40, gp4fp
9.36 intBitsToFloat

NAME

intBitsToFloat - returns the float value corresponding to a given bit representation of a scalar int value or vector of int values

SYNOPSIS

float intBitsToFloat(int x);
float1 intBitsToFloat(int1 x);
float2 intBitsToFloat(int2 x);
float3 intBitsToFloat(int3 x);
float4 intBitsToFloat(int4 x);

PARAMETERS

x

Integer vector or scalar to raw cast to a scalar float or vector of floats

DESCRIPTION

Returns the IEEE 754 float scalar value or vector values corresponding to a given 32-bit integer bit representation for a scalar int value or vector of int values.

This function is based on Java's java.lang.Float method of the same name. See:

http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Float.html

The Cg compiler can typically optimize intBitsToFloat so it has no instruction cost.

REFERENCE IMPLEMENTATION

intBitsToFloat operates consistent with the following ANSI C code:

float floatToRawIntBits(int x)
{
    union {
        float f; // assuming 32-bit IEEE 754 single-precision
        int i;   // assuming 32-bit 2’s complement int
    } u;

    u.i = x;
    return u.f;
}

PROFILE SUPPORT

intBitsToFloat is supported by the gp4vp, gp4gp, and gp4vp profiles.
intBitsToFloat is not supported by pre-G80 profiles.

SEE ALSO

ceil, floatToIntBits, floatToRawIntBits, floor, round, trunc
9.37 inverse

NAME
inverse - return the inverse matrix of a matrix

SYNOPSIS
float4x4 inverse(float4x4 A)
float3x3 inverse(float3x3 A)
float2x2 inverse(float2x2 A)

PARAMETERS
A
Matrix to invert.

DESCRIPTION
Returns the inverse of the matrix A.

REFERENCE IMPLEMENTATION
inverse for a float2x2 matrix can be implemented like this:

```c
float2x2 inverse(float2x2 A)
{
    float2x2 C;
    float det = determinant(A);
    C[0] = A._m11;
    C[1] = -A._m01;
    C[2] = -A._m10;
    C[3] = A._m00;
    return C / det;
}
```

PROFILE SUPPORT
inverse is supported in all profiles except fp20, vs_1_1, ps_1_1, ps_1_2, and ps_1_3.

SEE ALSO
determinant

9.38 isfinite

NAME
isfinite - test whether or not a scalar or each vector component is a finite value

SYNOPSIS
bool isfinite(float x);
bool1 isfinite(float1 x);
bool2 isfinite(float2 x);
bool3 isfinite(float3 x);
bool4 isfinite(float4 x);
bool  isfinite(half x);
bool1 isfinite(half1 x);
bool2 isfinite(half2 x);
bool3 isfinite(half3 x);
bool4 isfinite(half4 x);
bool  isfinite(fixed x);
bool1 isfinite(fixed1 x);
bool2 isfinite(fixed2 x);
bool3 isfinite(fixed3 x);
bool4 isfinite(fixed4 x);

PARAMETERS

x

Vector or scalar to test for finiteness.

DESCRIPTION

Returns whether or not a scalar or each vector component is a finite value. Infinity and not-a-number (NaN) values are not finite.

REFERENCE IMPLEMENTATION

isfinite for float3 vectors could be implemented like this.

bool3 isfinite(float3 s)
{
    // By IEEE 754 rule, 2*Inf equals Inf
    return (s == s) && (s == 0) || (s != 2*s);
}

PROFILE SUPPORT

isfinite is supported in all profiles except fp20.

SEE ALSO

isinf, isnan

9.39 isinf

NAME

isinf - test whether or not a scalar or each vector component is infinite

SYNOPSIS

bool  isinf(float x);
bool1 isinf(float1 x);
bool2 isinf(float2 x);
bool3 isinf(float3 x);
bool4 isinf(float4 x);
bool isinf(half x);
bool1 isinf(half1 x);
bool2 isinf(half2 x);
bool3 isinf(half3 x);
bool4 isinf(half4 x);

bool isinf(fixed x);
bool1 isinf(fixed1 x);
bool2 isinf(fixed2 x);
bool3 isinf(fixed3 x);
bool4 isinf(fixed4 x);

PARAMETERS

x

Vector or scalar to test if infinite.

DESCRIPTION

Returns whether or not a scalar or each vector component is a (negative or positive) infinite value. Finite and not-a-number (NaN) values are not infinite.

REFERENCE IMPLEMENTATION

isinf for float3 vectors could be implemented like this.

bool3 isinf(float3 s)
{
    // By IEEE 754 rule, 2*Inf equals Inf
    return (2*s == s) && (s != 0);
}

PROFILE SUPPORT

isinf is supported in all profiles except fp20.

SEE ALSO

isfinite, isnan

9.40 isnan

NAME

isnan - test whether or not a scalar or each vector component is not-a-number

SYNOPSIS

bool isnan(float x);
bool1 isnan(float1 x);
bool2 isnan(float2 x);
bool3 isnan(float3 x);
bool4 isnan(float4 x);

bool isnan(half x);
bool1 isnan(half1 x);
bool2 isnan(half2 x);
bool3 isnan(half3 x);
bool4 isnan(half4 x);
bool1 isnan(fixed x);
bool1 isnan(fixed1 x);
bool2 isnan(fixed2 x);
bool3 isnan(fixed3 x);
bool4 isnan(fixed4 x);

PARAMETERS

x
Vector or scalar to test for being NaN.

DESCRIPTION

Returns whether or not a scalar or each vector component is not-a-number (NaN). Finite and infinite values are not NaN.

REFERENCE IMPLEMENTATION

isnan for float3 vectors could be implemented like this.

```c
bool3 isnan(float3 s)
{
    // By IEEE 754 rule, NaN is not equal to NaN
    return s != s;
}
```

PROFILE SUPPORT

isnan is supported in all profiles except fp20.

SEE ALSO

isfinite, isninf

9.41 ldexp

NAME

ldexp - returns $x \times 2^n$

SYNOPSIS

```c
float ldexp(float x, float n);
float1 ldexp(float1 x, float1 n);
float2 ldexp(float2 x, float2 n);
float3 ldexp(float3 x, float3 n);
float4 ldexp(float4 x, float4 n);

half ldexp(half x, half n);
half1 ldexp(half1 x, half1 n);
half2 ldexp(half2 x, half2 n);
half3 ldexp(half3 x, half3 n);
half4 ldexp(half4 x, half4 n);
```
fixed ldexp(fixed x, fixed n);
fixed1 ldexp(fixed1 x, fixed1 n);
fixed2 ldexp(fixed2 x, fixed2 n);
fixed3 ldexp(fixed3 x, fixed3 n);
fixed4 ldexp(fixed4 x, fixed4 n);

PARAMETERS

x
    Vector or scalar.

n
    Vector or scalar for power with which to raise 2.

DESCRIPTION

ldexp returns \( x \) times 2 raised to the power \( n \).

REFERENCE IMPLEMENTATION

ldexp for float2 vectors \( x \) and \( n \) could be implemented as:

```c
float2 ldexp(float2 x, float2 n)
{
    return x * exp2(n);
}
```

PROFILE SUPPORT

ldexp is supported in all profiles but fp20.

SEE ALSO

exp2, modf, pow

9.42 length

NAME

length - return scalar Euclidean length of a vector

SYNOPSIS

```c
float length(float v);
float length(float1 v);
float length(float2 v);
float length(float3 v);
float length(float4 v);

half length(half v);
half length(half1 v);
half length(half2 v);
half length(half3 v);
half length(half4 v);

fixed length(fixed v);
fixed length(fixed1 v);
```
fixed length(fixed2 v);
fixed length(fixed3 v);
fixed length(fixed4 v);

PARAMETERS

v
Vector of which to determine the length.

DESCRIPTION

Returns the Euclidean length of a vector.

REFERENCE IMPLEMENTATION

length for a float3 vector could be implemented like this.

```cpp
float length(float3 v)
{
    return sqrt(dot(v,v));
}
```

PROFILE SUPPORT

length is supported in all profiles.
Support in the fp20 is limited.

SEE ALSO

max, normalize, sqrt, dot

9.43 lerp

NAME

lerp - returns linear interpolation of two scalars or vectors based on a weight

SYNOPSIS

```cpp
float lerp(float a, float b, float w);
float1 lerp(float1 a, float1 b, float1 w);
float2 lerp(float2 a, float2 b, float2 w);
float3 lerp(float3 a, float3 b, float3 w);
float4 lerp(float4 a, float4 b, float4 w);

float1 lerp(float1 a, float1 b, float w);
float2 lerp(float2 a, float2 b, float w);
float3 lerp(float3 a, float3 b, float w);
float4 lerp(float4 a, float4 b, float w);

half lerp(half a, half b, half w);
half1 lerp(half1 a, half1 b, half1 w);
half2 lerp(half2 a, half2 b, half2 w);
half3 lerp(half3 a, half3 b, half3 w);
half4 lerp(half4 a, half4 b, half4 w);
```
half1 lerp(half1 a, half1 b, half w);
half2 lerp(half2 a, half2 b, half w);
half3 lerp(half3 a, half3 b, half w);
half4 lerp(half4 a, half4 b, half w);

fixed lerp(fixed a, fixed b, fixed w);
fixed1 lerp(fixed1 a, fixed1 b, fixed1 w);
fixed2 lerp(fixed2 a, fixed2 b, fixed2 w);
fixed3 lerp(fixed3 a, fixed3 b, fixed3 w);
fixed4 lerp(fixed4 a, fixed4 b, fixed4 w);

fixed1 lerp(fixed1 a, fixed1 b, fixed w);
fixed2 lerp(fixed2 a, fixed2 b, fixed w);
fixed3 lerp(fixed3 a, fixed3 b, fixed w);
fixed4 lerp(fixed4 a, fixed4 b, fixed w);

PARAMETERS

a
Vector or scalar to weight; returned when w is zero.

b
Vector or scalar to weight; returned when w is one.

w
Vector or scalar weight.

DESCRIPTION

Returns the linear interpolation of a and b based on weight w.

a and b are either both scalars or both vectors of the same length. The weight w may be a scalar or a vector of the same length as a and b. w can be any value (so is not restricted to be between zero and one); if w has values outside the [0,1] range, it actually extrapolates.

lerp returns a when w is zero and returns b when w is one.

REFERENCE IMPLEMENTATION

lerp for float3 vectors for a and b and a float w could be implemented like this:

float3 lerp(float3 a, float3 b, float w)
{
    return a + w*(b-a);
}

PROFILE SUPPORT

lerp is supported in all profiles.

SEE ALSO

saturate, smoothstep, step
9.44 lit

NAME
lit - computes lighting coefficients for ambient, diffuse, and specular lighting contributions

SYNOPSIS
float4 lit(float NdotL, float NdotH, float m);

half4 lit(half NdotL, half NdotH, half m);

fixed4 lit(fixed NdotL, fixed NdotH, fixed m);

PARAMETERS

NdotL
The dot product of a normalized surface normal and a normalized light vector.

NdotH
The dot product of a normalized surface normal and a normalized half-angle vector (for Blinn-style specular) where the half-angle vector is the sum of the normalized view vector and normalized light vector. Alternatively, the dot product of a normalized light vector and a normalized view vector reflected by a normalized surface normal could be used (for Phong-style specular).

m
A specular exponent, typically described as a measure of shininess. The larger the exponent, the shinier the specular highlight, the smaller the exponent, the duller the specular highlight.

DESCRIPTION
The lit function is a helper function useful to compute lighting coefficients for ambient, diffuse, and specular lighting contributions. The function efficiently maps to a native instruction for most GPUs.

lit returns a 4-component vector arranged as follows:

x
The ambient coefficient that is always 1.0.

y
The diffuse coefficient that is zero if NdotL is less than zero, and NdotL otherwise.

z
The specular coefficient that is zero if either NdotL or NdotH are less than zero, and otherwise NdotH raised to the power m.

w
Always 1.0.

REFERENCE IMPLEMENTATION

lit accepting <float> parameters could be implemented this way:

float4 lit(float NdotL, float NdotH, float m)
{
    float specular = (NdotL > 0) ? pow(max(0.0, NdotH), m);
}
PROFILE SUPPORT

lit is supported in all profiles.

SEE ALSO

dot, max, normalize, pow

9.45 log10

NAME

log10 - returns the base-10 logarithm of scalars and vectors

SYNOPSIS

float log10(float a);
float1 log10(float1 a);
float2 log10(float2 a);
float3 log10(float3 a);
float4 log10(float4 a);

half log10(half a);
half1 log10(half1 a);
half2 log10(half2 a);
half3 log10(half3 a);
half4 log10(half4 a);

fixed log10(fixed a);
fixed1 log10(fixed1 a);
fixed2 log10(fixed2 a);
fixed3 log10(fixed3 a);
fixed4 log10(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the base-10 logarithm.

DESCRIPTION

Returns the base-10 logarithm a.

For vectors, the returned vector contains the base-10 logarithm of each element of the input vector.

REFERENCE IMPLEMENTATION

float3 log10(float3 a)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++) {
        rv[i] = log10(a[i]);  // this is the ANSI C standard library log10()
log10 is typically implemented with a native base-10 logarithm instruction.

PROFILE SUPPORT

log10 is fully supported in all profiles unless otherwise specified. Support in the fp20 is limited to constant compile-time evaluation.

SEE ALSO

exp, log, log2, pow

9.46 log2

NAME

log2 - returns the base-2 logarithm of scalars and vectors

SYNOPSIS

float log2(float a);
float1 log2(float1 a);
float2 log2(float2 a);
float3 log2(float3 a);
float4 log2(float4 a);
half log2(half a);
half1 log2(half1 a);
half2 log2(half2 a);
half3 log2(half3 a);
half4 log2(half4 a);
fixed log2(fixed a);
fixed1 log2(fixed1 a);
fixed2 log2(fixed2 a);
fixed3 log2(fixed3 a);
fixed4 log2(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the base-2 logarithm.

DESCRIPTION

Returns the base-2 logarithm a.

For vectors, the returned vector contains the base-2 logarithm of each element of the input vector.

REFERENCE IMPLEMENTATION

float3 log2(float3 a)
{
    float3 rv;
```c
int i;

for (i=0; i<3; i++) {
    rv[i] = log2(a[i]); // this is the ANSI C standard library log2()
}
return rv;
}
```

`log2` is typically implemented with a native base-2 logarithm instruction.

**PROFILE SUPPORT**

`log2` is fully supported in all profiles unless otherwise specified.
Support in the `fp20` is limited to constant compile-time evaluation.

**SEE ALSO**

`exp`, `log`, `log10`, `pow`

### 9.47 log

**NAME**

log - returns the natural logarithm of scalars and vectors

**SYNOPSIS**

```c
float log(float a);
float1 log(float1 a);
float2 log(float2 a);
float3 log(float3 a);
float4 log(float4 a);

half log(half a);
half1 log(half1 a);
half2 log(half2 a);
half3 log(half3 a);
half4 log(half4 a);

fixed log(fixed a);
fixed1 log(fixed1 a);
fixed2 log(fixed2 a);
fixed3 log(fixed3 a);
fixed4 log(fixed4 a);
```

**PARAMETERS**

```c
    a
```

Vector or scalar of which to determine the natural logarithm.

**DESCRIPTION**

Returns the natural logarithm `a`.

For vectors, the returned vector contains the natural logarithm of each element of the input vector.
REFERENCE IMPLEMENTATION

```c
float3 log(float3 a)
{
    float3 rv;
    int i;

    for (i=0; i<3; i++) {
        rv[i] = log(a[i]); // this is the ANSI C standard library log()
    }
    return rv;
}
```

`log` is typically implemented with a native base-2 logarithm instruction.

PROFILE SUPPORT

`log` is fully supported in all profiles unless otherwise specified.

Support in the `fp20` is limited to constant compile-time evaluation.

SEE ALSO

`exp, log10, log2, pow`

9.48 max

NAME

`max` - returns the maximum of two scalars or each respective component of two vectors

SYNOPSIS

```c
float max(float a, float b);
float1 max(float1 a, float1 b);
float2 max(float2 a, float2 b);
float3 max(float3 a, float3 b);
float4 max(float4 a, float4 b);

half max(half a, half b);
half1 max(half1 a, half1 b);
half2 max(half2 a, half2 b);
half3 max(half3 a, half3 b);
half4 max(half4 a, half4 b);

fixed max(fixed a, fixed b);
fixed1 max(fixed1 a, fixed1 b);
fixed2 max(fixed2 a, fixed2 b);
fixed3 max(fixed3 a, fixed3 b);
fixed4 max(fixed4 a, fixed4 b);
```

PARAMETERS

- **a**
  - Scalar or vector.
- **b**
Scalar or vector.

DESCRIPTION

Returns the maximum of two same-typed scalars \(a\) and \(b\) or the respective components of two same-typed vectors \(a\) and \(b\). The result is a three-component vector.

REFERENCE IMPLEMENTATION

\texttt{max} for \texttt{float3} vectors could be implemented this way:

```c
float3 max(float3 a, float3 b)
{
    return float3(a.x > b.x ? a.x : b.x,
                  a.y > b.y ? a.y : b.y,
                  a.z > b.z ? a.z : b.z);
}
```

PROFILE SUPPORT

\texttt{max} is supported in all profiles. \texttt{max} is implemented as a compiler built-in.

Support in the \texttt{fp20} is limited.

SEE ALSO

\textit{clamp, min}

9.49 min

NAME

\texttt{min} - returns the minimum of two scalars or each respective component of two vectors

SYNOPSIS

```c
float min(float a, float b);
float1 min(float1 a, float1 b);
float2 min(float2 a, float2 b);
float3 min(float3 a, float3 b);
float4 min(float4 a, float4 b);
half min(half a, half b);
half1 min(half1 a, half1 b);
half2 min(half2 a, half2 b);
half3 min(half3 a, half3 b);
half4 min(half4 a, half4 b);
fixed min(fixed a, fixed b);
fixed1 min(fixed1 a, fixed1 b);
fixed2 min(fixed2 a, fixed2 b);
fixed3 min(fixed3 a, fixed3 b);
fixed4 min(fixed4 a, fixed4 b);
```

PARAMETERS

\(a\)

Scalar or vector.
b
Scalar or vector.

**DESCRIPTION**

Returns the minimum of two same-typed scalars $a$ and $b$ or the respective components of two same-typed vectors $a$ and $b$. The result is a three-component vector.

**REFERENCE IMPLEMENTATION**

`min` for `float3` vectors could be implemented this way:

```c
float3 min(float3 a, float3 b)
{
    return float3(a.x < b.x ? a.x : b.x,
                   a.y < b.y ? a.y : b.y,
                   a.z < b.z ? a.z : b.z);
}
```

**PROFILE SUPPORT**

`min` is supported in all profiles. `min` is implemented as a compiler built-in.

Support in the `fp20` is limited.

**SEE ALSO**

`clamp`, `max`

### 9.50 modf

**NAME**

`modf` - decompose a float into integer and fractional parts

**SYNOPSIS**

```c
float modf(float x, out float i);
float1 modf(float1 x, out float1 i);
float2 modf(float2 x, out float2 i);
float3 modf(float3 x, out float3 i);
float4 modf(float4 x, out float4 i);

half modf(half x, out half i);
half1 modf(half1 x, out half1 i);
half2 modf(half2 x, out half2 i);
half3 modf(half3 x, out half3 i);
half4 modf(half4 x, out half4 i);

fixed modf(fixed x, out fixed i);
fixed1 modf(fixed1 x, out fixed1 i);
fixed2 modf(fixed2 x, out fixed2 i);
fixed3 modf(fixed3 x, out fixed3 i);
fixed4 modf(fixed4 x, out fixed4 i);
```
PARAMETERS

x
Vector or scalar.

i
Vector or scalar for integer part of x.

DESCRIPTION

modf returns the integer and fractional parts of x.

REFERENCE IMPLEMENTATION

PROFILE SUPPORT

modf is supported in all profiles except fp20.

SEE ALSO

fmod, round

9.51 mul

NAME

mul - multiply a matrix by a column vector, row vector by a matrix, or matrix by a matrix

SYNOPSIS

float4 mul(float4x4 M, float4 v);
float4 mul(float4x3 M, float3 v);
float4 mul(float4x2 M, float2 v);
float4 mul(float4x1 M, float1 v);
float3 mul(float3x4 M, float4 v);
float3 mul(float3x3 M, float3 v);
float3 mul(float3x2 M, float2 v);
float3 mul(float3x1 M, float1 v);
float2 mul(float2x4 M, float4 v);
float2 mul(float2x3 M, float3 v);
float2 mul(float2x2 M, float2 v);
float2 mul(float2x1 M, float1 v);
float1 mul(float1x4 M, float4 v);
float1 mul(float1x3 M, float3 v);
float1 mul(float1x2 M, float2 v);
float1 mul(float1x1 M, float1 v);

float4 mul(float4 v, float4x4 M);
float4 mul(float3 v, float3x4 M);
float4 mul(float2 v, float2x4 M);
float4 mul(float1 v, float1x4 M);
float3 mul(float4 v, float4x3 M);
float3 mul(float3 v, float3x3 M);
float3 mul(float2 v, float2x3 M);
float3 mul(float1 v, float1x3 M);
float2 mul(float4 v, float4x2 M);
float2 mul(float3 v, float3x2 M);
float2 mul(float2 v, float2x2 M);
float2 mul(float1 v, float1x2 M);
float1 mul(float4 v, float4x1 M);
float1 mul(float3 v, float3x1 M);
float1 mul(float2 v, float2x1 M);
float1 mul(float1 v, float1x1 M);

half4 mul(half4x4 M, half4 v);
half4 mul(half4x3 M, half3 v);
half4 mul(half4x2 M, half2 v);
half4 mul(half4x1 M, half1 v);
half3 mul(half3x4 M, half4 v);
half3 mul(half3x3 M, half3 v);
half3 mul(half3x2 M, half2 v);
half3 mul(half3x1 M, half1 v);
half2 mul(half2x4 M, half4 v);
half2 mul(half2x3 M, half3 v);
half2 mul(half2x2 M, half2 v);
half2 mul(half2x1 M, half1 v);
half1 mul(half1x4 M, half4 v);
half1 mul(half1x3 M, half3 v);
half1 mul(half1x2 M, half2 v);
half1 mul(half1x1 M, half1 v);

half4 mul(half4 v, half4x4 M);
half4 mul(half3 v, half3x4 M);
half4 mul(half2 v, half2x4 M);
half4 mul(half1 v, half1x4 M);
half3 mul(half4 v, half4x3 M);
half3 mul(half3 v, half3x3 M);
half3 mul(half2 v, half2x3 M);
half3 mul(half1 v, half1x3 M);
half2 mul(half4 v, half4x2 M);
half2 mul(half3 v, half3x2 M);
half2 mul(half2 v, half2x2 M);
half2 mul(half1 v, half1x2 M);
half1 mul(half4 v, half4x1 M);
half1 mul(half3 v, half3x1 M);
half1 mul(half2 v, half2x1 M);
half1 mul(half1 v, half1x1 M);

fixed4 mul(fixed4x4 M, fixed4 v);
fixed4 mul(fixed4x3 M, fixed3 v);
fixed4 mul(fixed4x2 M, fixed2 v);
fixed4 mul(fixed4x1 M, fixed1 v);
fixed3 mul(fixed3x4 M, fixed4 v);
fixed3 mul(fixed3x3 M, fixed3 v);
fixed3 mul(fixed3x2 M, fixed2 v);
fixed3 mul(fixed3x1 M, fixed1 v);
fixed2 mul(fixed2x4 M, fixed4 v);
fixed2 mul(fixed2x3 M, fixed3 v);
fixed2 mul(fixed2x2 M, fixed2 v);
fixed2 mul(fixed2x1 M, fixed1 v);
fixed1 mul(fixed1x4 M, fixed4 v);
fixed1 mul(fixed1x3 M, fixed3 v);
fixed1 mul(fixed1x2 M, fixed2 v);
fixed1 mul(fixed1x1 M, fixed1 v);
fixed4 mul(fixed4 v, fixed4x4 M);
fixed4 mul(fixed3 v, fixed3x4 M);
fixed4 mul(fixed2 v, fixed2x4 M);
fixed4 mul(fixed1 v, fixed1x4 M);
fixed3 mul(fixed4 v, fixed4x3 M);
fixed3 mul(fixed3 v, fixed3x3 M);
fixed3 mul(fixed2 v, fixed2x3 M);
fixed3 mul(fixed1 v, fixed1x3 M);
fixed2 mul(fixed4 v, fixed4x2 M);
fixed2 mul(fixed3 v, fixed3x2 M);
fixed2 mul(fixed2 v, fixed2x2 M);
fixed2 mul(fixed1 v, fixed1x2 M);
fixed1 mul(fixed4 v, fixed4x1 M);
fixed1 mul(fixed3 v, fixed3x1 M);
fixed1 mul(fixed2 v, fixed2x1 M);
fixed1 mul(fixed1 v, fixed1x1 M);

float1x1 mul(float1x1 A, float1x1 B);
float1x2 mul(float1x1 A, float1x2 B);
float1x3 mul(float1x1 A, float1x3 B);
float1x4 mul(float1x1 A, float1x4 B);

float1x1 mul(float1x2 A, float2x1 B);
float1x2 mul(float1x2 A, float2x2 B);
float1x3 mul(float1x2 A, float2x3 B);
float1x4 mul(float1x2 A, float2x4 B);

float1x1 mul(float1x3 A, float3x1 B);
float1x2 mul(float1x3 A, float3x2 B);
float1x3 mul(float1x3 A, float3x3 B);
float1x4 mul(float1x3 A, float3x4 B);

float1x1 mul(float1x4 A, float4x1 B);
float1x2 mul(float1x4 A, float4x2 B);
float1x3 mul(float1x4 A, float4x3 B);
float1x4 mul(float1x4 A, float4x4 B);

float2x1 mul(float2x1 A, float1x1 B);
float2x2 mul(float2x1 A, float1x2 B);
float2x3 mul(float2x1 A, float1x3 B);
float2x4 mul(float2x1 A, float1x4 B);

float2x1 mul(float2x2 A, float2x1 B);
float2x2 mul(float2x2 A, float2x2 B);
float2x3 mul(float2x2 A, float2x3 B);
float2x4 mul(float2x2 A, float2x4 B);

float2x1 mul(float2x3 A, float3x1 B);
float2x2 mul(float2x3 A, float3x2 B);
float2x3 mul(float2x3 A, float3x3 B);
float2x4 mul(float2x3 A, float3x4 B);

float2x1 mul(float2x4 A, float4x1 B);
float2x2 mul(float2x4 A, float4x2 B);
float2x3 mul(float2x4 A, float4x3 B);
float2x4 mul(float2x4 A, float4x4 B);

float3x1 mul(float3x1 A, float1x1 B);
float3x2 mul(float3x1 A, float1x2 B);
float3x3 mul(float3x1 A, float1x3 B);
float3x4 mul(float3x1 A, float1x4 B);

float3x1 mul(float3x2 A, float2x1 B);
float3x2 mul(float3x2 A, float2x2 B);
float3x3 mul(float3x2 A, float2x3 B);
float3x4 mul(float3x2 A, float2x4 B);

float3x1 mul(float3x3 A, float3x1 B);
float3x2 mul(float3x3 A, float3x2 B);
float3x3 mul(float3x3 A, float3x3 B);
float3x4 mul(float3x3 A, float3x4 B);

float3x1 mul(float3x4 A, float4x1 B);
float3x2 mul(float3x4 A, float4x2 B);
float3x3 mul(float3x4 A, float4x3 B);
float3x4 mul(float3x4 A, float4x4 B);

float4x1 mul(float4x1 A, float1x1 B);
float4x2 mul(float4x1 A, float1x2 B);
float4x3 mul(float4x1 A, float1x3 B);
float4x4 mul(float4x1 A, float1x4 B);

float4x1 mul(float4x2 A, float2x1 B);
float4x2 mul(float4x2 A, float2x2 B);
float4x3 mul(float4x2 A, float2x3 B);
float4x4 mul(float4x2 A, float2x4 B);

float4x1 mul(float4x3 A, float3x1 B);
float4x2 mul(float4x3 A, float3x2 B);
float4x3 mul(float4x3 A, float3x3 B);
float4x4 mul(float4x3 A, float3x4 B);

float4x1 mul(float4x4 A, float4x1 B);
float4x2 mul(float4x4 A, float4x2 B);
float4x3 mul(float4x4 A, float4x3 B);
float4x4 mul(float4x4 A, float4x4 B);

PARAMETERS

M
Matrix

v
Vector

A
Matrix

B
Matrix

DESCRIPTION

Returns the vector result of multiplying a matrix $M$ by a column vector $v$; a row vector $v$ by a matrix $M$; or a matrix $A$ by a second matrix $B$. 
The following are algebraically equal (if not necessarily numerically equal):

\[
\text{mul}(M, v) = \text{mul}(v, \text{transpose}(M))
\]
\[
\text{mul}(v, M) = \text{mul}(\text{transpose}(M), v)
\]

REFERENCES IMPLEMENTATION

\text{mul} for a \text{float4x3} matrix by a \text{float3} column vector could be implemented this way:

\[
\text{float4 mul(float4x3 M, float3 v)}
\]
\[
\{
\text{float4 r;}
\]
\[
\begin{align*}
\text{r.x} &= \text{dot}( M._m00_m01_m02, v ); \\
\text{r.y} &= \text{dot}( M._m10_m11_m12, v ); \\
\text{r.z} &= \text{dot}( M._m20_m21_m22, v ); \\
\text{r.w} &= \text{dot}( M._m30_m31_m32, v );
\end{align*}
\]
\[
\text{return r;}
\]

PROFILE SUPPORT

\text{mul} is supported in all profiles except the \text{fp20}, \text{ps\_1\_1}, \text{ps\_1\_2}, and \text{ps\_1\_3} fragment profiles.

The \text{fixed3} matrix-by-vector and vector-by-matrix multiplies are very efficient in the \text{fp30} profile.

SEE ALSO

\text{cross, dot, transpose}

9.52 normalize

NAME

\text{normalize} - normalizes a vector

SYNOPSIS

\[
\begin{align*}
\text{float normalize(float v);} \\
\text{float normalize(float1 v);} \\
\text{float normalize(float2 v);} \\
\text{float normalize(float3 v);} \\
\text{float normalize(float4 v);} \\
\text{half normalize(half v);} \\
\text{half normalize(half1 v);} \\
\text{half normalize(half2 v);} \\
\text{half normalize(half3 v);} \\
\text{half normalize(half4 v);} \\
\text{fixed normalize(fixed v);} \\
\text{fixed normalize(fixed1 v);} \\
\text{fixed normalize(fixed2 v);} \\
\text{fixed normalize(fixed3 v);} \\
\text{fixed normalize(fixed4 v);} \\
\end{align*}
\]
PARAMETERS

\( v \)

Vector to normalize.

DESCRIPTION

Returns the normalized version of a vector, meaning a vector in the same direction as the original vector but with a Euclidean length of one.

REFERENCE IMPLEMENTATION

\( \text{normalize} \) for a \( \text{float3} \) vector could be implemented like this.

```c
float3 normalize(float3 v)
{
    return rsqrt(dot(v,v))*v;
}
```

PROFILE SUPPORT

\( \text{normalize} \) is supported in all profiles except \( \text{fp20} \).

SEE ALSO

\( \text{distance} \), \( \text{dot} \), \( \text{length} \), \( \text{rsqrt} \), \( \text{sqrt} \)

9.53 pack

NAME

pack - packs multiple values into a single 32-bit result

SYNOPSIS

```c
float pack_2half(float2 a);
float pack_2half(half2 a);
float pack_2ushort(float2 a);
float pack_2ushort(half2 a);
float pack_4byte(float4 a);
float pack_4byte(half4 a);
float pack_4ubyte(float4 a);
float pack_4ubyte(half4 a);
```

PARAMETERS

\( a \)

Value to pack

DESCRIPTION

\( \text{pack\_2half} \) converts the components of \( a \) into a pair of 16-bit floating point values. The two converted components are then packed into a single 32-bit result.

\( \text{pack\_2ushort} \) converts the components of \( a \) into a pair of 16-bit unsigned integers. The two converted components are then packed into a single 32-bit return value.
pack_4byte converts the four components of a into 8-bit signed integers. The signed integers are such that a representation with all bits set to 0 corresponds to the value \(-128/127\), and a representation with all bits set to 1 corresponds to \(+127/127\). The four signed integers are then packed into a single 32-bit result.

pack_4ubyte converts the four components of a into 8-bit unsigned integers. The unsigned integers are such that a representation with all bits set to 0 corresponds to 0.0, and a representation with all bits set to 1 corresponds to 1.0. The four unsigned integers are then packed into a single 32-bit result.

**REFERENCE IMPLEMENTATION**

pack_2half could be implemented this way:

```c
float pack_2half(float2 a)
{
    float result;
    return result = ((half)a.y) << 16) | (half)a.x;
}
```

pack_2ushort could be implemented this way:

```c
float pack_2ushort(float2 a)
{
    float result;
    ushort.x = round(65535.0 * clamp(a.x, 0.0, 1.0));
    ushort.y = round(65535.0 * clamp(a.y, 0.0, 1.0));
    result = (ushort.y << 16) | ushort.y;
    return result;
}
```

pack_4byte could be implemented this way:

```c
float pack_4byte(half2 a)
{
    float result;
    ub.y = round(127 * clamp(a.y, -128/127, 127/127) + 128);
    ub.z = round(127 * clamp(a.z, -128/127, 127/127) + 128);
    ub.w = round(127 * clamp(a.w, -128/127, 127/127) + 128);
    return result = (ub.w << 24) | (ub.z << 16) | (ub.y << 8) | ub.x;
}
```

pack_4ubyte could be implemented this way:

```c
float pack_4ubyte(float4 a)
{
    float result;
    ub.x = round(255.0 * clamp(a.x, 0.0, 1.0));
    ub.y = round(255.0 * clamp(a.y, 0.0, 1.0));
    ub.z = round(255.0 * clamp(a.z, 0.0, 1.0));
    ub.w = round(255.0 * clamp(a.w, 0.0, 1.0));
    result = (ub.w << 24) | (ub.z << 16) | (ub.y << 8) | ub.x;
    return result;
}
```

**PROFILE SUPPORT**

pack is supported in *fp30, fp40, gp4, gp5*

**SEE ALSO**

unpack
9.54 pow

NAME

pow - returns \( x \) to the \( y \)-th power of scalars and vectors

SYNOPSIS

float pow(float \( x \), float \( y \));
float1 pow(float1 \( x \), float1 \( y \));
float2 pow(float2 \( x \), float2 \( y \));
float3 pow(float3 \( x \), float3 \( y \));
float4 pow(float4 \( x \), float4 \( y \));

half pow(half \( x \), half \( y \));
half1 pow(half1 \( x \), half1 \( y \));
half2 pow(half2 \( x \), half2 \( y \));
half3 pow(half3 \( x \), half3 \( y \));
half4 pow(half4 \( x \), half4 \( y \));

fixed pow(fixed \( x \), fixed \( y \));
fixed1 pow(fixed1 \( x \), fixed1 \( y \));
fixed2 pow(fixed2 \( x \), fixed2 \( y \));
fixed3 pow(fixed3 \( x \), fixed3 \( y \));
fixed4 pow(fixed4 \( x \), fixed4 \( y \));

PARAMETERS

\( x \)

A base value.

\( y \)

The power to raise the base.

DESCRIPTION

Returns \( x \) to the power \( y \).

For vectors, the returned vector contains the power of each element of the base vector raised to the respective element of the exponent vector.

REFERENCE IMPLEMENTATION

pow for float3 vectors could be implemented this way:

```c
float3 pow(float3 \( x \), float3 \( y \))
{
    float3 rv;

    for (int \( i=0; \) i<3; i++) {
        rv[i] = exp(\( x[i] \) * log(\( y[i] \)));
    }
    return rv;
}
```

PROFILE SUPPORT

exp is supported in all profiles.
Support in the \textit{fp20} is limited to constant compile-time evaluation.

\textbf{SEE ALSO}

\textit{exp, lit, log, rsqrt, sqrt}

\section*{9.55 radians}

\textbf{NAME}

\texttt{radians} - converts values of scalars and vectors from degrees to radians

\textbf{SYNOPSIS}

\begin{verbatim}
float radians(float a);
float1 radians(float1 a);
float2 radians(float2 a);
float3 radians(float3 a);
float4 radians(float4 a);

half radians(half a);
half1 radians(half1 a);
half2 radians(half2 a);
half3 radians(half3 a);
half4 radians(half4 a);

fixed radians(fixed a);
fixed1 radians(fixed1 a);
fixed2 radians(fixed2 a);
fixed3 radians(fixed3 a);
fixed4 radians(fixed4 a);
\end{verbatim}

\textbf{PARAMETERS}

\begin{verbatim}
a
Vector or scalar of which to convert from degrees to radians.
\end{verbatim}

\textbf{DESCRIPTION}

Returns the scalar or vector converted from degrees to radians.

For vectors, the returned vector contains each element of the input vector converted from degrees to radians.

\textbf{REFERENCE IMPLEMENTATION}

\texttt{radians} for a \texttt{float} scalar could be implemented like this.

\begin{verbatim}
float radians(float a)
{
    return 0.017453292 * a;
}
\end{verbatim}

\textbf{PROFILE SUPPORT}

\texttt{radians} is supported in all profiles except \textit{fp20}.

\textbf{SEE ALSO}

\textit{cos, degrees, sin, tan}
9.56 reflect

NAME

reflect - returns the reflectiton vector given an incidence vector and a normal vector.

SYNOPSIS

float reflect(float i, float n);
float2 reflect(float2 i, float2 n);
float3 reflect(float3 i, float3 n);
float4 reflect(float4 i, float4 n);

PARAMETERS

i
    Incidence vector.

n
    Normal vector.

DESCRIPTION

Returns the reflectiton vector given an incidence vector \( i \) and a normal vector \( n \). The resulting vector is the identical number of components as the two input vectors.

The normal vector \( n \) should be normalized. If \( n \) is normalized, the output vector will have the same length as the input incidence vector \( i \).

REFERENCE IMPLEMENTATION

reflect for float3 vectors could be implemented this way:

```cpp
float3 reflect(float3 i, float3 n)
{
    return i - 2.0 * n * dot(n, i);
}
```

PROFILE SUPPORT

reflect is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

dot, length, refract

9.57 refract

NAME

refract - computes a refraction vector.
SYNOPSIS

fixed3 refract(fixed3 i, fixed3 n, fixed eta);
half3 refract(half3 i, half3 n, half eta);
float3 refract(float3 i, float3 n, float eta);

PARAMETERS

i
  Incidence vector.

n
  Normal vector.

eta
  Ratio of indices of refraction at the surface interface.

DESCRIPTION

Returns a refraction vector given an incidence vector, a normal vector for a surface, and a ratio of indices of refraction at the surface’s interface.

The incidence vector \( i \) and normal vector \( n \) should be normalized.

REFERENCE IMPLEMENTATION

reflect for float3 vectors could be implemented this way:

```c
float3 refract( float3 i, float3 n, float eta )
{
  float cosi = dot(-i, n);
  float cost2 = 1.0f - eta * eta * (1.0f - cosi*cosi);
  float3 t = eta*i + ((eta*cosi - sqrt(abs(cost2))) * n);
  return t * (float3)(cost2 > 0);
}
```

PROFILE SUPPORT

refract is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

abs, cos, dot, reflect, sqrt

9.58 round

NAME

round - returns the rounded value of scalars or vectors

SYNOPSIS

```c
float round(float a);
float1 round(float1 a);
float2 round(float2 a);
float3 round(float3 a);
```
float4 round(float4 a);
half   round(half  a);
half1  round(half1 a);
half2  round(half2 a);
half3  round(half3 a);
half4  round(half4 a);
fixed  round(fixed a);
fixed1 round(fixed1 a);
fixed2 round(fixed2 a);
fixed3 round(fixed3 a);
fixed4 round(fixed4 a);

PARAMETERS

    a

Scalar or vector.

DESCRIPTION

Returns the rounded value of a scalar or vector.

For vectors, the returned vector contains the rounded value of each element of the input vector.

The round operation returns the nearest integer to the operand. The value returned by round() if the fractional portion of the operand is 0.5 is profile dependent. On older profiles without built-in round() support, round-to-nearest up rounding is used. On profiles newer than fp40/vp40, round-to-nearest even is used.

REFERENCE IMPLEMENTATION

round for float could be implemented this way:

// round-to-nearest even profiles
float round(float a)
{
    float x = a + 0.5;
    float f = floor(x);
    float r;
    if (x == f) {
        if (a > 0)
            r = f - fmod(f, 2);
        else
            r = f + fmod(f, 2);
    } else
        r = f;
    return r;
}

// round-to-nearest up profiles
float round(float a)
{
    return floor(x + 0.5);
}

PROFILE SUPPORT

round is supported in all profiles except fp20.
SEE ALSO
ceil, floor, fmod, trunc

9.59 rsqrt

NAME
rsqrt - returns reciprocal square root of scalars and vectors.

SYNOPSIS

float rsqrt(float a);
float1 rsqrt(float1 a);
float2 rsqrt(float2 a);
float3 rsqrt(float3 a);
float4 rsqrt(float4 a);

half rsqrt(half a);
half1 rsqrt(half1 a);
half2 rsqrt(half2 a);
half3 rsqrt(half3 a);
half4 rsqrt(half4 a);

fixed rsqrt(fixed a);
fixed1 rsqrt(fixed1 a);
fixed2 rsqrt(fixed2 a);
fixed3 rsqrt(fixed3 a);
fixed4 rsqrt(fixed4 a);

PARAMETERS

a
Vector or scalar of which to determine the reciprocal square root.

DESCRIPTION

Returns an approximation to the reciprocal square root of a.

For vectors, the returned vector contains the reciprocal square root of each element of the input vector.

The reciprocal square root of zero is ideally infinity. The square root of negative values ideally returns NaN (Not a Number).

REFERENCE IMPLEMENTATION

rsqrt is best implemented as a native reciprocal square root instruction, however rsqrt may be implemented via a pow function:

float3 rsqrt(float3 a)
{
    return pow(a, -0.5);
}

PROFILE SUPPORT

rsqrt is supported in all profiles unless otherwise specified. rsqrt is unsupported in the fp20 profile.

In certain profiles such as vp20, rsqrt computes the absolute value of a so negative values of a will not return NaN.
9.60 saturate

NAME

saturate - returns smallest integer not less than a scalar or each vector component.

SYNOPSIS

float saturate(float x);
float1 saturate(float1 x);
float2 saturate(float2 x);
float3 saturate(float3 x);
float4 saturate(float4 x);

half saturate(half x);
half1 saturate(half1 x);
half2 saturate(half2 x);
half3 saturate(half3 x);
half4 saturate(half4 x);

fixed saturate(fixed x);
fixed1 saturate(fixed1 x);
fixed2 saturate(fixed2 x);
fixed3 saturate(fixed3 x);
fixed4 saturate(fixed4 x);

PARAMETERS

x

Vector or scalar to saturate.

DESCRIPTION

Returns \( x \) saturated to the range \([0,1]\) as follows:

1. Returns 0 if \( x \) is less than 0; else
2. Returns 1 if \( x \) is greater than 1; else
3. Returns \( x \) otherwise.

For vectors, the returned vector contains the saturated result of each element of the vector \( x \) saturated to \([0,1]\).

REFERENCE IMPLEMENTATION

saturate for float scalars could be implemented like this.

```c
float saturate(float x)
{
    return max(0, min(1, x));
}
```

PROFILE SUPPORT

saturate is supported in all profiles.
saturate is very efficient in the fp20, fp30, and fp40 profiles.
SEE ALSO

clamp, max, min

9.61 sign

NAME

sign - returns sign of scalar or each vector component.

SYNOPSIS

float sign(float x);
float1 sign(float1 x);
float2 sign(float2 x);
float3 sign(float3 x);
float4 sign(float4 x);

half sign(half x);
half1 sign(half1 x);
half2 sign(half2 x);
half3 sign(half3 x);
half4 sign(half4 x);

fixed sign(fixed x);
fixed1 sign(fixed1 x);
fixed2 sign(fixed2 x);
fixed3 sign(fixed3 x);
fixed4 sign(fixed4 x);

PARAMETERS

x

Vector or scalar to determine its sign.

DESCRIPTION

Returns positive one, zero, or negative one for each of the components of x based on the component’s sign.

1. Returns -1 component if the respective component of x is negative.
2. Returns 0 component if the respective component of x is zero.
3. Returns 1 component if the respective component of x is positive.
4. Ideally, NaN returns NaN.

REFERENCE IMPLEMENTATION

sign for float3 could be implemented like this.

float3 sign(float x)
{
    float3 val = a > 0;
    return val - (a < 0);
}

PROFILE SUPPORT

sign is supported in all profiles except fp20.
sign is very efficient in the gp4vp, gp4gp, gp4fp, vp40, and vp30 profiles that support the native SSG instruction.

SEE ALSO

max, min, saturate, step

9.62 sincos

NAME

sincos - returns sine of scalars and vectors.

SYNOPSIS

void sincos(float a, out float s, out float c);
void sincos(float1 a, out float1 s, out float1 c);
void sincos(float2 a, out float2 s, out float2 c);
void sincos(float3 a, out float3 s, out float3 c);
void sincos(float4 a, out float4 s, out float4 c);
void sincos(half a, out half s, out half c);
void sincos(half1 a, out half1 s, out half1 c);
void sincos(half2 a, out half2 s, out half2 c);
void sincos(half3 a, out half3 s, out half3 c);
void sincos(half4 a, out half4 s, out half4 c);
void sincos(fixed a, out fixed s, out fixed c);
void sincos(fixed1 a, out fixed1 s, out fixed1 c);
void sincos(fixed2 a, out fixed2 s, out fixed2 c);
void sincos(fixed3 a, out fixed3 s, out fixed3 c);
void sincos(fixed4 a, out fixed4 s, out fixed4 c);

PARAMETERS

a

Input vector or scalar of which to determine the sine and cosine.

s

Output vector or scalar for sine results.

c

Output vector or scalar for cosine results.

DESCRIPTION

Outputs to s the sine of a in radians, and outputs to c the cosine of a in radians. The output values are in the range [-1,+1].

For vectors, the output vectors contains the sine or cosine respectively of each element of the input vector.

REFERENCE IMPLEMENTATION

sin is best implemented as a native sine instruction, however sin for a float scalar could be implemented by an approximation like this.
void sincos(float3 a, out float3 s, float3 out c)
{
    int i;

    for (i=0; i<3; i++) {
        s[i] = sin(a[i]);
        c[i] = cos(a[i]);
    }
}

PROFILE SUPPORT

sincos is fully supported in all profiles unless otherwise specified.

sincos is supported via an approximation (shown above) in the vs_1_1, vp20, and arbvp1 profiles.

sincos is unsupported in the fp20, ps_1_1, ps_1_2, and ps_1_3 profiles.

SEE ALSO

cos, sin

9.63 sinh

NAME

sinh - returns hyperbolic sine of scalars and vectors.

SYNOPSIS

float sinh(float a);
float1 sinh(float1 a);
float2 sinh(float2 a);
float3 sinh(float3 a);
float4 sinh(float4 a);

half sinh(half a);
half1 sinh(half1 a);
half2 sinh(half2 a);
half3 sinh(half3 a);
half4 sinh(half4 a);

fixed sinh(fixed a);
fixed1 sinh(fixed1 a);
fixed2 sinh(fixed2 a);
fixed3 sinh(fixed3 a);
fixed4 sinh(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the hyperbolic sine.

DESCRIPTION

Returns the hyperbolic sine of a.

For vectors, the returned vector contains the hyperbolic sine of each element of the input vector.
REFERENCE IMPLEMENTATION

\( \sinh \) for a scalar \( \text{float} \) could be implemented like this.

```c
float sinh(float x)
{
    return 0.5 * (exp(x) - exp(-x));
}
```

PROFILE SUPPORT

\( \sinh \) is supported in all profiles except \( fp20 \).

SEE ALSO

\( \text{acos, cos, cosh, exp, tanh} \)

9.64 \text{ sin}

NAME

\text{sin} - returns sine of scalars and vectors.

SYNOPSIS

```c
float sin(float a);
float1 sin(float1 a);
float2 sin(float2 a);
float3 sin(float3 a);
float4 sin(float4 a);

half sin(half a);
half1 sin(half1 a);
half2 sin(half2 a);
half3 sin(half3 a);
half4 sin(half4 a);

fixed sin(fixed a);
fixed1 sin(fixed1 a);
fixed2 sin(fixed2 a);
fixed3 sin(fixed3 a);
fixed4 sin(fixed4 a);
```

PARAMETERS

\( a \)

Vector or scalar of which to determine the sine.

DESCRIPTION

Returns the sine of \( a \) in radians. The return value is in the range \([-1, +1]\).

For vectors, the returned vector contains the sine of each element of the input vector.

REFERENCE IMPLEMENTATION

\text{sin} is best implemented as a native sine instruction, however \text{sin} for a \text{float} scalar could be implemented by an approximation like this.
float sin(float a)
{
    /* C simulation gives a max absolute error of less than 1.8e-7 */
    float4 c0 = float4( 0.0, 0.5,
                       1.0, 0.0 );
    float4 c1 = float4( 0.25, -9.0,
                       0.75, 0.159154943091 );
    float4 c2 = float4( 24.9808039603, -24.9808039603,
                       -60.1458091736, 60.1458091736 );
    float4 c3 = float4( 85.4537887573, -85.4537887573,
                       -64.9393539429, 64.9393539429 );
    float4 c4 = float4( 19.7392082214, -19.7392082214,
                       -1.0, 1.0 );
    /* r0.x = sin(a) */
    float3 r0, r1, r2;
    r1.x = c1.w * a - c1.x; // only difference from cos!
    r1.y = frac( r1.x ); // and extract fraction
    r2.x = (float) ( r1.y < c1.x ); // range check: 0.0 to 0.25
    r2.yz = (float2) ( r1.yy >= c1.yz ); // range check: 0.75 to 1.0
    r2.y = dot( r2, c4.zwz ); // range check: 0.25 to 0.75
    r0 = c0.xyz - r1.yyy; // range centering
    r0 = r0 * r0;
    r1 = c2.xy * r0 + c2.zw; // start power series
    r1 = r1 * r0 + c3.xy;
    r1 = r1 * r0 + c4.xy;
    r1 = r1 * r0 + c4.zw;
    r0.x = dot( r1, -r2 ); // range extract

    return r0.x;
}

PROFILE SUPPORT

sin is fully supported in all profiles unless otherwise specified.
sin is supported via an approximation (shown above) in the vs_1_1, vp20, and arbvp1 profiles.
sin is unsupported in the fp20, ps_1_1, ps_1_2, and ps_1_3 profiles.

SEE ALSO
asin, cos, dot, frac, sincos, tan

9.65 smoothstep

NAME

smoothstep - interpolate smoothly between two input values based on a third

SYNOPSIS

float smoothstep(float a, float b, float x);
float1 smoothstep(float1 a, float1 b, float1 x);
float2 smoothstep(float2 a, float2 b, float2 x);

9.65. smoothstep

755
float3 smoothstep(float3 a, float3 b, float3 x);
float4 smoothstep(float4 a, float4 b, float4 x);

half smoothstep(half a, half b, half x);
half1 smoothstep(half1 a, half1 b, half1 x);
half2 smoothstep(half2 a, half2 b, half2 x);
half3 smoothstep(half3 a, half3 b, half3 x);
half4 smoothstep(half4 a, half4 b, half4 x);

fixed smoothstep(fixed a, fixed b, fixed x);
fixed1 smoothstep(fixed1 a, fixed1 b, fixed1 x);
fixed2 smoothstep(fixed2 a, fixed2 b, fixed2 x);
fixed3 smoothstep(fixed3 a, fixed3 b, fixed3 x);
fixed4 smoothstep(fixed4 a, fixed4 b, fixed4 x);

PARAMETERS

a
Scalar or vector minimum reference value(s).

b
Scalar or vector minimum reference value(s).

x
Scalar or vector.

DESCRIPTION
Interpolates smoothly from 0 to 1 based on x compared to a and b.

1. Returns 0 if \( x < a < b \) or \( x > a > b \)
2. Returns 1 if \( x < b < a \) or \( x > b > a \)
3. Returns a value in the range \([0,1]\) for the domain \([a,b]\).

The slope of \( \text{smoothstep}(a,b,a) \) and \( \text{smoothstep}(a,b,b) \) is zero.

For vectors, the returned vector contains the smooth interpolation of each element of the vector \( x \).

REFERENCE IMPLEMENTATION

\text{smoothstep} for \( \text{float} \) scalars could be implemented this way:

```c
float smoothstep(float a, float b, float x)
{
    float t = saturate((x - a)/(b - a));
    return t*t*(3.0 - (2.0*t));
}
```

PROFILE SUPPORT

\text{smoothstep} is supported in all profiles except \text{fp20}.

SEE ALSO

\text{clamp}, \text{saturate}, \text{step}
9.66 sqrt

NAME

sqrt - returns square root of scalars and vectors.

SYNOPSIS

float sqrt(float a);
float1 sqrt(float1 a);
float2 sqrt(float2 a);
float3 sqrt(float3 a);
float4 sqrt(float4 a);

half sqrt(half a);
half1 sqrt(half1 a);
half2 sqrt(half2 a);
half3 sqrt(half3 a);
half4 sqrt(half4 a);

fixed sqrt(fixed a);
fixed1 sqrt(fixed1 a);
fixed2 sqrt(fixed2 a);
fixed3 sqrt(fixed3 a);
fixed4 sqrt(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the square root.

DESCRIPTION

Returns the square root of \( a \).

For vectors, the returned vector contains the square root of each element of the input vector.

The square root of zero is zero.

Ideally, the square root of negative values returns NaN (Not a Number).

sqrt often implemented as the reciprocal of a reciprocal square root approximation in older profiles.

REFERENCE IMPLEMENTATION

sqrt is best implemented as a native square root instruction, however sqrt may be implemented via a rsqrt function:

```c
float3 sqrt(float3 a)
{
    return 1.0 / rsqrt(a);
}
```

PROFILE SUPPORT

sqrt is fully supported in all profiles unless otherwise specified. sqrt is unsupported in the fp20 profile.

SEE ALSO

normalize, pow, rsqrt
9.67 step

NAME

step - implement a step function returning either zero or one

SYNOPSIS

float step(float a, float x);
float1 step(float1 a, float1 x);
float2 step(float2 a, float2 x);
float3 step(float3 a, float3 x);
float4 step(float4 a, float4 x);

half step(half a, half x);
half1 step(half1 a, half1 x);
half2 step(half2 a, half2 x);
half3 step(half3 a, half3 x);
half4 step(half4 a, half4 x);

fixed step(fixed a, fixed x);
fixed1 step(fixed1 a, fixed1 x);
fixed2 step(fixed2 a, fixed2 x);
fixed3 step(fixed3 a, fixed3 x);
fixed4 step(fixed4 a, fixed4 x);

PARAMETERS

a

Scalar or vector reference value.

x

Scalar or vector.

DESCRIPTION

Implements a step function returning one for each component of x that is greater than or equal to the corresponding component in the reference vector a, and zero otherwise.

REFERENCE IMPLEMENTATION

step for float3 vectors could be implemented this way:

float3 step(float3 a, float3 x)
{
    return x >= a;
}

PROFILE SUPPORT

step is supported in all profiles.

Support in the fp20 is limited.

SEE ALSO

max, min, saturate, smoothstep
9.68 \texttt{tanh}

\textbf{NAME}

\texttt{tanh} - returns hyperbolic tangent of scalars and vectors.

\textbf{SYNOPSIS}

\begin{verbatim}
float tanh(float a);
float1 tanh(float1 a);
float2 tanh(float2 a);
float3 tanh(float3 a);
float4 tanh(float4 a);

half tanh(half a);
half1 tanh(half1 a);
half2 tanh(half2 a);
half3 tanh(half3 a);
half4 tanh(half4 a);

fixed tanh(fixed a);
fixed1 tanh(fixed1 a);
fixed2 tanh(fixed2 a);
fixed3 tanh(fixed3 a);
fixed4 tanh(fixed4 a);
\end{verbatim}

\textbf{PARAMETERS}

\begin{verbatim}
a
Vector or scalar of which to determine the hyperbolic tangent.
\end{verbatim}

\textbf{DESCRIPTION}

Returns the hyperbolic tangent of \texttt{a}.

For vectors, the returned vector contains the hyperbolic tangent of each element of the input vector.

\textbf{REFERENCE IMPLEMENTATION}

\texttt{tanh} for a scalar \texttt{float} could be implemented like this.

\begin{verbatim}
float tanh(float x)
{
    float exp2x = exp(2*x);
    return (exp2x - 1) / (exp2x + 1);
}
\end{verbatim}

\textbf{PROFILE SUPPORT}

\texttt{tanh} is supported in all profiles except \texttt{fp20}.

\textbf{SEE ALSO}

\textit{atan, atan2, cosh, exp, sinh, tan}
9.69 tan

NAME

tan - returns tangent of scalars and vectors.

SYNOPSIS

float tan(float a);
float1 tan(float1 a);
float2 tan(float2 a);
float3 tan(float3 a);
float4 tan(float4 a);

half tan(half a);
half1 tan(half1 a);
half2 tan(half2 a);
half3 tan(half3 a);
half4 tan(half4 a);

fixed tan(fixed a);
fixed1 tan(fixed1 a);
fixed2 tan(fixed2 a);
fixed3 tan(fixed3 a);
fixed4 tan(fixed4 a);

PARAMETERS

a

Vector or scalar of which to determine the tangent.

DESCRIPTION

Returns the tangent of \( a \) in radians.

For vectors, the returned vector contains the tangent of each element of the input vector.

REFERENCE IMPLEMENTATION

tan can be implemented in terms of the sin and cos functions like this:

float tan(float a) {
    float s, c;
    sincos(a, s, c);
    return s / c;
}

PROFILE SUPPORT

tan is fully supported in all profiles unless otherwise specified.

tan is supported via approximations of sin and cos functions (see the respective sin and cos manual pages for details) in the vs_1_1, vp20, and arbvp1 profiles.

tan is unsupported in the fp20, ps_1_1, ps_1_2, and ps_1_3 profiles.

SEE ALSO

atan, atan2, cos, dot, frac, sin, sincos
9.70  tex1D

NAME

tex1D - performs a texture lookup in a given 1D sampler and, in some cases, a shadow comparison. May also use pre
computed derivatives if those are provided.

SYNOPSIS

float4 tex1D(sampler1D samp, float s)
float4 tex1D(sampler1D samp, float s, int texelOff)
float4 tex1D(sampler1D samp, float2 s)
float4 tex1D(sampler1D samp, float2 s, int texelOff)

float4 tex1D(sampler1D samp, float s, float dx, float dy)
float4 tex1D(sampler1D samp, float s, float dx, float dy, int texelOff)
float4 tex1D(sampler1D samp, float2 s, float dx, float dy)
float4 tex1D(sampler1D samp, float2 s, float dx, float dy, int texelOff)

int4 tex1D(isampler1D samp, float s);
int4 tex1D(isampler1D samp, float s, int texelOff);

int4 tex1D(isampler1D samp, float s, float dx, float dy)
int4 tex1D(isampler1D samp, float s, float dx, float dy, int texelOff)

unsigned int4 tex1D(usampler1D samp, float s);
unsigned int4 tex1D(usampler1D samp, float s, int texelOff);

unsigned int4 tex1D(usampler1D samp, float s, float dx, float dy)
unsigned int4 tex1D(usampler1D samp, float s, float dx, float dy, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. If an extra coordinate compared to the texture dimensionality is
present it is used to perform a shadow comparison. The value used in the shadow comparison is always
the last component of the coordinate vector.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, may use and derivatives dx and dy, also may perform
shadow comparison and use texel offset texelOff to compute final texel.
PROFILE SUPPORT

tex1D is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in fp40 and newer profiles, variants with texel offsets are only supported in gp4 and newer profiles, variants with integer textures are also only supported in gp4 and newer profiles.

SEE ALSO

tex1Dbias, tex1Dlod, tex1Dproj

9.71 tex1Dbias

NAME
tex1Dbias - 1D texture lookup with bias and optional texel offset.

SYNOPSIS

float4 tex1Dbias(sampler1D samp, float4 s)
float4 tex1Dbias(sampler1D samp, float4 s, int texelOff)

int4 tex1Dbias(isampler1D samp, float4 s)
int4 tex1Dbias(isampler1D samp, float4 s, int texelOff)

unsigned int4 tex1Dbias(usampler1D samp, float4 s)
unsigned int4 tex1Dbias(usampler1D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s.x
Coordinates to perform the lookup.

s.w
Level of detail bias value.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with bias in sampler samp using coordinates s. Level-of-detail bias can be used to artificially blur or sharpen the result of texture sampling.

PROFILE SUPPORT

tex1Dbias is supported in fragment profiles starting with fp30 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

tex1D, tex1Dlod, tex1Dcmpbias
9.72  tex1Dcmpbias

NAME
tex1Dcmpbias - performs a texture lookup with bias and shadow compare in a given sampler.

SYNOPSIS

float4 tex1Dcmpbias(sampler1D samp, float4 s)
float4 tex1Dcmpbias(sampler1D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the shadow comparison should be passed right after
the normal coordinates. The bias value should be passed as the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with shadow compare and bias in sampler samp using coordinates s.

PROFILE SUPPORT
tex1Dcmpbias is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants
with texelOff are only supported in gp4 and newer profiles.

SEE ALSO
tex1Dcmplod, tex1Dbias

9.73  tex1Dcmplod

NAME
tex1Dcmplod - performs a texture lookup with a specified level of detail and a shadow compare in a given sampler.

SYNOPSIS

float4 tex1Dcmplod(sampler1D samp, float4 s)
float4 tex1Dcmplod(sampler1D samp, float4 s, int texelOff)

int4 tex1Dcmplod(isampler1D samp, float4 s)
int4 tex1Dcmplod(isampler1D samp, float4 s, int texelOff)

unsigned int4 tex1Dcmplod(usampler1D samp, float4 s)
unsigned int4 tex1Dcmplod(usampler1D samp, float4 s, int texelOff)
PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the shadow comparison should be passed right after the normal coordinates. The level of detail corresponds to the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with shadow compare and a specified level of detail in sampler samp using coordinates s.

PROFILE SUPPORT

tex1Dcmplod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

tex1Dlod, tex1Dcmpbias

9.74 tex1Dfetch

NAME

tex1Dfetch - performs an unfiltered texture lookup in a given sampler.

SYNOPSIS

float4 tex1Dfetch(sampler1D samp, int4 s)
float4 tex1Dfetch(sampler1D samp, int4 s, int texelOff)

int4 tex1Dfetch(isampler1D samp, int4 s)
int4 tex1Dfetch(isampler1D samp, int4 s, int texelOff)

unsigned int4 tex1Dfetch(usampler1D samp, int4 s)
unsigned int4 tex1Dfetch(usampler1D samp, int4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The level of detail is stored in the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.
DESCRIPTION
Performs an unfiltered texture lookup in sampler samp using coordinates s. The level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

PROFILE SUPPORT
tex1Dfetch is only supported in gp4 and newer profiles.

SEE ALSO
tex1D, tex1DARRAYfetch

9.75 tex1Dlod

NAME
tex1Dlod - 1D texture lookup with specified level of detail and optional texel offset.

SYNOPSIS
float4 tex1Dlod(sampler1D samp, float4 s)
float4 tex1Dlod(sampler1D samp, float4 s, int texelOff)

int4 tex1Dlod(isampler1D samp, float4 s)
int4 tex1Dlod(isampler1D samp, float4 s, int texelOff)

unsigned int4 tex1Dlod(usampler1D samp, float4 s)
unsigned int4 tex1Dlod(usampler1D samp, float4 s, int texelOff)

PARAMETERS
samp
Sampler to lookup.
s.x
Coordinates to perform the lookup.
s.w
Level of detail.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION
Performs a texture lookup with a specified level of detail in sampler samp using coordinates s.

PROFILE SUPPORT
tex1Dlod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO
tex1D, tex1Dbias, tex1Dcmplod
9.76 tex1Dproj

NAME
tex1Dproj - performs a texture lookup with projection in a given sampler. May perform a shadow comparison if argument for shadow comparison is provided.

SYNOPSIS

float4 tex1Dproj(sampler1D samp, float2 s)
float4 tex1Dproj(sampler1D samp, float2 s, int texelOff)
float4 tex1Dproj(sampler1D samp, float3 s)
float4 tex1Dproj(sampler1D samp, float3 s, int texelOff)

int4 tex1Dproj(isampler1D samp, float2 s)
int4 tex1Dproj(isampler1D samp, float2 s, int texelOff)

unsigned int4 tex1Dproj(usampler1D samp, float2 s)
unsigned int4 tex1Dproj(usampler1D samp, float2 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the projection should be passed as the last component of the coordinate vector. The value used in the shadow comparison, if present, should be passed as the next-to-last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the coordinates used in the lookup are first projected, that is, divided by the last component of the coordinate vector and then used in the lookup. If an extra coordinate is present it is used to perform a shadow comparison, the value used in the shadow comparison is always the next-to-last component in the coordinate vector.

PROFILE SUPPORT

tex1Dproj is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in fp40 and newer profiles, variants with texel offsets are only supported in gp4 and newer profiles.

SEE ALSO

tex1D, tex1DARRAYproj

9.77 tex1Dsize

NAME
tex1Dsize - returns the size of a given texture image for a given level of detail.
SYNOPSIS

int3 tex1Dsize(sampler1D samp, int lod)
int3 tex1Dsize(isampler1D samp, int lod)
int3 tex1Dsize(usampler1D samp, int lod)

PARAMETERS

samp
Sampler to be queried for size.

lod
Level of detail to obtain size.

DESCRIPTION

Given a sampler and a level of detail the size of the corresponding texture image is returned as the result of the operation.

PROFILE SUPPORT

tex1Dsize is only supported in gp4 and newer profiles.

SEE ALSO

tex1D, tex1DARRAYsize

9.78 tex1DARRAYbias

NAME

tex1DARRAYbias - performs a texture lookup with bias in a given sampler array.

SYNOPSIS

float4 tex1DARRAYbias(sampler1DARRAY samp, float4 s)
float4 tex1DARRAYbias(sampler1DARRAY samp, float4 s, int texelOff)

int4 tex1DARRAYbias(isampler1DARRAY samp, float4 s)
int4 tex1DARRAYbias(isampler1DARRAY samp, float4 s, int texelOff)

unsigned int4 tex1DARRAYbias(usampler1DARRAY samp, float4 s)
unsigned int4 tex1DARRAYbias(usampler1DARRAY samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.

s
Coordinates to perform the lookup. The value used to select the layer should be passed in the vector component right after the regular coordinates. The bias value should be passed as the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.
DESCRIPTION
Performs a texture lookup with bias in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates.

PROFILE SUPPORT
tex1DARRAYbias is only supported in gp4 and newer profiles.

SEE ALSO
tex1DARRAY, tex1DARRAYlod

9.79 tex1DARRAYcmpbias

NAME
tex1DARRAYcmpbias - performs a texture lookup with shadow compare and bias in a given sampler array.

SYNOPSIS
float4 tex1DARRAYcmpbias(sampler1DARRAY samp, float4 s)
float4 tex1DARRAYcmpbias(sampler1DARRAY samp, float4 s, int texelOff)

PARAMETERS
samp
Sampler array to lookup.
s
Coordinates to perform the lookup. The value used to select the layer is the second coordinate, the third is the value used in the shadow comparison, the fourth corresponds to the bias value.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION
Performs a texture lookup with bias in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates, the lookup involves a shadow comparison and may use texel offset texelOff to compute the final texel.

PROFILE SUPPORT
tex1DARRAYcmpbias is only supported in gp4 and newer profiles.

SEE ALSO
tex1DARRAYbias, tex1DARRAYlod, tex1DARRAYcmplod

9.80 tex1DARRAYcmplod

NAME
tex1DARRAYcmplod - performs a texture lookup with shadow compare and a level of detail in a given sampler array.
SYNOPSIS

float4 tex1DARRAYcmplod(sampler1DARRAY samp, float4 s)
float4 tex1DARRAYcmplod(sampler1DARRAY samp, float4 s, int texelOff)

samp
Sampler array to look up.

s
Coordinates to perform the lookup. The value used to select the layer is the second coordinate, the third is the value used in the shadow comparison, the fourth corresponds to the level of detail.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with level of detail in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates, the lookup involves a shadow comparison and may use texel offset texelOff to compute the final texel.

PROFILE SUPPORT

tex1DARRAYcmplod is only supported in gp4 and newer profiles.

SEE ALSO
tex1DARRAYlod, tex1DARRAYbias, tex1DARRAYcmpbias

9.81 tex1DARRAYfetch

NAME
tex1DARRAYfetch - performs an unfiltered texture lookup in a given sampler array.

SYNOPSIS

float4 tex1DARRAYfetch(sampler1DARRAY samp, int4 s)
float4 tex1DARRAYfetch(sampler1DARRAY samp, int4 s, int texelOff)

int4 tex1DARRAYfetch(isampler1DARRAY samp, int4 s)
int4 tex1DARRAYfetch(isampler1DARRAY samp, int4 s, int texelOff)

unsigned int4 tex1DARRAYfetch(usampler1DARRAY samp, int4 s)
unsigned int4 tex1DARRAYfetch(usampler1DARRAY samp, int4 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.

s
Coordinates to perform the lookup, the layer is selected by the component right after the regular coordinates, the level of detail is provided by the last component of the coordinate vector.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION
Performs an unfiltered texture lookup in sampler array samp using coordinates s. The layer to be accessed is selected by the component right after the regular coordinates, the level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

PROFILE SUPPORT
tex1DARRAYfetch is only supported in gp4 and newer profiles.

SEE ALSO
tex1Dfetch

9.82 tex1DARRAYlod

NAME
tex1DARRAYlod - 1D texture array lookup with specified level of detail and optional texel offset.

SYNOPSIS
float4 tex1DARRAYlod(sampler1DARRAY samp, float4 s)
float4 tex1DARRAYlod(sampler1DARRAY samp, float4 s, int texelOff)

int4 tex1DARRAYlod(isampler1DARRAY samp, float4 s)
int4 tex1DARRAYlod(isampler1DARRAY samp, float4 s, int texelOff)

unsigned int4 tex1DARRAYlod(usampler1DARRAY samp, float4 s)
unsigned int4 tex1DARRAYlod(usampler1DARRAY samp, float4 s, int texelOff)

PARAMETERS
  samp
  Sampler array to lookup.
  s.x
  Coordinates to perform the lookup.
  s.y
  Texture array layer.
  s.w
  Level of detail.
  texelOff
  Offset to be added to obtain the final texel.

DESCRIPTION
Performs a texture lookup with a specified level of detail in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates.
PROFILE SUPPORT

tex1DARRAYlod is only supported in gp4 and newer profiles.

SEE ALSO

tex1DARRAY, tex1DARRAYbias

9.83 tex1DARRAY

NAME

tex1DARRAY - performs a texture lookup in a given sampler array may use pre computed derivatives and, in some cases, perform a shadow comparison.

SYNOPSIS

float4 tex1DARRAY(sampler1DARRAY samp, float2 s)
float4 tex1DARRAY(sampler1DARRAY samp, float2 s, int texelOff)
float4 tex1DARRAY(sampler1DARRAY samp, float3 s)
float4 tex1DARRAY(sampler1DARRAY samp, float3 s, int texelOff)

float4 tex1DARRAY(sampler1DARRAY samp, float2 s, float dx, float dy)
float4 tex1DARRAY(sampler1DARRAY samp, float2 s, float dx, float dy, int texelOff)
float4 tex1DARRAY(sampler1DARRAY samp, float3 s, float dx, float dy)
float4 tex1DARRAY(sampler1DARRAY samp, float3 s, float dx, float dy, int texelOff)

int4 tex1DARRAY(isampler1DARRAY samp, float2 s)
int4 tex1DARRAY(isampler1DARRAY samp, float2 s, int texelOff)
int4 tex1DARRAY(isampler1DARRAY samp, float2 s, float dx, float dy)
int4 tex1DARRAY(isampler1DARRAY samp, float2 s, float dx, float dy, int texelOff)

unsigned int4 tex1DARRAY(usampler1DARRAY samp, float2 s)
unsigned int4 tex1DARRAY(usampler1DARRAY samp, float2 s, int texelOff)
unsigned int4 tex1DARRAY(usampler1DARRAY samp, float2 s, float dx, float dy)
unsigned int4 tex1DARRAY(usampler1DARRAY samp, float2 s, float dx, float dy, int texelOff)

PARAMETERS

samp
Sampler array to look up.

s
Coordinates to perform the lookup. The value used to select the layer is passed immediately after the regular coordinates, if an extra coordinate is present it is used to perform a shadow comparison.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates. Also may use the derivatives dx and dy, the lookup may involve a shadow comparison and use texel offset texelOff to compute the final texel.

PROFILE SUPPORT

tex1DARRAY is only supported in gp4 and newer profiles.

SEE ALSO

tex1DARRAYbias, tex1DARRAYlod

9.84 tex1DARRAYproj

NAME

tex1DARRAYproj - performs a texture lookup with projection in a given sampler array. May perform a shadow comparison if argument for shadow comparison is provided.

SYNOPSIS

float4 tex1DARRAYproj(sampler1DARRAY samp, float3 s)
float4 tex1DARRAYproj(sampler1DARRAY samp, float3 s, int texelOff)
float4 tex1DARRAYproj(sampler1DARRAY samp, float4 s)
float4 tex1DARRAYproj(sampler1DARRAY samp, float4 s, int texelOff)

int4 tex1DARRAYproj(isampler1DARRAY samp, float3 s)
int4 tex1DARRAYproj(isampler1DARRAY samp, float3 s, int texelOff)

unsigned int4 tex1DARRAYproj(usampler1DARRAY samp, float3 s)
unsigned int4 tex1DARRAYproj(usampler1DARRAY samp, float3 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.

s
Coordinates to perform the lookup. The value used to select the layer should be passed as the component right after the lookup coordinates. The value used in the projection should be passed as the last component of the coordinate vector. The value used in the shadow comparison, if present, should be passed as the next-to-last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.
DESCRIPTION
Performs a texture lookup in sampler array samp using coordinates s, the layer used in the lookup is first selected using the coordinate component right after the regular coordinates. The coordinates used in the lookup are then projected, that is, divided by the last component of the coordinate vector and then used in the lookup. If an extra coordinate is present it is used to perform a shadow comparison, the value used in the shadow comparison is always the next-to-last component in the coordinate vector.

PROFILE SUPPORT
tex1DARRAYproj is only supported in gp4 and newer profiles.

SEE ALSO
tex1D, tex1Dproj

9.85 tex1DARRAYsize

NAME
tex1DARRAYsize - returns the size of a given texture array image for a given level of detail.

SYNOPSIS
int3 tex1DARRAYsize(sampler1DARRAY samp, int lod)
int3 tex1DARRAYsize(isampler1DARRAY samp, int lod)
int3 tex1DARRAYsize(usampler1DARRAY samp, int lod)

PARAMETERS
samp
Sampler to be queried for size.
lod
Level of detail to obtain size.

DESCRIPTION
Given a sampler array and a level of detail the size of one element of the corresponding texture array for a given level of detail is returned as a result of the operation.

PROFILE SUPPORT
tex1DARRAYsize is only supported in gp4 and newer profiles.

SEE ALSO
tex1Dsize

9.86 tex2Dbias

NAME
tex2Dbias - 2D texture lookup with bias and optional texel offset.
SYNOPSIS

float4 tex2Dbias(sampler2D samp, float4 s)
float4 tex2Dbias(sampler2D samp, float4 s, int texelOff)

int4 tex2Dbias(isampler2D samp, float4 s)
int4 tex2Dbias(isampler2D samp, float4 s, int texelOff)

unsigned int4 tex2Dbias(usampler2D samp, float4 s)
unsigned int4 tex2Dbias(usampler2D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s.xy
Coordinates to perform the lookup.

s.w
Level of detail bias value.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with bias in sampler samp using coordinates s. Level-of-detail bias can be used to artificially blur or sharpen the result of texture sampling.

PROFILE SUPPORT

tex2Dbias is supported in fragment profiles starting with fp30 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

tex2D, tex2Dlod, tex2Dcmpbias

9.87 tex2D

NAME

tex2D - performs a texture lookup in a given 2D sampler and, in some cases, a shadow comparison. May also use pre computed derivatives if those are provided.

SYNOPSIS

float4 tex2D(sampler2D samp, float2 s)
float4 tex2D(sampler2D samp, float2 s, int texelOff)
float4 tex2D(sampler2D samp, float3 s)
float4 tex2D(sampler2D samp, float3 s, int texelOff)

float4 tex2D(sampler2D samp, float2 s, float2 dx, float2 dy)
float4 tex2D(sampler2D samp, float2 s, float2 dx, float2 dy, int texelOff)
float4 tex2D(sampler2D samp, float3 s, float2 dx, float2 dy)
float4 tex2D(sampler2D samp, float3 s, float2 dx, float2 dy, int texelOff)

int4 tex2D(isampler2D samp, float2 s)
int4 tex2D(isampler2D samp, float2 s, int texelOff)
int4 tex2D(isampler2D samp, float2 s, float2 dx, float2 dy)
int4 tex2D(isampler2D samp, float2 s, float2 dx, float2 dy, int texelOff)

unsigned int4 tex2D(usampler2D samp, float2 s)
unsigned int4 tex2D(usampler2D samp, float2 s, int texelOff)
unsigned int4 tex2D(usampler2D samp, float2 s, float2 dx, float2 dy)
unsigned int4 tex2D(usampler2D samp, float2 s, float2 dx, float2 dy, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. If an extra coordinate compared to the texture dimensionality is present it is used to perform a shadow comparison. The value used in the shadow comparison is always the last component of the coordinate vector.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, may use and derivatives dx and dy, also may perform shadow comparison and use texel offset texelOff to compute final texel.

PROFILE SUPPORT

tex2D is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in fp40 and newer profiles, variants with texel offsets are only supported in gp4 and newer profiles. Variants with integer textures are also only supported in gp4 and newer profiles.

SEE ALSO
tex2Dbias, tex2Dlod, tex2Dproj

9.88 tex2Dcmpbias

NAME

tex2Dcmpbias - performs a texture lookup with bias and shadow compare in a given sampler.
SYNOPSIS

float4 tex2Dcmpbias(sampler2D samp, float4 s)
float4 tex2Dcmpbias(sampler2D samp, float4 s, int2 texelOff)

PARAMETERS

samo
Sampler to lookup.

s.xy
Coordinates to perform the lookup.

s.z
The value used in the shadow comparison.

s.w
The bias value.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with shadow compare and bias in sampler samp using coordinates s.

PROFILE SUPPORT

tex2Dcmpbias is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles.

SEE ALSO

tex2Dcmplod, tex2Dbias

9.89 tex2Dcmplod

NAME

tex2Dcmplod - performs a texture lookup with a specified level of detail and a shadow compare in a given sampler.

SYNOPSIS

float4 tex2Dcmplod(sampler2D samp, float4 s)
float4 tex2Dcmplod(sampler2D samp, float4 s, int texelOff)

PARAMETERS

samo
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the shadow comparison should be passed right after the normal coordinates. The level of detail corresponds to the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

**DESCRIPTION**

Performs a texture lookup with shadow compare and a specified level of detail in sampler samp using coordinates s.

**PROFILE SUPPORT**

tex2Dcmplod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles.

**SEE ALSO**
tex2Dlod, tex2Dcmpbias

### 9.90 tex2Dfetch

**NAME**

tex2Dfetch - performs an unfiltered texture lookup in a given sampler.

**SYNOPSIS**

```c
float4 tex2Dfetch(sampler2D samp, int4 s)
float4 tex2Dfetch(sampler2D samp, int4 s, int texelOff)

int4 tex2Dfetch(isampler2D samp, int4 s)
int4 tex2Dfetch(isampler2D samp, int4 s, int texelOff)

unsigned int4 tex2Dfetch(usampler2D samp, int4 s)
unsigned int4 tex2Dfetch(usampler2D samp, int4 s, int texelOff)
```

**PARAMETERS**

samp
  Sampler to lookup.

s
  Coordinates to perform the lookup. The level of detail is stored in the last component of the coordinate vector.

texelOff
  Offset to be added to obtain the final texel.

**DESCRIPTION**

Performs an unfiltered texture lookup in sampler samp using coordinates s. The level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

**PROFILE SUPPORT**

tex2Dfetch is only supported in gp4 and newer profiles.

**SEE ALSO**
tex2D, tex2DARRAYfetch
9.91 tex2Dlod

NAME
tex2Dlod - 2D texture lookup with specified level of detail and optional texel offset.

SYNOPSIS

float4 tex2Dlod(sampler2D samp, float4 s)
float4 tex2Dlod(sampler2D samp, float4 s, int texelOff)

int4 tex2Dlod(isampler2D samp, float4 s)
int4 tex2Dlod(isampler2D samp, float4 s, int texelOff)

unsigned int4 tex2Dlod(usampler2D samp, float4 s)
unsigned int4 tex2Dlod(usampler2D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s.xy
Coordinates to perform the lookup.

s.w
Level of detail.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with a specified level of detail in sampler samp using coordinates s.

PROFILE SUPPORT

tex2Dlod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with
texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4
and newer profiles.

SEE ALSO

tex2D, tex2Dbias, tex2Dcmplod

9.92 tex2Dproj

NAME
tex2Dproj - performs a texture lookup with projection in a given sampler. May perform a shadow comparison if
argument for shadow comparison is provided.
SYNOPSIS

float4 tex2Dproj(sampler2D samp, float3 s)
float4 tex2Dproj(sampler2D samp, float3 s, int texelOff)
float4 tex2Dproj(sampler2D samp, float4 s)
float4 tex2Dproj(sampler2D samp, float4 s, int texelOff)

int4 tex2Dproj(isampler2D samp, float3 s)
int4 tex2Dproj(isampler2D samp, float3 s, int texelOff)

unsinged int4 tex2Dproj(usampler2D samp, float3 s)
unsinged int4 tex2Dproj(usampler2D samp, float3 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the projection should be passed as the last component of the coordinate vector. The value used in the shadow comparison, if present, should be passed as the next-to-last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the coordinates used in the lookup are first projected, that is, divided by the last component of the coordinate vector and then used in the lookup. If an extra coordinate is present it is used to perform a shadow comparison, the value used in the shadow comparison is always the next-to-last component in the coordinate vector.

PROFILE SUPPORT

tex2Dproj is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in fp40 and newer profiles, variants with texel offsets are only supported in gp4 and newer profiles.

SEE ALSO

tex2D, tex2DARRAYproj

9.93 tex2Dsize

NAME

tex2Dsize - returns the size of a given texture image for a given level of detail.

SYNOPSIS

int3 tex2Dsize(sampler2D samp, int lod)
int3 tex2Dsize(isampler2D samp, int lod)
int3 tex2Dsize(usampler2D samp, int lod)
PARAMETERS

samp
Sampler to be queried for size.

lod
Level of detail to obtain size.

DESCRIPTION

Given a sampler and a level of detail the size of the corresponding texture image is returned as the result of the operation.

PROFILE SUPPORT

tex2Dsize is only supported in gp4 and newer profiles.

SEE ALSO
tex2D, tex2DARRAYsize

9.94 tex2DARRAYbias

NAME

tex2DARRAYbias - performs a texture lookup with bias in a given sampler array.

SYNOPSIS

float4 tex2DARRAYbias(sampler2DARRAY samp, float4 s)
float4 tex2DARRAYbias(sampler2DARRAY samp, float4 s, int texelOff)

int4 tex2DARRAYbias(isampler2DARRAY samp, float4 s)
int4 tex2DARRAYbias(isampler2DARRAY samp, float4 s, int texelOff)

unsigned int4 tex2DARRAYbias(usampler2DARRAY samp, float4 s)
unsigned int4 tex2DARRAYbias(usampler2DARRAY samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.

s
Coordinates to perform the lookup. The value used to select the layer should be passed in the vector component right after the regular coordinates. The bias value should be passed as the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with bias in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates.
PROFILE SUPPORT

tex2DARRAYbias is only supported in gp4 and newer profiles.

SEE ALSO

tex2DARRAY, tex2DARRAYlod

9.95 tex2DARRAYfetch

NAME
tex2DARRAYfetch - performs an unfiltered texture lookup in a given sampler array.

SYNOPSIS

float4 tex2DARRAYfetch(sampler2DARRAY samp, int4 s)
float4 tex2DARRAYfetch(sampler2DARRAY samp, int4 s, int texelOff)

int4 tex2DARRAYfetch(isampler2DARRAY samp, int4 s)
int4 tex2DARRAYfetch(isampler2DARRAY samp, int4 s, int texelOff)

unsigned int4 tex2DARRAYfetch(usampler2DARRAY samp, int4 s)
unsigned int4 tex2DARRAYfetch(usampler2DARRAY samp, int4 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.

s
Coordinates to perform the lookup, the layer is selected by the component right after the regular coordinates, the level of detail is provided by the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs an unfiltered texture lookup in sampler array samp using coordinates s. The layer to be accessed is selected by the component right after the regular coordinates, the level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

PROFILE SUPPORT

tex2DARRAYfetch is only supported in gp4 and newer profiles.

SEE ALSO

tex2Dfetch
9.96 tex2DARRAYlod

NAME
tex2DARRAYlod - 2D texture array lookup with specified level of detail.

SYNOPSIS

float4 tex2DARRAYlod(sampler2DARRAY samp, float4 s)
float4 tex2DARRAYlod(sampler2DARRAY samp, float4 s, int texelOff)

int4 tex2DARRAYlod(isampler2DARRAY samp, float4 s)
int4 tex2DARRAYlod(isampler2DARRAY samp, float4 s, int texelOff)

unsigned int4 tex2DARRAYlod(usampler2DARRAY samp, float4 s)
unsigned int4 tex2DARRAYlod(usampler2DARRAY samp, float4 s, int texelOff)

PARAMETERS

samp
  Sampler array to lookup.

s.xy
  Coordinates to perform the lookup.

s.z
  Texture array layer.

s.w
  Level of detail.

texelOff
  Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with a specified level of detail in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates.

PROFILE SUPPORT

tex2DARRAYlod is only supported in gp4 and newer profiles.

SEE ALSO
tex2DARRAY, tex2DARRAYbias

9.97 tex2DARRAY

NAME
tex2DARRAY - performs a texture lookup in a given sampler array may use pre computed derivatives and, in some cases, perform a shadow comparison.
SYNOPSIS

float4 tex2DARRAY(sampler2DARRAY samp, float3 s)
float4 tex2DARRAY(sampler2DARRAY samp, float3 s, int texelOff)
float4 tex2DARRAY(sampler2DARRAY samp, float4 s)
float4 tex2DARRAY(sampler2DARRAY samp, float4 s, int texelOff)
float4 tex2DARRAY(sampler2DARRAY samp, float3 s, float dx, float dy)
float4 tex2DARRAY(sampler2DARRAY samp, float3 s, float dx, float dy, int texelOff)
float4 tex2DARRAY(sampler2DARRAY samp, float4 s, float dx, float dy)
float4 tex2DARRAY(sampler2DARRAY samp, float4 s, float dx, float dy, int texelOff)
int4 tex2DARRAY(isampler2DARRAY samp, float3 s)
int4 tex2DARRAY(isampler2DARRAY samp, float3 s, int texelOff)
int4 tex2DARRAY(isampler2DARRAY samp, float3 s, float dx, float dy)
int4 tex2DARRAY(isampler2DARRAY samp, float3 s, float dx, float dy, int texelOff)
unsigned int4 tex2DARRAY(usampler2DARRAY samp, float3 s)
unsigned int4 tex2DARRAY(usampler2DARRAY samp, float3 s, int texelOff)
unsigned int4 tex2DARRAY(usampler2DARRAY samp, float3 s, float dx, float dy)
unsigned int4 tex2DARRAY(usampler2DARRAY samp, float3 s, float dx, float dy, int texelOff)

PARAMETERS

samp
Sampler array to look up.
s
Coordinates to perform the lookup. The value used to select the layer is passed immediately after the regular coordinates, if an extra coordinate is present it is used to perform a shadow comparison.
dx
Pre computed derivative along the x axis.
dy
Pre computed derivative along the y axis.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates. Also may use the derivatives dx and dy, the lookup may involve a shadow comparison and use texel offset texelOff to compute the final texel.

PROFILE SUPPORT

tex2DARRAY is only supported in gp4 and newer profiles.
SEE ALSO

tex2DARRAYbias, tex2DARRAYlod

9.98 tex2DARRAYproj

NAME
tex2DARRAYproj - performs a texture lookup with projection in a given sampler array.

SYNOPSIS

float4 tex2DARRAYproj(sampler2DARRAY samp, float4 s)
float4 tex2DARRAYproj(sampler2DARRAY samp, float4 s, int texelOff)

int4 tex2DARRAYproj(isampler2DARRAY samp, float4 s)
int4 tex2DARRAYproj(isampler2DARRAY samp, float4 s, int texelOff)

unsigned int4 tex2DARRAYproj(usampler2DARRAY samp, float4 s)
unsigned int4 tex2DARRAYproj(usampler2DARRAY samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler array to lookup.
s.xy
Coordinates to perform the lookup.
s.z
The layer within the array.
s.w
The value to use for the projection.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler array samp using coordinates s.xy, on the selected layer s.z using the given projection s.w. The coordinates are projected, that is, divided by the projection value before being used in the lookup.

PROFILE SUPPORT

tex2DARRAYproj is only supported in gp4 and newer profiles.

SEE ALSO

tex2D, tex2Dproj
9.99 tex2DARRAYsize

NAME

tex2DARRAYsize - returns the size of a given texture array image for a given level of detail.

SYNOPSIS

int3 tex2DARRAYsize(sampler2DARRAY samp, int lod)
int3 tex2DARRAYsize(isampler2DARRAY samp, int lod)
int3 tex2DARRAYsize(usampler2DARRAY samp, int lod)

PARAMETERS

samp
Sampler to be queried for size.
lod
Level of detail to obtain size.

DESCRIPTION

Given a sampler array and a level of detail the size of one element of the corresponding texture array for a given level of detail is returned as a result of the operation.

PROFILE SUPPORT

tex2DARRAYsize is only supported in gp4 and newer profiles.

SEE ALSO

tex2Dsize

9.100 tex2DMSfetch

NAME

tex2DMSfetch - performs an unfiltered texture lookup in a given multisample texture sampler.

SYNOPSIS

float4 tex2DMSfetch(sampler2DMS samp, int2 s, int n)
int4 tex2DMSfetch(isampler2DMS samp, int2 s, int n)
unsigned int4 tex2DMSfetch(usampler2DMS samp, int2 s, int n)

PARAMETERS

samp
Sampler to lookup.
s
2D coordinates to perform the lookup.
num
Multisample sample number.

**DESCRIPTION**

Performs an unfiltered texture lookup of sample *num* of sampler *samp* using coordinates *s*.

**PROFILE SUPPORT**

*tex2DMSfetch* is only supported in *gp5* and newer profiles.

*glsl* profile support requires *ARB_texture_multisample* extension or OpenGL 3.2

**SEE ALSO**

GL_ARB_texture_multisample
tex2Dfetch, tex2DARRAYfetch, tex2DMSsize

---

### 9.101 tex2DMSsize

**NAME**

tex2DMSsize - returns the size of a given multisample texture image for a given level of detail.

**SYNOPSIS**

```c
int3 tex2DMSsize(sampler2DMS samp)
int3 tex2DMSsize(isampler2DMS samp)
int3 tex2DMSsize(usampler2DMS samp)
```

**PARAMETERS**

- **samp**
  
  Sampler to be queried for size.

**DESCRIPTION**

Given a sampler the size of the corresponding texture image is returned as the result of the operation.

**PROFILE SUPPORT**

*tex2DMSsize* is only supported in *gp5* and newer profiles.

*glsl* profile support requires *ARB_texture_multisample* extension or OpenGL 3.2

**SEE ALSO**

GL_ARB_texture_multisample
tex2Dsize, tex2DMSfetch

### 9.102 tex2DMSARRAYfetch

**NAME**

tex2DMSARRAYfetch - performs an unfiltered texture lookup in a given multisample texture sampler.
**SYNOPSIS**

```c
float4 tex2DMSARRAYfetch(sampler2DMSARRAY samp, int3 s, int n)
int4 tex2DMSARRAYfetch(isampler2DMSARRAY samp, int3 s, int n)
unsigned int4 tex2DMSARRAYfetch(usampler2DMSARRAY samp, int3 s, int n)
```

**PARAMETERS**

<table>
<thead>
<tr>
<th>samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampler to lookup.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s.xy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D coordinates to perform the lookup.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>s.z</th>
</tr>
</thead>
<tbody>
<tr>
<td>The multisample array sampling layer for lookup.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>num</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multisample sample number.</td>
</tr>
</tbody>
</table>

**DESCRIPTION**

Performs an unfiltered texture lookup of sample `num` of layer `s.z` of sampler `samp` using coordinates `s.xy`.

**PROFILE SUPPORT**

`tex2DMSARRAYfetch` is only supported in `gp5` and newer profiles.

`glsl` profile support requires the [ARB_texture_multisample](https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_texture_multisample.txt) extension or OpenGL 3.2

**SEE ALSO**

[GL_ARB_texture_multisample](https://www.khronos.org/registry/OpenGL/extensions/ARB/ARB_texture_multisample.txt)

tex2Dfetch, tex2DARRAYfetch, tex2DMSARRAYsize

---

**9.103 tex2DMSARRAYsize**

**NAME**

tex2DMSARRAYsize - returns the size of a given multisample texture image for a given level of detail.

**SYNOPSIS**

```c
int3 tex2DMSARRAYsize(sampler2DMSARRAY samp)
int3 tex2DMSARRAYsize(isampler2DMSARRAY samp)
int3 tex2DMSARRAYsize(usampler2DMSARRAY samp)
```

**PARAMETERS**

<table>
<thead>
<tr>
<th>samp</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sampler to be queried for size.</td>
</tr>
</tbody>
</table>
DESCRIPTION
Given a sampler the size of the corresponding texture image is returned as the result of the operation.

PROFILE SUPPORT
tex2DMSARRAYsize is only supported in gp5 and newer profiles.
gls profile support requires ARB_texture_multisample extension or OpenGL 3.2

SEE ALSO
GL_ARB_texture_multisample
tex2Dsize, tex2DMSARRAYfetch

9.104 tex3Dbias

NAME
tex3Dbias - 3D texture lookup with bias and optional texel offset.

SYNOPSIS

float4 tex3Dbias(sampler3D samp, float4 s)
float4 tex3Dbias(sampler3D samp, float4 s, int texelOff)

int4 tex3Dbias(isampler3D samp, float4 s)
int4 tex3Dbias(isampler3D samp, float4 s, int texelOff)

unsigned int4 tex3Dbias(usampler3D samp, float4 s)
unsigned int4 tex3Dbias(usampler3D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.
s.xyz
Coordinates to perform the lookup.
s.w
Level of detail bias value.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION
Performs a texture lookup with bias in sampler samp using coordinates s. Level-of-detail bias can be used to artificially blur or sharpen the result of texture sampling.

PROFILE SUPPORT
tex3Dbias is supported in fragment profiles starting with fp30 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.
SEE ALSO

tex3D, tex3Dlod

9.105 tex3Dfetch

NAME
tex3Dfetch - performs an unfiltered texture lookup in a given sampler.

SYNOPSIS

float4 tex3Dfetch(sampler3D samp, int4 s)
float4 tex3Dfetch(sampler3D samp, int4 s, int texelOff)
int4 tex3Dfetch(isampler3D samp, int4 s)
int4 tex3Dfetch(isampler3D samp, int4 s, int texelOff)
unsigned int4 tex3Dfetch(usampler3D samp, int4 s)
unsigned int4 tex3Dfetch(usampler3D samp, int4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The level of detail is stored in the last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs an unfiltered texture lookup in sampler samp using coordinates s. The level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

PROFILE SUPPORT
tex3Dfetch is only supported in gp4 and newer profiles.

SEE ALSO
tex3D

9.106 tex3Dlod

NAME
tex3Dlod - 3D texture lookup with specified level of detail and optional texel offset.
SYNOPSIS

float4 tex3Dlod(sampler3D samp, float4 s)
float4 tex3Dlod(sampler3D samp, float4 s, int texelOff)

int4 tex3Dlod(isampler3D samp, float4 s)
int4 tex3Dlod(isampler3D samp, float4 s, int texelOff)

unsigned int4 tex3Dlod(usampler3D samp, float4 s)
unsigned int4 tex3Dlod(usampler3D samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.
s.xyz
Coordinates to perform the lookup.
s.w
Level of detail.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with a specified level of detail in sampler samp using coordinates s.

PROFILE SUPPORT

tex3Dlod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

tex3D, tex3Dbias

9.107 tex3D

NAME

tex3D - performs a texture lookup in a given 3D sampler. May also use pre computed derivatives if those are provided.

SYNOPSIS

float4 tex3D(sampler3D samp, float3 s)
float4 tex3D(sampler3D samp, float3 s, int texelOff)

float4 tex3D(sampler3D samp, float3 s, float3 dx, float3 dy)
float4 tex3D(sampler3D samp, float3 s, float3 dx, float3 dy, int texelOff)

int4 tex3D(isampler3D samp, float3 s)
int4 tex3D(isampler3D samp, float3 s, int texelOff)
int4 tex3D(isampler3D samp, float3 s, float3 dx, float3 dy)
int4 tex3D(isampler3D samp, float3 s, float3 dx, float3 dy, int texelOff)

unsigned int4 tex3D(usampler3D samp, float3 s)
unsigned int4 tex3D(usampler3D samp, float3 s, int texelOff)

unsigned int4 tex3D(usampler3D samp, float3 s, float3 dx, float3 dy)
unsigned int4 tex3D(usampler3D samp, float3 s, float3 dx, float3 dy, int texelOff)

PARAMETERS

samp
Sampler to lookup.
s
Coordinates to perform the lookup.
dx
Pre computed derivative along the x axis.
dy
Pre computed derivative along the y axis.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, may use and derivatives dx and dy, also may use texel offset texelOff to compute final texel.

PROFILE SUPPORT

tex3D is supported in all fragment profiles and all vertex profiles starting with vp40, variants with texel offsets are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

tex3Dbias, tex3Dlod, tex3Dproj

9.108 tex3Dproj

NAME

tex3Dproj - performs a texture lookup with projection in a given sampler.

SYNOPSIS

float4 tex3Dproj(sampler3D samp, float4 s)
float4 tex3Dproj(sampler3D samp, float4 s, int texelOff)

int4 tex3Dproj(isampler3D samp, float4 s)
int4 tex3Dproj(isampler3D samp, float4 s, int texelOff)
unsigned int4 tex3Dproj(usampler3D samp, float4 s)
unsigned int4 tex3Dproj(usampler3D samp, float4 s, int texelOff)

PARAMETERS
  samp
  Sampler to lookup.
  s
  Coordinates to perform the lookup. The value used in the projection should be passed as the last component
  of the coordinate vector.
  texelOff
  Offset to be added to obtain the final texel.

DESCRIPTION
Performs a texture lookup in sampler samp using coordinates s, the coordinates used in the lookup are first projected,
that is, divided by the last component of the coordinate vector and then used in the lookup.

PROFILE SUPPORT

tex3Dproj is supported in all fragment profiles and all vertex profiles starting with vp40, variants with texel offsets are
only supported in gp4 and newer profiles.

SEE ALSO
tex3D

9.109 tex3Dsize

NAME
tex3Dsize - returns the size of a given texture image for a given level of detail.

SYNOPSIS
int3 tex3Dsize(sampler3D samp, int lod)
int3 tex3Dsize(isampler3D samp, int lod)
int3 tex3Dsize(usampler3D samp, int lod)

PARAMETERS
  samp
  Sampler to be queried for size.
  lod
  Level of detail to obtain size.

DESCRIPTION
Given a sampler and a level of detail the size of the corresponding texture image is returned as the result of the operation.
PROFILE SUPPORT

tex3Dsize is only supported in gp4 and newer profiles.

SEE ALSO

tex3D

9.110 texBUF

NAME

texBUF - performs an unfiltered texture lookup in a given texture buffer sampler.

SYNOPSIS

float4 texBUF(samplerBUF samp, int s)

int4 texBUF(isamplerBUF samp, int s)

unsigned int4 texBUF(usamplerBUF samp, int s)

PARAMETERS

samp
   Sampler to lookup.

s
   Coordinates to perform the lookup.

DESCRIPTION

Performs an unfiltered texture lookup in texture buffer sampler samp using coordinates s.

Texture buffer samplers are created with the EXT_texture_buffer_object extension. See:


Texture buffer object samplers roughly correspond to the tbuffer functionality of DirectX 10.

PROFILE SUPPORT

texBUF is supported in gp4vp, gp4gp, and gp4fp profiles.

SEE ALSO

tex1D, texBUFsize

9.111 texBUFsize

NAME

texBUFsize - returns the size of a given texture image for a given level of detail.
SYNOPSIS

int3 texBUFsize(samplerBUF samp, int lod)
int3 texBUFsize(isamplerBUF samp, int lod)
int3 texBUFsize(usamplerBUF samp, int lod)

PARAMETERS

samp
Sampler to be queried for size.
lod
Level of detail to obtain size.

DESCRIPTION

Given a sampler and a level of detail the size (width in \( x \), height in \( y \), and depth in \( z \)) of the corresponding texture buffer is returned as the result of the operation.

Because texture buffers lack mipmaps, the \( lod \) parameter is unused.

Texture buffer samplers are created with the `EXT_texture_buffer_object` extension. See:


Texture buffer object samplers roughly correspond to the `tbuffer` functionality of DirectX 10.

PROFILE SUPPORT

texBUF is supported in `gp4vp`, `gp4gp`, and `gp4fp` profiles.

SEE ALSO

tex1Dsize, texBUF

9.112 texCUBEARRAYbias

NAME

texCUBEARRAYbias - cube array texture lookup with bias.

SYNOPSIS

float4 texCUBEARRAYbias(samplerCUBEARRAY samp, float4 s, float bias)
int4 texCUBEARRAYbias(isamplerCUBEARRAY samp, float4 s, float bias)
unsigned int4 texCUBEARRAYbias(usamplerCUBEARRAY samp, float4 s, float bias)

PARAMETERS

samp
Cube array sampler array to look up.
s.xyz
Coordinates to perform the lookup.
s.w
Cube map array layer.

bias
Level of detail bias.

DESCRIPTION
Performs a texture lookup in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates. Level-of-detail bias can be used to artificially blur or sharpen the result of texture sampling.

PROFILE SUPPORT
texCUBEARRAYbias is only supported in gp5 and newer profiles.

SEE ALSO
texCUBEARRAY, texCUBEARRAYlod

9.113 texCUBEARRAYlod

NAME
texCUBEARRAYlod - cube array texture lookup with specified level of detail.

SYNOPSIS
float4 texCUBEARRAYlod(samplerCUBEARRAY samp, float4 s, float lod)
int4 texCUBEARRAYlod(isamplerCUBEARRAY samp, float4 s, float lod)
unsigned int4 texCUBEARRAYlod(usamplerCUBEARRAY samp, float4 s, float lod)

PARAMETERS
samp
Cube array sampler to look up.
s.xyz
Coordinates to perform the lookup.
s.w
Cube map array layer.
lod
Level of detail.

DESCRIPTION
Performs a texture lookup in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates.

PROFILE SUPPORT
texCUBEARRAYlod is only supported in gp5 and newer profiles.
SEE ALSO
texCUBEARRAY, texCUBEARRAYbias

9.114 texCUBEARRAY

NAME
texCUBEARRAY - cube array texture lookup with optional pre-computed derivatives.

SYNOPSIS

float4 texCUBEARRAY(samplerCUBEARRAY samp, float4 s)
float4 texCUBEARRAY(samplerCUBEARRAY samp, float4 s, float3 dx, float3 dy)

int4 texCUBEARRAY(isamplerCUBEARRAY samp, float4 s)
int4 texCUBEARRAY(isamplerCUBEARRAY samp, float4 s, float3 dx, float3 dy)

unsigned int4 texCUBEARRAY(usamplerCUBEARRAY samp, float4 s)
unsigned int4 texCUBEARRAY(usamplerCUBEARRAY samp, float4 s, float3 dx, float3 dy)

PARAMETERS

samp
Cube array sampler array to look up.

s.xyz
Coordinates to perform the lookup.

s.w
Cube map array layer.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the texture to be sampled is selected from the layer specified in the coordinates. Also may use the derivatives dx and dy.

PROFILE SUPPORT
texCUBEARRAY is only supported in gp5 and newer profiles.

SEE ALSO
texCUBEARRAYbias, texCUBEARRAYlod, texCUBEARRAYsize
9.115 texCUBEARRAYsize

NAME
texCUBEARRAYsize - returns the size of a given texture array image for a given level of detail.

SYNOPSIS
int3 texCUBEARRAYsize(samplerCUBEARRAY samp, int lod)
int3 texCUBEARRAYsize(isamplerCUBEARRAY samp, int lod)
int3 texCUBEARRAYsize(usamplerCUBEARRAY samp, int lod)

PARAMETERS
samp
Sampler to be queried for size.
lod
Level of detail to obtain size.

DESCRIPTION
Given a sampler array and a level of detail the size of one element of the corresponding texture array for a given level of detail is returned as a result of the operation.

PROFILE SUPPORT
texCUBEARRAYsize is only supported in gp5 and newer profiles.

SEE ALSO
texCUBEsize

texCUBEbias

9.116 texCUBEbias

NAME
texCUBEbias - cube texture lookup with bias.

SYNOPSIS
float4 texCUBEbias(samplerCUBE samp, float4 s)
int4 texCUBEbias(isamplerCUBE samp, float4 s)
unsigned int4 texCUBEbias(usamplerCUBE samp, float4 s)

PARAMETERS
samp
Sampler to lookup.
s.xyz
Coordinates to perform the lookup.
s.w
Level of detail bias value.

**DESCRIPTION**

Performs a texture lookup with bias in sampler `samp` using coordinates `s`.

**PROFILE SUPPORT**

`texCUBEdtias` is supported in fragment profiles starting with `fp30` and in vertex profiles starting with `vp40`. Variants with integer samplers are only supported in `gp4` and newer profiles.

**SEE ALSO**

`texCUBEdlod`

### 9.117 `texCUBEdlod`

**NAME**

`texCUBEdlod` - cube texture lookup with specified level of detail.

**SYNOPSIS**

```plaintext
float4 texCUBEdlod(samplerCUBE samp, float4 s)
int4 texCUBEdlod(isamplerCUBE samp, float4 s)
unsigned int4 texCUBEdlod(usamplerCUBE samp, float4 s)
```

**PARAMETERS**

- `samp`
  
  Sampler to lookup.
  
  `s.xyz`
  
  Coordinates to perform the lookup.
  
  `s.w`
  
  Level of detail.

**DESCRIPTION**

Performs a texture lookup with a specified level of detail in sampler `samp` using coordinates `s`.

**PROFILE SUPPORT**

`texCUBEdlod` is supported in fragment profiles starting with `fp40` and in vertex profiles starting with `vp40`. Variants with integer samplers are only supported in `gp4` and newer profiles.

**SEE ALSO**

`texCUBEdtias`
9.118 texCUBE

NAME

texCUBE - performs a texture lookup in a given CUBE sampler and, in some cases, a shadow comparison. May also use pre computed derivatives if those are provided.

SYNOPSIS

float4 texCUBE(samplerCUBE samp, float3 s)
float4 texCUBE(samplerCUBE samp, float4 s)
float4 texCUBE(samplerCUBE samp, float3 s, float3 dx, float3 dy)
float4 texCUBE(samplerCUBE samp, float4 s, float3 dx, float3 dy)
int4 texCUBE(isamplerCUBE samp, float3 s)
int4 texCUBE(isamplerCUBE samp, float3 s, float3 dx, float3 dy)
unsigned int4 texCUBE(usamplerCUBE samp, float3 s)
unsigned int4 texCUBE(usamplerCUBE samp, float3 s, float3 dx, float3 dy)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. If an extra coordinate compared to the texture dimensionality is present it is used to perform a shadow comparison. The value used in the shadow comparison is always the last component of the coordinate vector.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, may use and derivatives dx and dy, also may perform shadow comparison.

PROFILE SUPPORT

texCUBE is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

texCUBEbias, texCUBElod, texCUBEproj
9.119 texCUBEproj

NAME

texCUBEproj - performs a texture lookup with projection in a given sampler.

SYNOPSIS

float4 texCUBEproj(samplerCUBE samp, float4 s)

int4 texCUBEproj(isamplerCUBE samp, float4 s)

unsigned int4 texCUBEproj(usamplerCUBE samp, float4 s)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the projection should be passed as the last compo-
nent of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the coordinates used in the lookup are first projected,
that is, divided by the last component of the coordinate vector and then used in the lookup.

PROFILE SUPPORT

texCUBEproj is supported in all fragment profiles and all vertex profiles starting with vp40. Variants with integer
samplers are only supported in gp4 and newer profiles.

SEE ALSO

texCUBE

9.120 texCUBESize

NAME

texCUBESize - returns the size of a given texture image for a given level of detail.

SYNOPSIS

int3 texCUBESize(samplerCUBE samp, int lod)

int3 texCUBESize(isamplerCUBE samp, int lod)

int3 texCUBESize(usamplerCUBE samp, int lod)
PARAMETERS

samp
Sampler to be queried for size.

lod
Level of detail to obtain size.

DESCRIPTION

Given a sampler and a level of detail the size of the corresponding texture image is returned as the result of the operation.

PROFILE SUPPORT

texCUBESize is only supported in gp4 and newer profiles.

SEE ALSO

texCUBE

9.121 texRBUF

NAME

texRBUF - performs an unfiltered multisample texture lookup in a given renderbuffer sampler.

SYNOPSIS

float4 texRBUF(samplerBUF samp, int2 c, int s)
int4 texRBUF(isamplerBUF samp, int2 c, int s)
uint4 texBUF(usamplerRBUF samp, int2 c, int s)

PARAMETERS

samp
Sampler to lookup.

c
Coordinates to perform the lookup.

s
sample within a multi-sampled renderbuffer.

DESCRIPTION

Performs an unfiltered multisample texture lookup in renderbuffer sampler samp using coordinates c and sample s. Renderbuffer samplers are created with the NV_explicit_multisample extension. See:

GL_NV_explicit_multisample

PROFILE SUPPORT

texRBUF is supported in gp4, gp5, and glsl profiles.
### 9.122 texRBUFsize

**NAME**

texRBUFsize - returns the size of a given renderbuffer texture.

**SYNOPSIS**

```c
int2 texRBUFsize(samplerRBUF samp)
int2 texRBUFsize(isamplerRBUF samp)
int2 texRBUFsize(usamplerRBUF samp)
```

**PARAMETERS**

- `samp`
  
  Sampler to be queried for size.

**DESCRIPTION**

Given a sampler the size (width in x, height in y) of the corresponding renderbuffer is returned as the result of the operation.

Renderbuffer samplers are created with the `NV_explicit_multisample` extension. See:

- GL_NV_explicit_multisample

**PROFILE SUPPORT**

texRBUFsize is supported in `gp4`, `gp5`, and `glsl` profiles.

**SEE ALSO**

texRBUF

### 9.123 texRECTbias

**NAME**

texRECTbias - rectangle texture lookup with bias and optional texel offset.

**SYNOPSIS**

```c
float4 texRECTbias(samplerRECT samp, float4 s)
float4 texRECTbias(samplerRECT samp, float4 s, int2 texelOff)
int4 texRECTbias(isamplerRECT samp, float4 s)
int4 texRECTbias(isamplerRECT samp, float4 s, int2 texelOff)
unsigned int4 texRECTbias(usamplerRECT samp, float4 s)
unsigned int4 texRECTbias(usamplerRECT samp, float4 s, int2 texelOff)
```
PARAMETERS

samp
Sampler to lookup.
s.xy
Coordinates to perform the lookup.
s.w
Level of detail bias value.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with bias in sampler samp using coordinates s.

PROFILE SUPPORT

texRECTbias is supported in fragment profiles starting with fp30 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.

SEE ALSO

texRECTlod, texRECTfetch

9.124 texRECTfetch

NAME

texRECTfetch - unfiltered rectangle texture lookup.

SYNOPSIS

float4 texRECTfetch(samplerRECT samp, int4 s)
float4 texRECTfetch(samplerRECT samp, int4 s, int2 texelOff)
int4 texRECTfetch(isamplerRECT samp, int4 s)
int4 texRECTfetch(isamplerRECT samp, int4 s, int2 texelOff)
unsigned int4 texRECTfetch(usamplerRECT samp, int4 s)
unsigned int4 texRECTfetch(usamplerRECT samp, int4 s, int2 texelOff)

PARAMETERS

samp
Sampler to lookup.
s
Coordinates to perform the lookup. The level of detail is stored in the last component of the coordinate vector.
texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs an unfiltered texture lookup in sampler samp using coordinates s. The level of detail is provided by the last component of the coordinate vector. May use texel offset texelOff to compute final texel.

PROFILE SUPPORT

texRECTFetch is only supported in gp4 and newer profiles.

SEE ALSO

texRECTlod, texRECTbias, texRECT

9.125 texRECTlod

NAME

texRECTlod - rectangle texture lookup with specified level of detail and optional texel offset.

SYNOPSIS

float4 texRECTlod(samplerRECT samp, float4 s)
float4 texRECTlod(samplerRECT samp, float4 s, int texelOff)

int4 texRECTlod(isamplerRECT samp, float4 s)
int4 texRECTlod(isamplerRECT samp, float4 s, int texelOff)

unsigned int4 texRECTlod(usamplerRECT samp, float4 s)
unsigned int4 texRECTlod(usamplerRECT samp, float4 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s.xyz
Coordinates to perform the lookup.

s.w
Level of detail.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup with a specified level of detail in sampler samp using coordinates s.

PROFILE SUPPORT

texRECTlod is supported in fragment profiles starting with fp40 and in vertex profiles starting with vp40. Variants with texelOff are only supported in gp4 and newer profiles. Variants with integer samplers are also only supported in gp4 and newer profiles.
9.126 texRECT

NAME

texRECT - performs a texture lookup in a given RECT sampler and, in some cases, a shadow comparison. May also use pre computed derivatives if those are provided.

SYNOPSIS

float4 texRECT(samplerRECT samp, float2 s)
float4 texRECT(samplerRECT samp, float2 s, int texelOff)
float4 texRECT(samplerRECT samp, float3 s)
float4 texRECT(samplerRECT samp, float3 s, int texelOff)

float4 texRECT(samplerRECT samp, float2 s, float2 dx, float2 dy)
float4 texRECT(samplerRECT samp, float2 s, float2 dx, float2 dy, int texelOff)
float4 texRECT(samplerRECT samp, float3 s, float2 dx, float2 dy)
float4 texRECT(samplerRECT samp, float3 s, float2 dx, float2 dy, int texelOff)

int4 texRECT(isamplerRECT samp, float2 s)
int4 texRECT(isamplerRECT samp, float2 s, int texelOff)

int4 texRECT(isamplerRECT samp, float2 s, float2 dx, float2 dy)
int4 texRECT(isamplerRECT samp, float2 s, float2 dx, float2 dy, int texelOff)

unsigned int4 texRECT(usamplerRECT samp, float2 s)
unsigned int4 texRECT(usamplerRECT samp, float2 s, int texelOff)

unsigned int4 texRECT(usamplerRECT samp, float2 s, float2 dx, float2 dy)
unsigned int4 texRECT(usamplerRECT samp, float2 s, float2 dx, float2 dy, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. If an extra coordinate compared to the texture dimensionality is present it is used to perform a shadow comparison. The value used in the shadow comparison is always the last component of the coordinate vector.

dx
Pre computed derivative along the x axis.

dy
Pre computed derivative along the y axis.
texelOff
Offset to be added to obtain the final texel.
DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, may use and derivatives dx and dy, also may perform shadow comparison and use texel offset texelOff to compute final texel.

PROFILE SUPPORT

texRECT is supported in all fragment profiles and all vertex profiles starting with vp40, variants with shadow comparison are only supported in fp40 and newer profiles, variants with texel offsets are only supported in gp4 and newer profiles. Variants with integer samplers are only supported in gp4 and newer profiles.

SEE ALSO

texRECTbias, texRECTlod, texRECTproj

9.127 texRECTproj

NAME

texRECTproj - performs a texture lookup with projection in a given sampler. May perform a shadow comparison if argument for shadow comparison is provided.

SYNOPSIS

float4 texRECTproj(samplerRECT samp, float3 s)
float4 texRECTproj(samplerRECT samp, float3 s, int texelOff)
float4 texRECTproj(samplerRECT samp, float4 s)
float4 texRECTproj(samplerRECT samp, float4 s, int texelOff)

int4 texRECTproj(isamplerRECT samp, float3 s)
int4 texRECTproj(isamplerRECT samp, float3 s, int texelOff)

unsigned int4 texRECTproj(usamplerRECT samp, float3 s)
unsigned int4 texRECTproj(usamplerRECT samp, float3 s, int texelOff)

PARAMETERS

samp
Sampler to lookup.

s
Coordinates to perform the lookup. The value used in the projection should be passed as the last component of the coordinate vector. The value used in the shadow comparison, if present, should be passed as the next-to-last component of the coordinate vector.

texelOff
Offset to be added to obtain the final texel.

DESCRIPTION

Performs a texture lookup in sampler samp using coordinates s, the coordinates used in the lookup are first projected, that is, divided by the last component of the coordinate vector and then used in the lookup. If an extra coordinate is present it is used to perform a shadow comparison, the value used in the shadow comparison is always the next-to-last component in the coordinate vector.
PROFILE SUPPORT

texRECTproj is supported in all fragment profiles and all vertex profiles starting with *vp40*, variants with shadow comparison are only supported in *fp40* and newer profiles, variants with texel offsets are only supported in *gp4* and newer profiles.

SEE ALSO

texRECT

9.128 texRECTsize

NAME

texRECTsize - returns the size of a given texture image for a given level of detail.

SYNOPSIS

```cpp
int3 texRECTsize(samplerRECT samp, int lod)
int3 texRECTsize(isamplerRECT samp, int lod)
int3 texRECTsize(usamplerRECT samp, int lod)
```

PARAMETERS

- **samp**
  - Sampler to be queried for size.
- **lod**
  - Level of detail to obtain size.

DESCRIPTION

Given a sampler and a level of detail the size of the corresponding texture image is returned as the result of the operation.

PROFILE SUPPORT

texRECTsize is only supported in *gp4* and newer profiles.

SEE ALSO

texRECT

9.129 transpose

NAME

transpose - returns transpose matrix of a matrix

SYNOPSIS
float4x4 transpose(float4x4 A)
float3x4 transpose(float3x4 A)
float2x4 transpose(float4x2 A)
float1x4 transpose(float4x1 A)

float4x3 transpose(float3x4 A)
float3x3 transpose(float3x3 A)
float2x3 transpose(float3x2 A)
float1x3 transpose(float3x1 A)

float4x2 transpose(float2x4 A)
float3x2 transpose(float2x3 A)
float2x2 transpose(float2x2 A)
float1x2 transpose(float2x1 A)

float4x1 transpose(float1x4 A)
float3x1 transpose(float1x3 A)
float2x1 transpose(float1x2 A)
float1x1 transpose(float1x1 A)

PARAMETERS

A
Matrix to transpose.

DESCRIPTION

Returns the transpose of the matrix A.

REFERENCE IMPLEMENTATION

transpose for a float4x3 matrix can be implemented like this:

```c
float4x3 transpose(float3x4 A)
{
    float4x3 C;
    C[0] = A._m00_m10_m20;
    C[1] = A._m01_m11_m21;
    C[2] = A._m02_m12_m22;
    C[3] = A._m03_m13_m23;
    return C;
}
```

PROFILE SUPPORT

transpose is supported in all profiles.

SEE ALSO

determinant, mul
9.130 \texttt{trunc}

\textbf{NAME}

\texttt{trunc} - returns largest integer not greater than a scalar or each vector component.

\textbf{SYNOPSIS}

\begin{verbatim}
float  trunc(float  x);
float1 trunc(float1 x);
float2 trunc(float2 x);
float3 trunc(float3 x);
float4 trunc(float4 x);

half  trunc(half  x);
half1 trunc(half1 x);
half2 trunc(half2 x);
half3 trunc(half3 x);
half4 trunc(half4 x);

fixed trunc(fixed  x);
fixed1 trunc(fixed1 x);
fixed2 trunc(fixed2 x);
fixed3 trunc(fixed3 x);
fixed4 trunc(fixed4 x);
\end{verbatim}

\textbf{PARAMETERS}

\texttt{x}

Vector or scalar which to truncate.

\textbf{DESCRIPTION}

Returns the integral value nearest to but no larger in magnitude than \texttt{x}.

\textbf{REFERENCE IMPLEMENTATION}

\texttt{trunc} for a \texttt{float3} vector could be implemented like this.

\begin{verbatim}
def trunc(float3 v)
{
    float3 rv;
    int  i;
    for (i=0; i<3; i++) {
        float  x = v[i];
        rv[i] = x < 0 ? -floor(-x) : floor(x);
    }
    return rv;
}
\end{verbatim}

\textbf{PROFILE SUPPORT}

\texttt{trunc} is supported in all profiles except \texttt{fp20}.

\textbf{SEE ALSO}

\texttt{ceil, floor, round}
9.131 unpack

NAME

unpack - unpacks a single values into different formatted values

SYNOPSIS

half2 unpack_2half(float a);
float2 unpack_2ushort(float a);
half4 unpack_4byte(float a);
half4 unpack_4ubyte(float a);

PARAMETERS

a

Value to unpack

DESCRIPTION

unpack_2half unpacks a 32-bit integer value into two 16-bit floating point value.

unpack_2ushort unpacks two 16-bit unsigned integer values from a and scales the results into individual floating point values between 0.0 and 1.0.

unpack_4byte unpacks four 8-bit integers from a and scales the results into individual 16-bit floating point values between -(128/127) and +(127/127).

unpack_4ubyte unpacks the four 8-bit integers in a and scales the results into individual 16-bit floating point values between 0.0 and 1.0.

REFERENCE IMPLEMENTATION

unpack_2half could be implemented this way:

```cpp
half2 unpack_2half(float a)
{
    result.x = (a >> 0) & 0xFF;
    result.y = (a >> 16) & 0xFF;
    return result;
}
```

unpack_2ushort could be implemented this way:

```cpp
float2 unpack_2ushort(float a)
{
    float2 result;
    result.x = ((x >> 0) & 0xFFFF) / 65535.0;
    result.y = ((x >> 16) & 0xFFFF) / 65535.0;
    return result;
}
```

unpack_4byte could be implemented this way:

```cpp
half4 unpack_4byte(float a)
{
    half4 result;
    result.x = (((a >> 0) & 0xFF) - 128) / 127.0;
    result.y = (((a >> 8) & 0xFF) - 128) / 127.0;
    result.z = (((a >> 16) & 0xFF) - 128) / 127.0;
    return result;
}
```
result.w = (((a >> 24) & 0xFF) - 128) / 127.0;

return result;
}

unpack_4ubyte could be implemented this way:

half4 unpack_4ubyte(float a)
{
    half4 result ;
    result.x = ((a >> 0) & 0xFF) / 255.0;
    result.y = ((a >> 8) & 0xFF) / 255.0;
    result.z = ((a >> 16) & 0xFF) / 255.0;
    result.w = ((a >> 24) & 0xFF) / 255.0;
    return result;
}

PROFILE SUPPORT

unpack is supported in fp30, fp40, gp4, gp5

SEE ALSO

pack
10.1 AddressUi

NAME
AddressUi - 3D API U texture addressing mode for unit \( i \)

USAGE
AddressU[i] = int(mode)

VALID ENUMERANTS
mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION
Sets the Direct3D u-direction wrapping mode for texture unit \( i \), where \( i \) is in the range 0-15. See sampler state AddressU for details.

The standard reset callback sets the AddressU[i] state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AddressU, AddressVi, AddressWi

10.2 AddressVi

NAME
AddressVi - 3D API V texture addressing mode for unit \( i \)

USAGE
AddressV[i] = int(mode)
VALID ENUMERANTS

mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION

Sets the Direct3D v-direction wrapping mode for texture unit $i$, where $i$ is in the range 0-15. See sampler state AddressV for details.

The standard reset callback sets the AddressV[i] state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AddressV, AddressUi, AddressWi

10.3 AddressWi

NAME

AddressWi - 3D API W texture addressing mode for unit $i$

USAGE

AddressW[i] = int(mode)

VALID ENUMERANTS

mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION

Sets the Direct3D w-direction wrapping mode for texture unit $i$, where $i$ is in the range 0-15. See sampler state AddressW for details.

The standard reset callback sets the AddressW[i] state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AddressW, AddressUi, AddressVi
10.4 AlphaArg0

NAME

AlphaArg0 - 3D API AlphaArg0

USAGE

AlphaArg0 = int(third)

VALID ENUMERANTS

third: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION

Specify the alpha channel selector operand for triadic operations. See the D3DTSS_ALPHAARG0 texture stage state description for DirectX 9.

The standard reset callback sets the AlphaArg0 state to int(Current).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.5 AlphaArg1

NAME

AlphaArg1 - 3D API AlphaArg1

USAGE

AlphaArg1 = int(first)

VALID ENUMERANTS

first: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION

Specify the first alpha argument for the stage. See the D3DTSS_ALPHAARG1 texture stage state description for DirectX 9.

The standard reset callback sets the AlphaArg1 state to int(Texture).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaArg0, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.6 AlphaArg2

NAME

AlphaArg2 - 3D API AlphaArg2

USAGE

AlphaArg2 = int(second)

VALID ENUMERANTS

second: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION

Specify the second alpha argument for the stage. See the D3DTSS_ALPHAARG2 texture stage state description for DirectX 9.

The standard reset callback sets the AlphaArg2 state to int(Current).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaArg0, AlphaArg1, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.7 AlphaBlendEnable

NAME

AlphaBlendEnable - 3D API alpha blend enable

USAGE

AlphaBlendEnable = bool(enable)

VALID ENUMERANTS

enable: true, false
**DESCRIPTION**
Enable/disable alpha blending. See the GL_BLEND description on the OpenGL glEnable manual page for details.
See the D3DRS_ALPHABLENDENABLE render state description for DirectX 9.
The standard reset callback sets the AlphaBlendEnable state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**
OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**
DirectX 9

**SEE ALSO**
BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

### 10.8 AlphaFunc

**NAME**
AlphaFunc - 3D API alpha function

**USAGE**
AlphaFunc = float(func) AlphaFunc = float2(func, ref)

**VALID ENUMERANTS**
func: Never, Less, LEqual, Equal, Greater, NotEqual, GEQual, Always

**DESCRIPTION**
Specify the alpha comparison function. See the OpenGL glAlphaFunc manual page for details. See the D3DRS_ALPHAFunc render state description for DirectX 9.
The standard reset callback sets the AlphaFunc state to float(Always) or float2(Always, 0).

**OPENGL FUNCTIONALITY REQUIREMENTS**
OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**
DirectX 9

**SEE ALSO**
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha
10.9 AlphaOp

NAME

AlphaOp - 3D API alpha operator

USAGE

AlphaOp = int(op)

VALID ENUMERANTS

op: Disable, SelectArg1, SelectArg2, Modulate, Modulate2x, Modulate4x, Add, AddSigned, AddSigned2x, Subtract, AddSmooth, BlendDiffuseAlpha, BlendTextureAlpha, BlendFactorAlpha, BlendTextureAlphaPM, BlendCurrentAlpha, PreModulate, ModulateAlpha_AddColor, ModulateColor_AddAlpha, ModulateInvAlpha_AddColor, ModulateInvColor_AddAlpha, BumpEnvMap, BumpEnvMapLuminance, DotProduct3, MultiplyAdd, Lerp

DESCRIPTION

Specify the alpha blending operation for this texture stage. See the D3DTSS_ALPHAOP texture stage state description for DirectX 9.

The standard reset callback sets the AlphaOp state to int(SelectArg1) if $i$ is zero and int(Disable) otherwise.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.10 AlphaRef

NAME

AlphaRef - 3D API alpha ref

USAGE

AlphaRef = float(ref)

VALID ENUMERANTS

ref: floating point value

DESCRIPTION

Specify the alpha reference value. See the OpenGL glAlphaFunc manual page for details. See the D3DRS_ALPHAREF render state description for DirectX 9.

The standard reset callback sets the AlphaRef state to float(0).
OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.11 AlphaTestEnable

NAME
AlphaTestEnable - 3D API alpha test enable

USAGE
AlphaTestEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable alpha testing. See the GL_ALPHA_TEST description on the OpenGL glEnable manual page for details. See the D3DRS_ALPHATESTENABLE render state description for DirectX 9.
The standard reset callback sets the AlphaTestEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.12 AmbientMaterialSource

NAME
AmbientMaterialSource - 3D API ambient material source

USAGE
AmbientMaterialSource = int(enable)
VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable front and back ambient color material. See the GL_AMBIENT description on the OpenGL glColorMaterial manual page for details. See the D3DRS_AMBIENTMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the AmbientMaterialSource state to int(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.13 Ambient

NAME

Ambient - 3D API ambient

USAGE

Ambient = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Specify the ambient intensity of the entire scene. See the GL_LIGHT_MODEL_AMBIENT description on the OpenGL glLightModel manual page for details. See the D3DRS_AMBIENT render state description for DirectX 9.

The standard reset callback sets the Ambient state to float4(0.2, 0.2, 0.2, 1.0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightModelAmbient
10.14 AutoNormalEnable

NAME
AutoNormalEnable - 3D API auto normal enable

USAGE
AutoNormalEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable automatic normal generation. See the GL_AUTO_NORMAL description on the OpenGL glEnable manual page for details. See the D3DRS_NORMALIZENORMALS render state description for DirectX 9.
The standard reset callback sets the AutoNormalEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
NormalizeEnable, RescaleNormalEnable, NormalizeNormals

10.15 BlendColor

NAME
BlendColor - 3D API blend color

USAGE
BlendColor = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Set the blend color. See the OpenGL glBlendColor manual page for details. See the D3DRS_BLENDFACTOR render state description for DirectX 9.
The standard reset callback sets the BlendColor state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
10.16 BlendEnable

NAME
BlendEnable - 3D API blend enable

USAGE
BlendEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable color blending. See the GL_BLEND description on the OpenGL glEnable manual page for details. See the D3DRS_ALPHABLENDENABLE render state description for DirectX 9.

The standard reset callback sets the BlendEnable state to bool(false).

OPENCORE FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaBlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.17 BlendEquation

NAME
BlendEquation - 3D API blend equation

USAGE
BlendEquation = int(mode)

VALID ENUMERANTS
mode: FuncAdd, FuncSubtract, FuncReverseSubtract, Add, Subtract, ReverseSubtract, Min, Max, LogicOp
DESCRIPTION

Specify the RGB and alpha blend equation together. See the OpenGL glBlendEquation manual page for details. See the D3DRS_BLENDOP render state description for DirectX 9.

The standard reset callback sets the BlendEquation state to int(Add).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.18 BlendEquationSeparate

NAME

BlendEquationSeparate - 3D API blend equation separate

USAGE

BlendEquationSeparate = int2(modeRGB, modeAlpha)

VALID ENUMERANTS

modeRGB, modeAlpha: FuncAdd, FuncSubtract, FuncReverseSubtract, Add, Subtract, ReverseSubtract, Min, Max, LogicOp

DESCRIPTION

Specify the RGB and alpha blend equation separately. See the OpenGL glBlendEquationSeparate manual page for details. See the D3DRS_BLENDOP render state description for DirectX 9.

The standard reset callback sets the BlendEquationSeparate state to int2(FuncAdd, FuncAdd).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable
10.19 BlendFunc

NAME
BlendFunc - 3D API blend function

USAGE
BlendFunc = int2(sfactor, dfactor)

VALID ENUMERANTS
sfactor, dfactor: Zero, One, DestColor, OneMinusDestColor, InvDestColor, SrcAlpha, OneMinusSrcAlpha, InvSrcAlpha, DstAlpha, OneMinusDstAlpha, InvDestAlpha, SrcAlphaSaturate, SrcAlphaSat, SrcColor, OneMinusSrcColor, InvSrcColor, ConstantColor, BlendFactor, OneMinusConstantColor, InvBlendFactor, ConstantAlpha, OneMinusConstantAlpha

DESCRIPTION
Specify the source and destination blending functions. See the OpenGL glBlendFunc manual page for details. See the D3DRS_SRCBLEND and D3DRS_DESTBLEND render state descriptions for DirectX 9.

The standard reset callback sets the BlendFunc state to int2(One, Zero).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.20 BlendFuncSeparate

NAME
BlendFuncSeparate - 3D API blend function separate

USAGE
BlendFuncSeparate = int4(srcRGB, dstRGB, srcAlpha, dstAlpha)

VALID ENUMERANTS
srcRGB, dstRGB, srcAlpha, dstAlpha: Zero, One, DestColor, OneMinusDestColor, InvDestColor, SrcAlpha, OneMinusSrcAlpha, InvSrcAlpha, DstAlpha, OneMinusDstAlpha, InvDestAlpha, SrcAlphaSaturate, SrcAlphaSat, SrcColor, OneMinusSrcColor, InvSrcColor, ConstantColor, BlendFactor, OneMinusConstantColor, InvBlendFactor, ConstantAlpha, OneMinusConstantAlpha
DESCRIPTION

Specify the source and destination blending functions. See the OpenGL glBlendFuncSeparate manual page for details. See the D3DRS_SEPARATEALPHABLENDENABLE, D3DRS_SRCBLEND, D3DRS_DESTBLEND, D3DRS_SRCBLENDALPHA, and D3DRS_DESTBLENDALPHA render state descriptions for DirectX 9.

The standard reset callback sets the BlendFuncSeparate state to int4(One, Zero, One, Zero).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.21 BlendOpAlpha

NAME

BlendOpAlpha - 3D API blend operator alpha

USAGE

BlendOpAlpha = int(blendop)

VALID ENUMERANTS

blendop: Add, Subtract, RevSubtract, Min, Max

DESCRIPTION

Set the Direct3D separate alpha blending operation. See the D3DRS_BLENDOPALPHA render state description for DirectX 9.

The standard reset callback sets the BlendOpAlpha state to int(Add).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BlendOp, AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable
10.22 BlendOp

NAME

BlendOp - 3D API blend operator

USAGE

BlendOp = int(func)

VALID ENUMERANTS

func: FuncAdd, FuncSubtract, FuncReverseSubtract, Add, Subtract, ReverseSubtract, Min, Max, LogicOp

DESCRIPTION

Specify the blending equation. See the OpenGL glBlendEquation manual page for details. See the D3DRS_BLENDOP render state description for DirectX 9.

The standard reset callback sets the BlendOp state to int(FuncAdd).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.23 BorderColori

NAME

BorderColori - 3D API texture border color

USAGE

BorderColor[i] = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Set the Direct3D sampler border color for texture unit i, where i is in the range 0-15. See sampler state BorderRadius for details.

The standard reset callback sets the BorderColor[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BorderColor

10.24 BumpEnvLOffset

NAME

BumpEnvLOffset - 3D API bump environment L offset

USAGE

BumpEnvLOffset = float(offset)

VALID ENUMERANTS

offset: floating point value

DESCRIPTION

Specify the offset value for bump-map luminance. See the D3DTSS_BUMPENVLOFFSET texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvLOffset state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BumpEnvLScale, BumpEnvMat00, BumpEnvMat01, BumpEnvMat10, BumpEnvMat11

10.25 BumpEnvLScale

NAME

BumpEnvLScale - 3D API bump environment L scale

USAGE

BumpEnvLScale = float(scale)

VALID ENUMERANTS

scale: floating point value
DESCRIPTION
Specify the scale value for bump-map luminance. See the D3DTSS_BUMPENVLSCALE texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvLScale state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
BumpEnvLOffset, BumpEnvMat00, BumpEnvMat01, BumpEnvMat10, BumpEnvMat11

10.26 BumpEnvMat00

NAME
BumpEnvMat00 - 3D API bump environment mat00

USAGE
BumpEnvMat00 = float(c00)

VALID ENUMERANTS

c00: floating point value

DESCRIPTION
Specify that this texture-stage state is a floating-point value for the [0][0] coefficient in a bump-mapping matrix. See the D3DTSS_BUMPENVMAT00 texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvMat00 state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
BumpEnvMat01, BumpEnvMat10, BumpEnvMat11, BumpEnvLOffset, BumpEnvLScale

10.27 BumpEnvMat01

NAME
BumpEnvMat01 - 3D API bump environment mat01
**USAGE**

BumpEnvMat01 = float(c01)

**VALID ENUMERANTS**

c01: floating point value

**DESCRIPTION**

Specify that this texture-stage state is a floating-point value for the [0][1] coefficient in a bump-mapping matrix. See the D3DTSS_BUMPENVMAT01 texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvMat01 state to float(0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

BumpEnvMat00, BumpEnvMat10, BumpEnvMat11, BumpEnvLOffset, BumpEnvLScale

---

**10.28 BumpEnvMat10**

**NAME**

BumpEnvMat10 - 3D API bump environment mat10

**USAGE**

BumpEnvMat10 = float(c10)

**VALID ENUMERANTS**

c10: floating point value

**DESCRIPTION**

Specify that this texture-stage state is a floating-point value for the [1][0] coefficient in a bump-mapping matrix. See the D3DTSS_BUMPENVMAT10 texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvMat10 state to float(0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

BumpEnvMat00, BumpEnvMat01, BumpEnvMat11, BumpEnvLOffset, BumpEnvLScale
10.29 BumpEnvMat11

NAME

BumpEnvMat11 - 3D API bump environment mat11

USAGE

BumpEnvMat11 = float(c11)

VALID ENUMERANTS

c11: floating point value

DESCRIPTION

Specify that this texture-stage state is a floating-point value for the \([1][1]\) coefficient in a bump-mapping matrix. See the D3DTSS_BUMPENVMAT11 texture stage state description for DirectX 9.

The standard reset callback sets the BumpEnvMat11 state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BumpEnvMat00, BumpEnvMat01, BumpEnvMat10, BumpEnvLOffset, BumpEnvLScale

10.30 ClearColor

NAME

ClearColor - 3D API clear color

USAGE

ClearColor = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Specify the values used by to clear the color buffers. See the OpenGL glClearColor manual page for details. See the D3DCLEAR_TARGET flag description for the IDirect3DDevice9::Clear method.

The standard reset callback sets the ClearColor state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
10.31 ClearDepth

NAME
ClearDepth - 3D API clear depth

USAGE
ClearDepth = float(z)

VALID ENUMERANTS
z: floating point value

DESCRIPTION
Specify the value used by to clear the depth buffer. See the OpenGL glClearDepth manual page for details. See the D3DCLEAR_ZBUFFER flag description for the IDirect3DDevice9::Clear method.

The standard reset callback sets the ClearDepth state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearColor, ClearStencil

10.32 ClearStencil

NAME
ClearStencil - 3D API clear stencil

USAGE
ClearStencil = int(val)

VALID ENUMERANTS
val: integer value
DESCRIPTION

Specify the value used by to clear the stencil buffer. See the OpenGL glClearStencil manual page for details. See the D3DCLEAR_STENCIL flag description for the IDirect3DDevice9::Clear method.

The standard reset callback sets the ClearStencil state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.33 Clipping

NAME

Clipping - 3D API clipping

USAGE

Clipping = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable Direct3D primitive clipping. See the D3DRS_CLIPPING render state description for DirectX 9.

The standard reset callback sets the Clipping state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClipPlane, ClipPlaneEnable

10.34 ClipPlaneEnable

NAME

ClipPlaneEnable - 3D API clip plane enable
USAGE
ClipPlaneEnable[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable user-defined clipping planes. See the OpenGL glClipPlane manual page for details. See the D3DRS_CLIPPLANEENABLE render state description for DirectX 9.
The standard reset callback sets the ClipPlaneEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClipPlane, Clipping

10.35 ClipPlane

NAME
ClipPlane - 3D API clip plane

USAGE
ClipPlane[i] = float4(a, b, c, d)

VALID ENUMERANTS
a, b, c, d: floating point values

DESCRIPTION
Specify an clip plane. See the OpenGL glClipPlane manual page for details. See the IDirect3DDevice9::SetClipPlane method for details.
The standard reset callback sets the ClipPlane[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClipPlaneEnable, Clipping
10.36 ColorArg0

NAME
ColorArg0 - 3D API color arg0

USAGE
ColorArg0 = int(third)

VALID ENUMERANTS
third: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION
Specify settings for the third color operand for triadic operations. See the D3DTSS_COLORARG0 texture stage state description for DirectX 9.

The standard reset callback sets the ColorArg0 state to int(Current).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg1, ColorArg2, ColorOp, BlendOpAlpha

10.37 ColorArg1

NAME
ColorArg1 - 3D API color arg1

USAGE
ColorArg1 = int(first)

VALID ENUMERANTS
first: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION
Specify that this texture-stage state is the first color argument for the stage. See the D3DTSS_COLORARG1 texture stage state description for DirectX 9.

The standard reset callback sets the ColorArg1 state to int(Texture).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg2, ColorOp, BlendOpAlpha

10.38 ColorArg2

NAME
ColorArg2 - 3D API color arg2

USAGE
ColorArg2 = int(second)

VALID ENUMERANTS
second: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION
Specify that this texture-stage state is the second color argument for the stage. See the D3DTSS_COLORARG2 texture stage state description for DirectX 9.

The standard reset callback sets the ColorArg2 state to int(Current).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg2, ColorOp, BlendOpAlpha

10.39 ColorLogicOpEnable

NAME
ColorLogicOpEnable - 3D API color logic operation enable

USAGE
ColorLogicOpEnable = bool(enable)

VALID ENUMERANTS
enable: true, false
DESCRIPTION
Enable/disable OpenGL logical pixel operations. See the GL_COLOR_LOGIC_OP description on the OpenGL glEnable manual page for details.
The standard reset callback sets the ColorLogicOpEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LogicOp, LogicOpEnable

10.40 ColorMask

NAME
ColorMask - 3D API color mask

USAGE
ColorMask = bool4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: true, false

DESCRIPTION
Enable/disable writing to color components in the frame buffer. See the OpenGL glColorMask manual page for details.
See the D3DRS_COLORWRITEENABLE1 render state description for DirectX 9.
The standard reset callback sets the ColorMask state to bool4(true, true, true, true).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
DepthMask, StencilMask

10.41 ColorMaterial

NAME
ColorMaterial - 3D API color material
**USAGE**

ColorMaterial = int2(sides, mat)

**VALID ENUMERANTS**

sides: Front, Back, FrontAndBack
mat: Emission, Emissive, Ambient, Diffuse, Specular, AmbientAndDiffuse

**DESCRIPTION**

Specify which material parameters track the current color. See the OpenGL glColorMaterial manual page for details. See the D3DRS_ALPHABLENDENABLE render state description for DirectX 9.

The standard reset callback sets the ColorMaterial state to int2(FrontAndBack, AmbientAndDiffuse).

**OPENVGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**


### 10.42 ColorMatrix

**NAME**

ColorMatrix - 3D API color matrix

**USAGE**

ColorMatrix = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

**VALID ENUMERANTS**

fi: floating point values

**DESCRIPTION**

Set the values of the color matrix. See the GL_COLOR description on the OpenGL glMatrixMode manual page for details.

The standard reset callback sets the ColorMatrix state to the identity matrix.

**OPENVGL FUNCTIONALITY REQUIREMENTS**

Support for extension ARB_imaging.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

ColorTransform
10.43 ColorOp

NAME

ColorOp - 3D API color operation

USAGE

ColorOp = int(op)

VALID ENUMERANTS

op: Disable, SelectArg1, SelectArg2, Modulate, Modulate2x, Modulate4x, Add, AddSigned, AddSigned2x, Subtract, AddSmooth, BlendDiffuseAlpha, BlendTextureAlpha, BlendFactorAlpha, BlendTextureAlphaPM, BlendCurrentAlpha, PreModulate, ModulateAlpha_AddColor, ModulateColor_AddAlpha, ModulateInvAlpha_AddColor, ModulateInvColor_AddAlpha, BumpEnvMap, BumpEnvMapLuminance, DotProduct3, MultiplyAdd, Lerp

DESCRIPTION

Set the texture environment mode. See the GL_TEXTURE_ENV_MODE description on the OpenGL glTexEnv manual page for details. See the D3DTSS_COLOROP texture stage state description for DirectX 9.

The standard reset callback sets the ColorOp state to int(Modulate) or int(Disable).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

TextureEnvMode, AlphaArg0, AlphaArg1, AlphaArg2, AlphaBlendEnable, AlphaFunc, AlphaOp, AlphaRef, AlphaTestEnable, ColorArg0, ColorArg1, ColorArg2, BlendOpAlpha

10.44 ColorTransform

NAME

ColorTransform - 3D API color transform

USAGE

ColorTransform[i] = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the color matrix. See the GL_COLOR description on the OpenGL glMatrixMode manual page for details.

The standard reset callback sets the ColorTransform[i] state to the identity matrix.
OPENGL FUNCTIONALITY REQUIREMENTS
Support for extension ARB_imaging.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
ColorMatrix

10.45 ColorVertex

NAME
ColorVertex - 3D API color vertex

USAGE
ColorVertex = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable material parameter tracking of the current color. See the GL_COLOR_MATERIAL description on the OpenGL glEnable manual page for details. See the D3DRS_COLORVERTEX render state description for DirectX 9. The standard reset callback sets the ColorVertex state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ColorMaterial, AmbientMaterialSource, DiffuseMaterialSource, EmissiveMaterialSource, SpecularMaterialSource

10.46 ColorWriteEnable1

NAME
ColorWriteEnable1 - 3D API color write enable1

USAGE
ColorWriteEnable1 = bool4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: true, false
DESCRIPTION
Enable/disable writing of frame buffer color components. See the D3DRS_COLORWRITEENABLE1 render state description for DirectX 9.

The standard reset callback sets the ColorWriteEnable1 state to bool4(true, true, true, true).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ColorWriteEnable, ColorWriteEnable2, ColorWriteEnable3, BlendColor, ClearColor, ColorArg0, ColorArg1, ColorArg2, ColorLogicOpEnable, ColorMask, ColorMaterial, ColorMatrix, ColorOp, ColorTransform, ColorVertex

10.47 ColorWriteEnable2

NAME
ColorWriteEnable2 - 3D API color write enable2

USAGE
ColorWriteEnable2 = bool4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: true, false

DESCRIPTION
Enable/disable writing of frame buffer color components. See the D3DRS_COLORWRITEENABLE2 render state description for DirectX 9.

The standard reset callback sets the ColorWriteEnable2 state to bool4(true, true, true, true).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ColorWriteEnable, ColorWriteEnable1, ColorWriteEnable3, BlendColor, ClearColor, ColorArg0, ColorArg1, ColorArg2, ColorLogicOpEnable, ColorMask, ColorMaterial, ColorMatrix, ColorOp, ColorTransform, ColorVertex
10.48 ColorWriteEnable3

NAME

ColorWriteEnable3 - 3D API color write enable3

USAGE

ColorWriteEnable3 = bool4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: true, false

DESCRIPTION

Enable/disable writing of frame buffer color components. See the D3DRS_COLORWRITEENABLE3 render state description for DirectX 9.

The standard reset callback sets the ColorWriteEnable3 state to bool4(true, true, true, true).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ColorWriteEnable, ColorWriteEnable1, ColorWriteEnable2, BlendColor, ClearColor, ColorArg0, ColorArg1, ColorArg2, ColorLogicOpEnable, ColorMask, ColorMaterial, ColorMatrix, ColorOp, ColorTransform, ColorVertex

10.49 ColorWriteEnable

NAME

ColorWriteEnable - 3D API color write enable

USAGE

ColorWriteEnable = bool4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: true, false

DESCRIPTION

Enable/disable writing of frame buffer color components. See the OpenGL glColorMask manual page for details. See the D3DRS_COLORWRITEENABLE render state description for DirectX 9.

The standard reset callback sets the ColorWriteEnable state to bool4(true, true, true, true).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BlendColor,ClearColor, ColorArg0, ColorArg1, ColorArg2, ColorLogicOpEnable, ColorMask, ColorMaterial, ColorMatrix, ColorOp, ColorTransform, ColorVertex, ColorWriteEnable1, ColorWriteEnable2, ColorWriteEnable3

10.50 CullFaceEnable

NAME

CullFaceEnable - 3D API cull face enable

USAGE

CullFaceEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL face culling. See the GL_CULL_FACE description on the OpenGL glEnable manual page for details.

The standard reset callback sets the CullFaceEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

CullFace, CullMode, FrontFace

10.51 CullFace

NAME

CullFace - 3D API cull face

USAGE

CullFace = int(face)

VALID ENUMERANTS

face: Front, Back, FrontAndBack
DESCRIPTION
Specify orientation of candidates for facet culling. See the OpenGL glCullFace manual page for details.
The standard reset callback sets the CullFace state to int(Back).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
CullFaceEnable, CullMode, FrontFace

10.52 CullMode

NAME
CullMode - 3D API cull mode

USAGE
CullMode = int(face)

VALID ENUMERANTS
face: Front, Back, FrontAndBack

DESCRIPTION
Specify orientation of candidates for facet culling. See the OpenGL glCullFace manual page for details.
The standard reset callback sets the CullMode state to int(Back).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
CullFace, CullFaceEnable, FrontFace

10.53 DepthBias

NAME
DepthBias - 3D API depth bias

USAGE
DepthBias = float(bias)
VALID ENUMERANTS

bias: floating point value

DESCRIPTION

Specify the constant offset used to compute a depth offset for polygons. See the units parameter description on the OpenGL glPolygonOffset manual page for details. See the D3DRS_DEPTHBIAS render state description for DirectX 9.

The standard reset callback sets the DepthBias state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.1

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PolygonOffset, SlopScaleDepthBias

10.54 DepthBoundsEnable

NAME

DepthBoundsEnable - 3D API depth bounds enable

USAGE

DepthBoundsEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL depth bounds testing. See the OpenGL EXT_depth_bounds_test specification for details. The standard reset callback sets the DepthBoundsEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Support for extension EXT_depth_bounds_test.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthClampEnable, DepthFunc, DepthMask, DepthRange, DepthTestEnable, SlopScaleDepthBias
10.55 DepthBounds

NAME

DepthBounds - 3D API depth bounds

USAGE

DepthBounds = float2(zmin, zmax)

VALID ENUMERANTS

zmin, zmax: floating point values

DESCRIPTION

Specify the minimum and maximum values for depth bounds testing. See the OpenGL EXT_depth_bounds_test specification for details.

The standard reset callback sets the DepthBounds state to float2(0, 1).

OPENVGL FUNCTIONALITY REQUIREMENTS

Support for extension EXT_depth_bounds_test.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

DepthBoundsEnable

10.56 DepthClampEnable

NAME

DepthClampEnable - 3D API depth clamp enable

USAGE

DepthClampEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL depth clamping. See the OpenGL NV_depth_clamp specification for details.

The standard reset callback sets the DepthClampEnable state to bool(false).

OPENVGL FUNCTIONALITY REQUIREMENTS

Support for extension NV_depth_clamp.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.
SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthFunc, DepthMask, DepthRange, DepthTestEnable, SlopeScaleDepthBias

10.57 DepthFunc

NAME

DepthFunc - 3D API depth func

USAGE

DepthFunc = int(func)

VALID ENUMERANTS

func: Never, Less, LEqual, LessEqual, Equal, Greater, NotEqual, GEqual, GreaterEqual, Always

DESCRIPTION

Specify the function used for depth buffer comparisons. See the OpenGL glVertexFunc manual page for details. See the D3DRS_ZFUNC render state description for DirectX 9.

The standard reset callback sets the DepthFunc state to int(Less).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthTestEnable, DepthMask, DepthRange, DepthTestEnable, SlopeScaleDepthBias, ZEnable, ZFunc, ZWriteEnable

10.58 DepthMask

NAME

DepthMask - 3D API depth mask

USAGE

DepthMask = bool(enable)

VALID ENUMERANTS

enable: true, false
**DESCRIPTION**

Specify whether the depth buffer is enabled for writing. See the OpenGL glDepthMask manual page for details. See the D3DRS_ZWRITEENABLE render state description for DirectX 9.

The standard reset callback sets the DepthMask state to bool(true).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

*ColorMask, StencilMask*

### 10.59 DepthRange

**NAME**

*DepthRange* - 3D API depth range

**USAGE**

DepthRange = float2(near, far)

**VALID ENUMERANTS**

near, far: floating point values

**DESCRIPTION**

Specify the mapping of the near and far clipping planes to window coordinates. See the OpenGL glDepthRange manual page for details. See the IDirect3DDevice9::SetViewport method for details.

The standard reset callback sets the DepthRange state to float2(0, 1).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

*DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthFunc, DepthMask, DepthTestEnable*

### 10.60 DepthTestEnable

**NAME**

*DepthTestEnable* - 3D API depth test enable
USAGE

DepthTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable depth testing. See the GL_DEPTH_TEST description on the OpenGL glEnable manual page for details. See the D3DRS_ZENABLE render state description for DirectX 9.

The standard reset callback sets the DepthTestEnable state to bool(false).

OPengl FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthFunc, DepthMask, DepthRange, SlopScaleDepthBias, ZEnable, ZFunc, ZWriteEnable

10.61 DestBlendAlpha

NAME

DestBlendAlpha - 3D API destination alpha blending type

USAGE

DestBlendAlpha = int(blend)

VALID ENUMERANTS

blend: Zero, One, DestColor, InvDestColor, SrcAlpha, InvSrcAlpha, DstAlpha, InvDestAlpha, SrcAlphaSat, SrcColor, InvSrcColor, BlendFactor, InvBlendFactor

DESCRIPTION

Set the Direct3D separate alpha blending destination blend type. See the D3DRS_DESTBLENDALPHA render state description for DirectX 9.

The standard reset callback sets the DestBlendAlpha state to int(Zero).

OPengl FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9
SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, SeparateAlphaBlendEnable

10.62 DestBlend

NAME

DestBlend - 3D API dest blend

USAGE

DestBlend = int(dfactor)

VALID ENUMERANTS

dfactor: Zero, One, DestColor, InvDestColor, SrcAlpha, InvSrcAlpha, DstAlpha, InvDestAlpha, SrcAlphaSat, SrcColor, InvSrcColor, BlendFactor, InvBlendFactor

DESCRIPTION

Specify the destination blend factor. See the OpenGL glBlendFunc manual page for details. See the D3DRS_DESTBLENDALPHA render state description for DirectX 9.

The standard reset callback sets the DestBlend state to int(Zero).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, SeparateAlphaBlendEnable

10.63 DiffuseMaterialSource

NAME

DiffuseMaterialSource - 3D API diffuse material source

USAGE

DiffuseMaterialSource = int(enable)

VALID ENUMERANTS

enable: true, false
DESCRIPTION
Enable/disable front and back diffuse color material. See the GL_DIFFUSE description on the OpenGL glColorMaterial manual page for details. See the D3DRS_DIFFUSEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the DiffuseMaterialSource state to int(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.64 DitherEnable

NAME
DitherEnable - 3D API dither enable

USAGE
DitherEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable dithering. See the GL_DITHER description on the OpenGL glEnable manual page for details. See the D3DRS_DITHERENABLE render state description for DirectX 9.

The standard reset callback sets the DitherEnable state to bool(true).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearColor, ColorWriteEnable
10.65 EmissiveMaterialSource

NAME

EmissiveMaterialSource - 3D API emissive material source

USAGE

EmissiveMaterialSource = int(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable front and back emissive color material. See the GL_EMISSION description on the OpenGL glColorMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the EmissiveMaterialSource state to int(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.66 FillMode

NAME

FillMode - 3D API fill mode

USAGE

FillMode = int(mode)

VALID ENUMERANTS

mode: Solid, Wireframe, Point

DESCRIPTION

Specify the rasterization mode for front facing polygons. See the OpenGL glPolygonMode manual page for details. See the D3DRS_FILLMODE render state description for DirectX 9.

The standard reset callback sets the FillMode state to int(Solid).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

PolygonMode

10.67 FogColor

NAME
FogColor - 3D API fog color

USAGE
FogColor = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the fog color. See the GL_FOG_COLOR description on the OpenGL glFog manual page for details. See the
D3DRS_FOGCOLOR render state description for DirectX 9.

The standard reset callback sets the FogColor state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, FogVertex-
Mode, RangeFogEnable

10.68 FogCoordSrc

NAME
FogCoordSrc - 3D API fog coord src

USAGE
FogCoordSrc = int(src)

VALID ENUMERANTS
GL src: FragmentDepth, FogCoord D3D9 src: None, Exp, Exp2, Linear
DESCRIPTION

Specify whether to use fragment depth or a per-vertex fog coordinate in fog computations. See the GL_FOG_COORDINATE_SOURCE_EXT description in the OpenGL EXT_fog_coord specification for details. See the D3DRS_FOGVERTEXMODE render state description for DirectX 9.

The standard reset callback sets the FogCoordSrc state to int(FragmentDepth).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.4 or support for extension EXT_fog_coord.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, FogVertexMode, RangeFogEnable

10.69 FogDensity

NAME

FogDensity - 3D API fog density

USAGE

FogDensity = float(density)

VALID ENUMERANTS

density: floating point value

DESCRIPTION

Specify the fog density to use in exponential fog equations. See the GL_FOG_DENSITY description on the OpenGL glFog manual page for details. See the D3DRS_FOGDENSITY render state description for DirectX 9.

The standard reset callback sets the FogDensity state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, FogVertexMode, RangeFogEnable
10.70 FogDistanceMode

NAME
FogDistanceMode - 3D API fog distance mode

USAGE
FogDistanceMode = int(mode)

VALID ENUMERANTS
mode: EyeRadial, EyePlane, EyePlaneAbsolute

DESCRIPTION
Specify how OpenGL computes the distance used when computing the fog factor. See the
GL_FOG_DISTANCE_MODE_NV description the OpenGL OpenGL NV_fog_distance specification for details.
The standard reset callback sets the FogDistanceMode state to int(EyePlaneAbsolute).

OPENGL FUNCTIONALITY REQUIREMENTS
Support for extension NV_fog_distance.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
FogColor, FogCoordSrc, FogDensity, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, FogVertexMode,
RangeFogEnable

10.71 FogEnable

NAME
FogEnable - 3D API fog enable

USAGE
FogEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable fogging. See the GL_FOG description on the OpenGL glEnable manual page for details. See the
D3DRS_FOGENABLE render state description for DirectX 9.
The standard reset callback sets the FogEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnd, FogMode, FogStart, FogTableMode, FogVertexMode, RangeFogEnable

10.72 FogEnd

NAME

FogEnd - 3D API fog end

USAGE

FogEnd = float(end)

VALID ENUMERANTS

end: floating point value

DESCRIPTION

Specify the far distance used in the linear fog equation. See the GL_FOG_END description on the OpenGL glFog manual page for details. See the D3DRS_FOGEND render state description for DirectX 9.

The standard reset callback sets the FogEnd state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogMode, FogStart, FogTableMode, FogVertexMode, RangeFogEnable

10.73 FogMode

NAME

FogMode - 3D API fog mode

USAGE

FogMode = int(mode)

VALID ENUMERANTS

mode: Linear, Exp, Exp2
DESCRIPTION

Specify the equation used to compute the fog blend factor. See the GL_FOG_MODE description on the OpenGL glFog manual page for details. See the D3DRS_FOGTABLEMODE render state description for DirectX 9.

The standard reset callback sets the FogMode state to int(Exp).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogStart, FogTableMode, FogVertexMode, RangeFogEnable

10.74 FogStart

NAME

FogStart - 3D API fog start

USAGE

FogStart = float(start)

VALID ENUMERANTS

start: floating point value

DESCRIPTION

Specify the near distance used in the linear fog equation. See the GL_FOG_START description on the OpenGL glFog manual page for details. See the D3DRS_FOGSTART render state description for DirectX 9.

The standard reset callback sets the FogStart state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogTableMode, FogVertexMode, RangeFogEnable
10.75 FogTableMode

NAME

FogTableMode - 3D API fog table mode

USAGE

FogTableMode = int(mode)

VALID ENUMERANTS

mode: Linear, Exp, Exp2

DESCRIPTION

Specify the equation used to compute the fog blend factor. See the GL_FOG_MODE description on the OpenGL glFog manual page for details. See the D3DRS_FOGTABLEMODE render state description for DirectX 9.

The standard reset callback sets the FogTableMode state to int(Exp).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogVertexMode, RangeFogEnable

10.76 FogVertexMode

NAME

FogVertexMode - 3D API fog vertex mode

USAGE

FogVertexMode = int(mode)

VALID ENUMERANTS

mode: None, Exp, Exp2, Linear

DESCRIPTION

Specify whether to use the fog coordinate or fragment depth as distance value in the fog computation.

See the GL_FOG_COORDINATE_SOURCE description on the OpenGL glFog manual page for details. See the D3DRS_FOGVERTEXMODE render state description for DirectX 9.

The standard reset callback sets the FogVertexMode state to int(GL_FRAGMENT_DEPTH_EXT).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.4 or support for extension EXT_fog_coord.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, RangeFogEnable

10.77 FragmentEnvParameter

NAME

FragmentEnvParameter - 3D API fragment env parameter

USAGE

FragmentEnvParameter[i] = float4(x, y, z, w)

VALID ENUMERANTS

x, y, z, w: floating point values

DESCRIPTION

Set fragment program environment parameter. See the OpenGL glProgramEnvParameter4fvARB manual page for details.

The standard reset callback sets the FragmentEnvParameter[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter, PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4

10.78 FragmentLocalParameter

NAME

FragmentLocalParameter - 3D API fragment local parameter

USAGE

FragmentLocalParameter[i] = float4(x, y, z, w)
VALID ENUMERANTS

x, y, z, w: floating point values

DESCRIPTION

Set fragment program local parameter. See the OpenGL glProgramLocalParameter4fvARB manual page for details. The standard reset callback sets the FragmentLocalParameter[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

FragmentEnvParameter, VertexEnvParameter, VertexLocalParameter, PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4

10.79 FragmentProgram

NAME

FragmentProgram - 3D API fragment program

USAGE

FragmentProgram = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

prog: Null

DESCRIPTION

Compile a fragment program. See the specification of the OpenGL fragment profile for details. See the specification of the Direct3D fragment shaders for details. The standard reset callback sets the FragmentProgram state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.80 FrontFace

NAME

FrontFace - 3D API front face

USAGE

FrontFace = int(windingOrder)

VALID ENUMERANTS

windingOrder: CW, CCW

DESCRIPTION

Specifies the winding order for front-facing polygons. See the OpenGL glFrontFace manual page for details.
The standard reset callback sets the FrontFace state to int(CCW).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

CullFace, CullFaceEnable, CullMode

10.81 GeometryProgram

NAME

GeometryProgram - 3D API geometry program

USAGE

GeometryProgram = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

prog: Null

DESCRIPTION

Compile a geometry program. See the specification of the OpenGL geometry profile for details. See the specification
of the Direct3D 10 geometry shaders for details.
The standard reset callback sets the GeometryProgram state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 10
10.82 GeometryShader

NAME

GeometryShader - 3D API geometry shader

USAGE

GeometryShader = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

shader: Null

DESCRIPTION

Compile a geometry shader. See the specification of the OpenGL geometry profile for details. See the specification of the Direct3D 10 geometry shaders for details.

The standard reset callback sets the GeometryShader state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 10

SEE ALSO


10.83 IndexedVertexBlendEnable

NAME

IndexedVertexBlendEnable - 3D API indexed vertex blend enable

USAGE

IndexedVertexBlendEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

SEE ALSO

DESCRIPTION
Enable/disable Direct3D indexed vertex blending. See the D3DRS_INDEXEDVERTEXBLENDENABLE render state description for DirectX 9.

The standard reset callback sets the IndexedVertexBlendEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, SrcBlend, VertexBlend, BlendOpAlpha, SrcBlendAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.84 LastPixel

NAME
LastPixel - 3D API last pixel

USAGE
LastPixel = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable drawing of the last pixel in a line. See the D3DRS_LASTPIXEL render state description for DirectX 9.

The standard reset callback sets the LastPixel state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
FillMode, PolygonMode, DitherEnable
10.85 LightAmbient

NAME

**LightAmbient** - 3D API light ambient

USAGE

LightAmbient[i] = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Specify the ambient intensity of the light. See the GL_AMBIENT description on the OpenGL glLight manual page for details. See the Ambient member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightAmbient[i] state to float4(0, 0, 0, 1).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.86 LightAttenuation0

NAME

**LightAttenuation0** - 3D API light attenuation0

USAGE

LightAttenuation0[i] = float(constant_attenuation)

VALID ENUMERANTS

constant_attenuation: floating point value

DESCRIPTION

Specify the constant light attenuation factor. See the GL_CONSTANT_ATTENUATION description on the OpenGL glLight manual page for details. See the Attenuation0 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightAttenuation0[i] state to float(1).
OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.87 LightAttenuation1

NAME
LightAttenuation1 - 3D API light attenuation1

USAGE
LightAttenuation1[i] = float(linear_attenuation)

VALID ENUMERANTS
linear_attenuation: floating point value

DESCRIPTION
Specify the linear light attenuation factor. See the GL_LINEAR_ATTENUATION description on the OpenGL glLight manual page for details. See the Attenuation1 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightAttenuation1[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.88 LightAttenuation2

NAME
LightAttenuation2 - 3D API light attenuation2
USAGE
LightAttenuation2[i] = float(quadratic_attenuation)

VALID ENUMERANTS
quadratic_attenuation: floating point value

DESCRIPTION
Specify the quadratic light attenuation factor. See the GL_QUADRATIC_ATTENUATION description on the OpenGL glLight manual page for details. See the Attenuation2 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightAttenuation2[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
LightAmbient, LightAttenuation0, LightAttenuation1, LightConstantAttenuation, LightDiffuse, LightDirection, LightEnable, Light Falloff, LightLinearAttenuation, LightModelAmbient, LightModelColorControl, LightModelLocalViewerEnable, LightModelTwoSideEnable, LightPhi, LightPosition, LightQuadraticAttenuation, LightRange, LightSpecular, LightSpotCutoff, LightSpotDirection, LightSpotExponent, LightTheta, LightType, LightingEnable

10.89 LightConstantAttenuation

NAME
LightConstantAttenuation - 3D API light constant attenuation

USAGE
LightConstantAttenuation[i] = float(constant_attenuation)

VALID ENUMERANTS
constant_attenuation: floating point value

DESCRIPTION
Specify the constant light attenuation factor. See the GL_CONSTANT_ATTENUATION description on the OpenGL glLight manual page for details. See the Attenuation0 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightConstantAttenuation[i] state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9
10.90 LightDiffuse

NAME
LightDiffuse - 3D API light diffuse

USAGE
LightDiffuse[i] = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the diffuse intensity of the light. See the GL_DIFFUSE description on the OpenGL glLight manual page for details. See the Diffuse member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightDiffuse[i] state to float4(1, 1, 1, 1) when i is 0 and float4(0, 0, 0, 1) otherwise.

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.91 LightDirection

NAME
LightDirection - 3D API light direction

USAGE
LightDirection[i] = float4(x, y, z, w)

VALID ENUMERANTS
x, y, z, w: floating point values
DESCRIPTION
Specify the lights direction. See the GL_SPOT_DIRECTION description on the OpenGL glLight manual page for details. See the Direction member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightDirection[i] state to float4(0, 0, -1, 1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
LightAmbient, LightAttenuation0, LightAttenuation1, LightAttenuation2, LightConstantAttenuation, LightDiffuse, LightEnable, LightFalloff, LightLinearAttenuation, LightModelAmbient, LightModelColorControl, LightModelLocalViewerEnable, LightModelTwoSideEnable, LightPhi, LightPosition, LightQuadraticAttenuation, LightRange, LightSpecular, LightSpotCutoff, LightSpotDirection, LightSpotExponent, LightTheta, LightType, LightingEnable

10.92 LightEnable

NAME
LightEnable - 3D API light enable

USAGE
LightEnable[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable a light for use in the lighting computations. See the GL_LIGHTi description on the OpenGL glEnable manual page for details. See the IDirect3DDevice9::SetLight method for details.

The standard reset callback sets the LightEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
10.93 LightFalloff

NAME

LightFalloff - 3D API light falloff

USAGE

LightFalloff[i] = float(angle)

VALID ENUMERANTS

angle: floating point value

DESCRIPTION

Specify the light's maximum spread angle. See the GL_SPOT_CUTOFF description on the OpenGL glLight manual page for details. See the Falloff member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightFalloff[i] state to float(180).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.94 LightingEnable

NAME

LightingEnable - 3D API lighting enable

USAGE

LightingEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable lighting. See the GL_LIGHTING description on the OpenGL glEnable manual page for details. See the D3DRS_LIGHTING render state description for DirectX 9.

The standard reset callback sets the LightingEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
10.95 Lighting

NAME

Lighting - 3D API light enable

USAGE

Lighting = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable a light for use in the lighting computations. See the IDirect3DDevice9::SetLight method for details. The standard reset callback sets the Lighting state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.96 LightLinearAttenuation

NAME

LightLinearAttenuation - 3D API light linear attenuation

USAGE

LightLinearAttenuation[i] = float(linear_attenuation)
VALID ENUMERANTS

linear_attenuation: floating point value

DESCRIPTION

Specify the linear light attenuation factor. See the GL_LINEAR_ATTENUATION description on the OpenGL glLight
manual page for details. See the Attenuation1 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight
method.

The standard reset callback sets the LightLinearAttenuation[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightAmbient, LightAttenuation0, LightAttenuation1, LightAttenuation2, LightConstantAttenuation, LightDiffuse,
LightDirection, LightEnable, LightFalloff, LightModelAmbient, LightModelColorControl, LightModelLocalViewer-
Enable, LightModelTwoSideEnable, LightPhi, LightPosition, LightQuadraticAttenuation, LightRange, LightSpecular,
LightSpotCutoff, LightSpotDirection, LightSpotExponent, LightTheta, LightType, LightingEnable

10.97 LightModelAmbient

NAME

LightModelAmbient - 3D API light model ambient

USAGE

LightModelAmbient = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Specify the ambient intensity of the entire scene. See the GL_LIGHT_MODEL_AMBIENT description on the OpenGL glLightModel manual page for details. See the D3DRS_AMBIENT render state description for DirectX 9.

The standard reset callback sets the LightModelAmbient state to float4(0.2, 0.2, 0.2, 1.0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9
SEE ALSO


10.98 LightModelColorControl

NAME

LightModelColorControl - 3D API light model color control

USAGE

```
LightModelColorControl = int(control)
```

VALID ENUMERANTS

color: SingleColor, SeparateSpecular

DESCRIPTION

Specify whether a separate specular color is generated during the lighting computations. See the GL_LIGHT_MODEL_COLOR_CONTROL description on the OpenGL glLightModel manual page for details.

The standard reset callback sets the LightModelColorControl state to int(SingleColor).

OPENVGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.2 or support for extension EXT_separate_specular_color.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO


10.99 LightModelLocalViewerEnable

NAME

LightModelLocalViewerEnable - 3D API light model local viewer enable

USAGE

```
LightModelLocalViewerEnable = bool(enable)
```

VALID ENUMERANTS

enable: true, false
DESCRIPTION

Enable/disable the OpenGL local viewer lighting model. See the GL_LIGHT_MODEL_LOCAL_VIEWER description on the OpenGL glLightModel manual page for details.

The standard reset callback sets the LightModelLocalViewerEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO


10.100 LightModelTwoSideEnable

NAME

LightModelTwoSideEnable - 3D API light model two side enable

USAGE

LightModelTwoSideEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL two-sided lighting. See the GL_LIGHT_MODEL_TWO_SIDE description on the OpenGL glLightModel manual page for details.

The standard reset callback sets the LightModelTwoSideEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

10.101 LightPhi

NAME
LightPhi - 3D API light phi

USAGE
LightPhi[i] = float(phi)

VALID ENUMERANTS
phi: floating point value

DESCRIPTION
Specify the outer edge of the spotlight’s outer cone. See the Phi member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightPhi[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.102 LightPosition

NAME
LightPosition - 3D API light position

USAGE
LightPosition[i] = float4(x, y, z, w)

VALID ENUMERANTS
x, y, z, w: floating point values

DESCRIPTION
Specify the lights position in object coordinates. See the GL_POSITION description on the OpenGL glLight manual page for details. See the Position member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightPosition[i] state to float4(0, 0, 1, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
10.103 **LightQuadraticAttenuation**

**NAME**

*LightQuadraticAttenuation* - 3D API light quadratic attenuation

**USAGE**

`LightQuadraticAttenuation[i] = float(quadratic_attenuation)`

**VALID ENUMERANTS**

quadratic_attenuation: floating point value

**DESCRIPTION**

Specify the quadric light attenuation factor. See the GL_QUADRATIC_ATTENUATION description on the OpenGL glLight manual page for details. See the Attenuation2 member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the `LightQuadraticAttenuation[i]` state to float(0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

`LightAmbient, LightAttenuation0, LightAttenuation1, LightAttenuation2, LightConstantAttenuation, LightDiffuse, LightDirection, LightEnable, LightFalloff, LightLinearAttenuation, LightModelAmbient, LightModelColorControl, LightModelLocalViewerEnable, LightModelTwoSideEnable, LightPhi, LightQuadraticAttenuation, LightRange, LightSpecular, LightSpotCutoff, LightSpotDirection, LightSpotExponent, LightTheta, LightType, LightingEnable`
VALID ENUMERANTS
range: floating point value

DESCRIPTION
Specify the maximum distance for which a light has any effect. See the Range member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.
The standard reset callback sets the LightRange[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
LightAmbient, LightAttenuation0, LightAttenuation1, LightAttenuation2, LightConstantAttenuation, LightDiffuse,
LightDirection, LightEnable, LightFalloff, LightLinearAttenuation, LightModelAmbient, LightModelColorControl,
LightModelLocalViewerEnable, LightModelTwoSideEnable, LightPhi, LightPosition, LightQuadraticAttenuation,
LightSpecular, LightSpotCutoff, LightSpotDirection, LightSpotExponent, LightTheta, LightType, LightingEnable

10.105 LightSpecular

NAME
LightSpecular - 3D API light specular

USAGE
LightSpecular[i] = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the specular intensity of the light. See the GL_SPECULAR description on the OpenGL glLight manual page
for details. See the Specular member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.
The standard reset callback sets the LightSpecular[i] state to float4(1, 1, 1, 1) when i is 0 and float4(0, 0, 0, 1)
otherwise.

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9
10.106 LightSpotCutoff

NAME
LightSpotCutoff - 3D API light spot cutoff

USAGE
LightSpotCutoff[i] = float(cutoff)

VALID ENUMERANTS

cutoff: floating point value

DESCRIPTION
Specify the lights maximum spread angle. See the GL_SPOT_CUTOFF description on the OpenGL glLight manual page for details. See the Falloff member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightSpotCutoff[i] state to float(180).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.107 LightSpotDirection

NAME
LightSpotDirection - 3D API light spot direction

USAGE
LightSpotDirection[i] = float4(x, y, z, w)

VALID ENUMERANTS

x, y, z, w: floating point values
DESCRIPTION

Specify the lights direction. See the GL_SPOT_DIRECTION description on the OpenGL gLight manual page for details. See the Direction member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightSpotDirection[i] state to float4(0, 0, -1, 1).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.108 LightSpotExponent

NAME

LightSpotExponent - 3D API light spot exponent

USAGE

LightSpotExponent[i] = float(exp)

VALID ENUMERANTS

exp: floating point value

DESCRIPTION

Specify the lights intensity distribution. See the GL_SPOT_EXPONENT description on the OpenGL gLight manual page for details.

The standard reset callback sets the LightSpotExponent[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

10.109 LightTheta

NAME

LightTheta - 3D API light theta

USAGE

LightTheta[i] = float(theta)

VALID ENUMERANTS

theta: floating point value

DESCRIPTION

Specify the angle in radians of a spotlight’s inner cone. See the Theta member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightTheta[i] state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.110 LightType

NAME

LightType - 3D API light type

USAGE

LightType[i] = int(type)

VALID ENUMERANTS

type: integer value

DESCRIPTION

Specify the type of light source. See the Type member of the D3DLIGHT9 argument to IDirect3DDevice9::SetLight method.

The standard reset callback sets the LightType[i] state to int(Point).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.111 LineSmoothEnable

NAME

LineSmoothEnable - 3D API line smooth enable

USAGE

LineSmoothEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable antialiased line rendering. See the GL_LINE_SMOOTH description on the OpenGL glEnable manual page for details. See the D3DRS_ANTIALIASEDLINEENABLE render state description for DirectX 9.

The standard reset callback sets the LineSmoothEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PointSmoothEnable, PolygonSmoothEnable, LineStipple, LineStippleEnable, LineWidth

10.112 LineStippleEnable

NAME

LineStippleEnable - 3D API line stipple enable

USAGE

LineStippleEnable = bool(enable)

VALID ENUMERANTS

enable: true, false
DESCRIPTION
Enable/disable OpenGL line stippling. See the GL_LINE_STIPPLE description on the OpenGL glEnable manual page for details.
The standard reset callback sets the LineStippleEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LineSmoothEnable, LineStipple, LineWidth, PolygonStippleEnable

10.113 LineStipple

NAME
LineStipple - 3D API line stipple

USAGE
LineStipple = int2(factor, pattern)

VALID ENUMERANTS
factor: integer value pattern: integer value

DESCRIPTION
Specify the line stippling factor and pattern. See the OpenGL glLineStipple manual page for details.
The standard reset callback sets the LineStipple state to int2(1, ~0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LineStippleEnable, PolygonStippleEnable

10.114 LineWidth

NAME
LineWidth - 3D API line width
USAGE
LineWidth = float(width)

VALID ENUMERANTS
width: floating point value

DESCRIPTION
Specify the width of rasterized lines. See the OpenGL glLineWidth manual page for details.
The standard reset callback sets the LineWidth state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PointSize, PointSizeMax, PointSizeMin

10.115 LocalViewer

NAME
LocalViewer - 3D API local viewer

USAGE
LocalViewer = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable computation of specular reflections using the local viewer model. See the
GL_LIGHT_MODEL_LOCAL_VIEWER description on the OpenGL glLightModel manual page for details.
See the D3DRS_LOCALVIEWER render state description for DirectX 9.
The standard reset callback sets the LocalViewer state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
LightModelLocalViewerEnable
10.116 LogicOpEnable

NAME

LogicOpEnable - 3D API logic op enable

USAGE

LogicOpEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL color logical operations. See the GL_COLOR_LOGIC_OP description on the OpenGL glEnable manual page for details.

The standard reset callback sets the LogicOpEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

ColorLogicOpEnable, LogicOp

10.117 LogicOp

NAME

LogicOp - 3D API logic op

USAGE

LogicOp = int(op)

VALID ENUMERANTS

op: Clear, And, AndReverse, Copy, AndInverted, Noop, Xor, Or, Nor, Equiv, Invert, OrReverse, CopyInverted, Nand, Set

DESCRIPTION

Specify a logical pixel operation for fragment color and color buffer values. See the OpenGL glLogicOp manual page for details.

The standard reset callback sets the LogicOp state to int(Copy).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

ColorLogicOpEnable, LogicOpEnable

10.118 MagFilteri

NAME

MagFilteri - 3D API mag filter

USAGE

MagFilteri[i] = int(type)

VALID ENUMERANTS

type: Point, Nearest, Linear

DESCRIPTION

Set the Direct3D sampler magnification filter type for texture unit i, where i is in the range 0-15. See sampler state MagFilter for details.

The standard reset callback sets the MagFilter[i] state to int(Point).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MagFilter, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.119 MaterialAmbient

NAME

MaterialAmbient - 3D API material ambient

USAGE

MaterialAmbient = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values
DESCRIPTION

Specify the materials ambient reflectance values. See the GL_AMBIENT description on the OpenGL glMaterial manual page for details. See the D3DRS_AMBIENTMATERIALSOURCE render state description for DirectX 9. The standard reset callback sets the MaterialAmbient state to float4(0.2, 0.2, 0.2, 1.0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.120 MaterialDiffuse

NAME

MaterialDiffuse - 3D API material diffuse

USAGE

MaterialDiffuse = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Specify the materials diffuse reflectance values. See the GL_DIFFUSE description on the OpenGL glMaterial manual page for details. See the D3DRS_DIFFUSEMATERIALSOURCE render state description for DirectX 9. The standard reset callback sets the MaterialDiffuse state to float4(0.8, 0.8, 0.8, 1.0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.121 MaterialEmission

NAME
MaterialEmission - 3D API material emission

USAGE
MaterialEmission = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the materials emissive light intensity. See the GL_EMISSION description on the OpenGL glMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the MaterialEmission state to float4(0, 0, 0, 1).

OpenGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.122 MaterialEmissive

NAME
MaterialEmissive - 3D API material emissive

USAGE
MaterialEmissive = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the materials emissive light intensity. See the GL_EMISSION description on the OpenGL glMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the MaterialEmissive state to float4(0, 0, 0, 1).

OpenGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.123 MaterialPower

NAME

MaterialPower - 3D API material power

USAGE

MaterialPower = float(shininess)

VALID ENUMERANTS

shininess: floating point value

DESCRIPTION

Specifies the materials specular exponent. See the GL_SHININESS description on the OpenGL glMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the MaterialPower state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.124 MaterialShininess

NAME

MaterialShininess - 3D API material shininess

USAGE

MaterialShininess = float(shininess)

VALID ENUMERANTS

shininess: floating point value
DESCRIPTION
Specifies the materials specular exponent. See the GL_SHININESS description on the OpenGL glMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the MaterialShininess state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.125 MaterialSpecular

NAME
MaterialSpecular - 3D API material specular

USAGE
MaterialSpecular = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the materials specular reflectance values. See the GL_SPECULAR description on the OpenGL glMaterial manual page for details. See the D3DRS_EMISSIVEMATERIALSOURCE render state description for DirectX 9.

The standard reset callback sets the MaterialSpecular state to float4(0, 0, 0, 1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
10.126  **MaxAnisotropy**

**NAME**

**MaxAnisotropy** - 3D API max anisotropy

**USAGE**

MaxAnisotropy[i] = int(max)

**VALID ENUMERANTS**

max: integer value

**DESCRIPTION**

Set the Direct3D maximum anisotropy value for texture unit i, where i is in the range 0-15. See sampler state *MaxAnisotropy* for details.

The standard reset callback sets the MaxAnisotropy[i] state to int(1).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

*MaxAnisotropy*

10.127  **MaxMipLevel**

**NAME**

**MaxMipLevel** - 3D API max mip level

**USAGE**

MaxMipLevel[i] = int(max)

**VALID ENUMERANTS**

max: integer value

**DESCRIPTION**

Set the Direct3D maximum mipmap lod index for texture unit i, where i is in the range 0-15. See sampler state *MaxMipLevel* for details.

The standard reset callback sets the MaxMipLevel[i] state to int(0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

SEE ALSO

MaxMipLevel, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MinFilter, MipFilter, MipMapLodBias

10.128 MinFilter

10.128 MinFilter

NAME

MinFilter - 3D API min filter

USAGE

MinFilter[i] = int(type)

VALID ENUMERANTS

type: Anisotropic, GaussianQuad, Linear, LinearMipMapLinear, LinearMipMapNearest, Nearest, NearestMipMapLinear, NearestMipMapNearest, None, Point, PyramidalQuad

DESCRIPTION

Set the Direct3D sampler minification filter type for texture unit i, where i is in the range 0-15. See sampler state MinFilter for details.

The standard reset callback sets the MinFilter[i] state to int(Point).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MinFilter, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MipFilter, MipMapLodBias

10.129 MipFilter

10.129 MipFilter

NAME

MipFilter - 3D API mip filter

USAGE

MipFilter[i] = int(type)
VALID ENUMERANTS

type: None, Point, Linear, Anisotropic, PyramidalQuad, GaussianQuad

DESCRIPTION

Set the Direct3D minification mipmap filter for texture unit \(i\), where \(i\) is in the range 0-15. See sampler state \texttt{MipFilter}\ for details.

The standard reset callback sets the MipFilter\([i]\) state to \texttt{int(None)}.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

\texttt{MipFilter, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MinFilter, MipMapLodBias}

### 10.130 MipMapLodBiasi

NAME

\texttt{MipMapLodBiasi} - 3D API mip map lod bias

USAGE

\texttt{MipMapLodBias}[i] = float(bias)

VALID ENUMERANTS

bias: floating point value

DESCRIPTION

Set the Direct3D mipmap lod bias. See the D3DSAMP_MIPMAPLODBIAS sampler state description for DirectX 9.

The standard reset callback sets the MipMapLodBias\([i]\) state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

\texttt{MipMapLodBias, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MinFilter, MipFilter}
10.131 ModelViewMatrix

NAME

ModelViewMatrix - 3D API model view matrix

USAGE

ModelViewMatrix = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the modelview matrix. See the GL_MODELVIEW description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_WORLD and D3DTS_VIEW transform state types for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the ModelViewMatrix state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ModelViewTransform, ProjectionMatrix, ProjectionTransform, ViewTransform, WorldTransform

10.132 ModelViewTransform

NAME

ModelViewTransform - 3D API model view transform

USAGE

ModelViewTransform = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the modelview matrix. See the GL_MODELVIEW description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_WORLD and D3DTS_VIEW transform state types for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the ModelViewTransform state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ModelViewMatrix, ProjectionMatrix, ProjectionTransform, ViewTransform, WorldTransform

10.133 MultiSampleAntialias

NAME
MultiSampleAntialias - 3D API multi sample antialias

USAGE
MultiSampleAntialias = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable using multiple fragment samples to compute the final pixel color. See the GL_MULTISAMPLE description on the OpenGL glEnable manual page for details. See the D3DRS_MULTISAMPLEANTIALIAS render state description for DirectX 9.

The standard reset callback sets the MultiSampleAntialias state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.3 or support for extension ARB_multisample.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
MultiSampleMask, MultisampleEnable, SampleAlphaToOneEnable, SampleCoverageEnable

10.134 MultisampleEnable

NAME
MultisampleEnable - 3D API multisample enable

USAGE
MultisampleEnable = bool(enable)

VALID ENUMERANTS
enable: true, false
DESCRIPTION

Enable/disable multisampling for antialiasing. See the GL_MULTISAMPLE_ARB description on the OpenGL glEnable manual page for details. See the D3DRS_MULTISAMPLEANTIALIAS render state description for DirectX 9.

The standard reset callback sets the MultisampleEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.3 or support for extension ARB_multisample.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MultiSampleAntialias, MultiSampleMask, SampleAlphaToCoverageEnable, SampleAlphaToOneEnable, SampleCoverageEnable

10.135 MultiSampleMask

NAME

MultiSampleMask - 3D API multisample mask

USAGE

MultiSampleMask = int(mask)

VALID ENUMERANTS

mask: integer value

DESCRIPTION

Specify a bit mask for multisample render targets. See the D3DRS_MULTISAMPLEMASK render state description for DirectX 9.

The standard reset callback sets the MultiSampleMask state to int(~0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MultiSampleAntialias, MultiSampleEnable, SampleAlphaToOneEnable, SampleCoverageEnable
10.136 NormalizeEnable

NAME

NormalizeEnable - 3D API normalize enable

USAGE

NormalizeEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable automatic normalization of vertex normals. See the GL_NORMALIZE description on the OpenGL glEnable manual page for details. See the D3DRS_NORMALIZENORMALS render state description for DirectX 9.

The standard reset callback sets the NormalizeEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AutoNormalEnable, RescaleNormalEnable, NormalizeNormals

10.137 NormalizeNormals

NAME

NormalizeNormals - 3D API automatic normalization of normals

USAGE

NormalizeNormals = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable automatic normalization of vertex normals. See the D3DRS_NORMALIZENORMALS render state description for DirectX 9.

The standard reset callback sets the NormalizeNormals state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AutoNormalEnable, NormalizeEnable, RescaleNormalEnable

10.138  PatchSegments

NAME

PatchSegments - 3D API patch segments

USAGE

PatchSegments = float(mode)

VALID ENUMERANTS

mode: floating point value

DESCRIPTION

Enable/disable N-patches. See the IDirect3DDevice9::SetNPatchMode method for details.
The standard reset callback sets the PatchSegments state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.139  PixelShaderConstant1

NAME

PixelShaderConstant1 - 3D API pixel shader constant1

USAGE

PixelShaderConstant1[i] = float4(f1, f2, f3, f4)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantF method for details.
The standard reset callback does nothing.
Cg 3.1 Reference Manual, Release 3.1

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.140 PixelShaderConstant2

NAME
PixelShaderConstant2 - 3D API pixel shader constant2

USAGE
PixelShaderConstant2[i] = float4x2(f1, f2, f3, f4, f5, f6, f7, f8)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set a Direct3D pixel shader constant. See the IDirect3DDevice9:SetPixelShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.141 PixelShaderConstant3

NAME
PixelShaderConstant3 - 3D API pixel shader constant3
USAGE

PixelShaderConstant3[i] = float4x3(f1, f2, f3, ..., f9, f10, f11, f12)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstant1, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstant1, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.142 PixelShaderConstant4

NAME

PixelShaderConstant4 - 3D API pixel shader constant4

USAGE

PixelShaderConstant4[i] = float4x4(f1, f2, f3, f4, ..., f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9
10.143 **PixelShaderConstantB**

**NAME**

PixelShaderConstantB - 3D API pixel shader constantB

**USAGE**

PixelShaderConstantB[i] = bool4(b1, b2, b3, b4)

**VALID ENUMERANTS**

bi: true, false

**DESCRIPTION**

Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantB method for details.

The standard reset callback does nothing.

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.144 **PixelShaderConstantF**

**NAME**

PixelShaderConstantF - 3D API pixel shader constantF

**USAGE**

PixelShaderConstantF[i] = float4(f1, f2, f3, f4)

**VALID ENUMERANTS**

fi: floating point values
DESCRIPTION
Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.145 PixelShaderConstantI

NAME
PixelShaderConstantI - 3D API pixel shader constantI

USAGE
PixelShaderConstantI[i] = int4(i1, i2, i3, i4)

VALID ENUMERANTS
ii: integer values

DESCRIPTION
Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantI method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter
10.146 PixelShaderConstant

NAME

PixelShaderConstant - 3D API pixel shader constant

USAGE

PixelShaderConstant[i] = float4x4(f1, f2, f3, f4, ..., f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D pixel shader constant. See the IDirect3DDevice9::SetPixelShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderCon-
stantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderCon-
tant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertex-
ShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.147 PixelShader

NAME

PixelShader - 3D API pixel shader

USAGE

PixelShader = compile profile “-po profileOption” main(args);

VALID ENUMERANTS

shader: Null

DESCRIPTION

Compile a pixel shader. See the specification of the OpenGL fragment profile for details. See the specification of the
Direct3D pixel shaders for details.
The standard reset callback sets the PixelShader state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.148 PointDistanceAttenuation

NAME

PointDistanceAttenuation - 3D API point distance attenuation

USAGE

PointDistanceAttenuation = float3(x, y, z)

VALID ENUMERANTS

x, y, z: floating point values

DESCRIPTION

Specify the coefficients used to scale the computed point size. See the GL_POINT_DISTANCE_ATTENUATION description on the OpenGL glPointParameter manual page for details.

The standard reset callback sets the PointDistanceAttenuation state to float3(1, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

PointFadeThresholdSize

10.149 PointFadeThresholdSize

NAME

PointFadeThresholdSize - 3D API point fade threshold size

USAGE

PointFadeThresholdSize = float(size)

VALID ENUMERANTS

size: floating point value
DESCRIPTION
Specify the threshold to which point sizes are clamped. See the GL_POINT_FADE_THRESHOLD_SIZE description on the OpenGL glPointParameter manual page for details.
The standard reset callback sets the PointFadeThresholdSize state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PointDistanceAttenuation

10.150  PointScale_A

NAME
PointScale_A - 3D API point scale A

USAGE
PointScale_A = float(size)

VALID ENUMERANTS
size: floating point value

DESCRIPTION
Specify a distance-based size attenuation value for point primitives. See the D3DRS_POINTSCALE_A render state description for DirectX 9.
The standard reset callback sets the PointScale_A state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PointScaleEnable, PointScale_B, PointScale_C, PointSizeMax, PointSizeMin, PointSize, VertexProgramPointSizeEnable

10.151  PointScale_B

NAME
PointScale_B - 3D API point scale B
USAGE
PointScale_B = float(size)

VALID ENUMERANTS
size: floating point value

DESCRIPTION
Specify a distance-based size attenuation value for point primitives. See the D3DRS_POINTSCALE_B render state description for DirectX 9.

The standard reset callback sets the PointScale_B state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PointSizeMax, PointSizeMin, PointSize

10.152 PointScale_C

NAME
PointScale_C - 3D API point scale C

USAGE
PointScale_C = float(size)

VALID ENUMERANTS
size: floating point value

DESCRIPTION
Specify a distance-based size attenuation value for point primitives. See the D3DRS_POINTSCALE_C render state description for DirectX 9.

The standard reset callback sets the PointScale_C state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PointSizeMax, PointSizeMin, PointSize
10.153 PointScaleEnable

NAME

PointScaleEnable - 3D API point scale enable

USAGE

PointScaleEnable = bool(enable)

VALID ENUMERANTS

disable: true, false

DESCRIPTION

Enable/disable Direct3D camera space point scaling. See the D3DRS_POINTSCALEENABLE render state description for DirectX 9.

The standard reset callback sets the PointScaleEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PointSizeMax, PointScale_A, PointScale_B, PointScale_C, PointSizeMin, PointSize, VertexProgramPointSizeEnable

10.154 PointSizeMax

NAME

PointSizeMax - 3D API point size max

USAGE

PointSizeMax = float(max)

VALID ENUMERANTS

max: floating point value

DESCRIPTION

Specify the maximum point size. See the GL_POINT_SIZE_MAX description on the OpenGL glPointParameter manual page for details. See the D3DRS_POINTSIZEMAX render state description for DirectX 9.

The standard reset callback sets the PointSizeMax state to the maximum value returned by glGet for parameter value GL_SMOOTH_POINT_SIZE_RANGE.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

PointSizeEnable, PointScale_A, PointScale_B, PointScale_C, PointSizeMin, PointSize, VertexProgramPointSizeEnable

10.155 PointSizeMin

NAME

PointSizeMin - 3D API point size min

USAGE

PointSizeMin = float(min)

VALID ENUMERANTS

min: floating point value

DESCRIPTION

Specify the minimum point size. See the GL_POINT_SIZE_MIN description on the OpenGL glPointParameter manual page for details. See the D3DRS_POINTSIZE_MIN render state description for DirectX 9.

The standard reset callback sets the PointSizeMin state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PointSizeEnable, PointScale_A, PointScale_B, PointScale_C, PointSizeMax, PointSize, VertexProgramPointSizeEnable

10.156 PointSize

NAME

PointSize - 3D API point size

USAGE

PointSize = float(size)

VALID ENUMERANTS

size: floating point value
DESCRIPTION
Specify the diameter of rasterized points. See the OpenGL glPointSize manual page for details. See the D3DRS_POINTSIZE render state description for DirectX 9.
The standard reset callback sets the PointSize state to float(1).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PointSizeMax, PointSizeMin, VertexProgramPointSizeEnable

10.157 PointSmoothEnable

NAME
PointSmoothEnable - 3D API point smooth enable

USAGE
PointSmoothEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL antialiased point rendering. See the GL_POINT_SMOOTH description on the OpenGL glEnable manual page for details.
The standard reset callback sets the PointSmoothEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LineSmoothEnable, PolygonSmoothEnable

10.158 PointSpriteCoordOrigin

NAME
PointSpriteCoordOrigin - 3D API point sprite coordinate origin
USAGE
PointSpriteCoordOrigin = int(origin)

VALID ENUMERANTS
origin: LowerLeft, UpperLeft

DESCRIPTION
Specify the texture coordinate origin for point sprites. See the GL_POINT_SPRITE_COORD_ORIGIN description on the OpenGL glPointParameter manual page for details.
The standard reset callback sets the PointSpriteCoordOrigin state to int(UpperLeft).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 2.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PointSpriteCoordReplace, PointSpriteEnable, PointSpriteRMode

10.159 PointSpriteCoordReplace

NAME
PointSpriteCoordReplace - 3D API point sprite coordinate replace

USAGE
PointSpriteCoordReplace[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Replace texture coordinates with point sprite texture coordinates. See the description of GL_COORD_REPLACE_ARB in the OpenGL ARB_point_sprite specification for details.
The standard reset callback sets the PointSpriteCoordReplace[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Support for extension ARB_point_sprite.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PointSpriteCoordOrigin, PointSpriteEnable, PointSpriteRMode
10.160 PointSpriteEnable

NAME

PointSpriteEnable - 3D API point sprite enable

USAGE

PointSpriteEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable texture coordinate computation for points. See the GL_POINT_SPRITE_ARB description on the OpenGL glEnable manual page for details. See the D3DRS_POINTSPRITEENABLE render state description for DirectX 9.

The standard reset callback sets the PointSpriteEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Support for extension ARB_point_sprite.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PointSpriteCoordOrigin, PointSpriteCoordReplace, PointSpriteRMode

10.161 PointSpriteRMode

NAME

PointSpriteRMode - 3D API point sprite R mode

USAGE

PointSpriteRMode = int(mode)

VALID ENUMERANTS

mode: Zero, S, R

DESCRIPTION

Specify the point sprite R coordinate mode. See the description of GL_POINT_SPRITE_R_MODE_NV in the OpenGL NV_point_sprite specification for details.

The standard reset callback sets the PointSpriteRMode state to int(Zero).

OPENGL FUNCTIONALITY REQUIREMENTS

Support for extension NV_point_sprite.
DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PointSpriteCoordOrigin, PointSpriteCoordReplace, PointSpriteEnable

10.162 PolygonMode

NAME
PolygonMode - 3D API polygon mode

USAGE
PolygonMode = int2(face, mode)

VALID ENUMERANTS
face: Front, Back, FrontAndBack mode: Point, Line, Fill, Solid, Wireframe

DESCRIPTION
Specify the polygon face rasterization mode. See the OpenGL glPolygonMode manual page for details. See the D3DRS_FILLMODE render state description for DirectX 9.

The standard reset callback sets the PolygonMode state to int2(FrontAndBack, Fill).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
FillMode

10.163 PolygonOffsetFillEnable

NAME
PolygonOffsetFillEnable - 3D API polygon offset fill enable

USAGE
PolygonOffsetFillEnable = bool(enable)

VALID ENUMERANTS
enable: true, false
DESCRIPTION
Enable/disable OpenGL depth offsetting for polygons rendered as filled primitives. See the GL_POLYGON_OFFSET_FILL description on the OpenGL glEnable manual page for details.

The standard reset callback sets the PolygonOffsetFillEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PolygonOffset, PolygonOffsetLineEnable, PolygonOffsetPointEnable

10.164 PolygonOffsetLineEnable

NAME
PolygonOffsetLineEnable - 3D API polygon offset line enable

USAGE
PolygonOffsetLineEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL depth offsetting for polygons rendered as line primitives. See the GL_POLYGON_OFFSET_LINE description on the OpenGL glEnable manual page for details.

The standard reset callback sets the PolygonOffsetLineEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
PolygonOffset, PolygonOffsetFillEnable, PolygonOffsetPointEnable

10.165 PolygonOffset

NAME
PolygonOffset - 3D API polygon offset
**USAGE**

PolygonOffset = float2(factor, units)

**VALID ENUMERANTS**

factor, units: floating point values

**DESCRIPTION**

Specify the scale factor and constant used to compute a depth offset for polygons. See the OpenGL glPolygonOffset manual page for details.

The standard reset callback sets the PolygonOffset state to float2(0, 0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

SlopScaleDepthBias, PolygonOffsetFillEnable, PolygonOffsetLineEnable, PolygonOffsetPointEnable

### 10.16.6 PolygonOffsetPointEnable

**NAME**

PolygonOffsetPointEnable - 3D API polygon offset point enable

**USAGE**

PolygonOffsetPointEnable = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Enable/disable OpenGL depth offsetting for polygons rendered as point primitives. See the GL_POLYGON_OFFSET_POINT description on the OpenGL glEnable manual page for details.

The standard reset callback sets the PolygonOffsetPointEnable state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

PolygonOffset, PolygonOffsetFillEnable, PolygonOffsetLineEnable
10.167 PolygonSmoothEnable

NAME
PolygonSmoothEnable - 3D API polygon smooth enable

USAGE
PolygonSmoothEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL antialiased rendering of polygons. See the GL_POLYGON_SMOOTH description on the OpenGL glEnable manual page for details.

The standard reset callback sets the PolygonSmoothEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LineSmoothEnable, PointSmoothEnable

10.168 PolygonStippleEnable

NAME
PolygonStippleEnable - 3D API polygon stipple enable

USAGE
PolygonStippleEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL polygon stippling. See the GL_POLYGON_STIPPLE description on the OpenGL glEnable manual page for details.

The standard reset callback sets the PolygonStippleEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
LineStipple, LineStippleEnable

10.169 ProjectionMatrix

NAME
ProjectionMatrix - 3D API projection matrix

USAGE
ProjectionMatrix = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set the values of the projection matrix. See the GL_PROJECTION description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_PROJECTION transform state type for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the ProjectionMatrix state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ModelViewMatrix, ModelViewTransform, ProjectionTransform, ViewTransform, WorldTransform

10.170 ProjectionTransform

NAME
ProjectionTransform - 3D API projection transform

USAGE
ProjectionTransform = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS
fi: floating point values
DESCRIPTION

Set the values of the projection matrix. See the GL_PROJECTION description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_PROJECTION transform state type for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the ProjectionTransform state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ModelViewMatrix, ModelViewTransform, ProjectionMatrix, ViewTransform, WorldTransform

10.171 RangeFogEnable

NAME

RangeFogEnable - 3D API range fog enable

USAGE

RangeFogEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable Direct3D range-based vertex fog. See the D3DRS_RANGEFOGENABLE render state description for DirectX 9.

The standard reset callback sets the RangeFogEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

FogColor, FogCoordSrc, FogDensity, FogDistanceMode, FogEnable, FogEnd, FogMode, FogStart, FogTableMode, FogVertexMode
10.172 RescaleNormalEnable

NAME
RescaleNormalEnable - 3D API rescale normal enable

USAGE
RescaleNormalEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable rescaling of vertex normal vectors by OpenGL. See the GL_RESCALE_NORMAL description on the OpenGL glEnable manual page for details.
The standard reset callback sets the RescaleNormalEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.2 or support for extension EXT_rescale_normal.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
AutoNormalEnable, NormalizeEnable, NormalizeNormals

10.173 ResultArg

NAME
ResultArg - 3D API result arg

USAGE
ResultArg = int(reg)

VALID ENUMERANTS
reg: Constant, Current, Diffuse, SelectMask, Specular, Temp, Texture, TFactor

DESCRIPTION
Specify the destination register for the result of this stage. See the D3DTSS_RESULTARG texture stage state description for DirectX 9.
The standard reset callback sets the ResultArg state to int(Current).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.174 SampleAlphaToCoverageEnable

NAME

SampleAlphaToCoverageEnable - 3D API sample alpha to coverage enable

USAGE

SampleAlphaToCoverageEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL alpha coverage sampling. See the GL_SAMPLE_ALPHA_TO_COVERAGE_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the SampleAlphaToCoverageEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.3 or support for extension ARB_multisample.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

SampleAlphaToOneEnable, SampleCoverageEnable, MultiSampleAntialias, MultiSampleMask, MultisampleEnable

10.175 SampleAlphaToOneEnable

NAME

SampleAlphaToOneEnable - 3D API sample alpha to one enable

USAGE

SampleAlphaToOneEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL maximum coverage sampling. See the GL_SAMPLE_ALPHA_TO_ONE_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the SampleAlphaToOneEnable state to bool(false).
OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.3 or support for extension ARB_multisample.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
SampleAlphaToCoverageEnable, SampleCoverageEnable, MultiSampleAntialias, MultiSampleMask, MultisampleEnable

10.176 SampleCoverageEnable

NAME
SampleCoverageEnable - 3D API sample coverage enable

USAGE
SampleCoverageEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL multisample coverage operations. See the GL_SAMPLE_COVERAGE_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the SampleCoverageEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.3 or support for extension ARB_multisample.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
SampleAlphaToCoverageEnable, SampleAlphaToOneEnable, MultiSampleAntialias, MultiSampleMask, MultisampleEnable

10.177 Sampler

NAME
Sampler - 3D API sampler

USAGE
Sampler[i] = sampler(s)
VALID ENUMERANTS

s: sampler

DESCRIPTION

Set all Direct3D sampler states for sampler s.
The standard reset callback resets all states for sampler s.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture

10.178 Scissor

NAME

Scissor - 3D API scissor

USAGE

Scissor = int4(x, y, w, h)

VALID ENUMERANTS

x, y, w, h: integer values

DESCRIPTION

Specify the scissor box in window coordinates. See the OpenGL glScissor manual page for details. See the description of the IDirect3DDevice9::SetScissorRect method for details

The standard reset callback sets the Scissor state to int4(0, 0, 10000, 10000).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ScissorTestEnable
10.179 ScissorTestEnable

NAME
ScissorTestEnable - 3D API scissor test enable

USAGE
ScissorTestEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable scissor testing. See the GL_SCISSOR_TEST description on the OpenGL glEnable manual page for
details. See the D3DRS_SCISSORTESTENABLE render state description for DirectX 9.
The standard reset callback sets the ScissorTestEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Scissor

10.180 SeparateAlphaBlendEnable

NAME
SeparateAlphaBlendEnable - 3D API separate alpha blend enable

USAGE
SeparateAlphaBlendEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enables Direct3D’s separate alpha blend mode. See the D3DRS_SEPARATEALPHABLENDENABLE render state
description for DirectX 9.
The standard reset callback sets the SeparateAlphaBlendEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, DestBlendAlpha, SrcBlendAlpha

10.181 ShadeModel

NAME

ShadeModel - 3D API shade model

USAGE

ShadeModel = int(model)

VALID ENUMERANTS

model: Flat, Smooth

DESCRIPTION

Specify the shading model. See the OpenGL glShadeModel manual page for details. See the D3DRS_SHademode render state description for DirectX 9.

The standard reset callback sets the ShadeModel state to int(Smooth).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ShadeModel

10.182 ShadeMode

NAME

ShadeMode - 3D API shade mode

USAGE

ShadeMode = int(model)

VALID ENUMERANTS

model: Flat, Smooth, Gouraud, Phong
DESCRIPTION
Specify the shading model. See the OpenGL glShadeModel manual page for details. See the D3DRS_SHADEMODE render state description for DirectX 9.
The standard reset callback sets the ShadeMode state to int(Smooth).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ShadeModel

10.183 SlopScaleDepthBias

NAME
SlopScaleDepthBias - 3D API slop scale depth bias

USAGE
SlopScaleDepthBias = float(factor)

VALID ENUMERANTS
factor: floating point value

DESCRIPTION
Specify the scale factor used to compute a depth offset for polygons. See the factor parameter description on the OpenGL glPolygonOffset manual page for details. See the D3DRS_SLOPESCALEDEPTHBIAS render state description for DirectX 9.
The standard reset callback sets the SlopScaleDepthBias state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PolygonOffset

10.184 SpecularEnable

NAME
SpecularEnable - 3D API specular enable
USAGE
SpecularEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable Direct3D specular highlighting. See the D3DRS_SPECULARENABLE render state description for DirectX 9.
The standard reset callback sets the SpecularEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
LightModelColorControl

10.185 SpecularMaterialSource

NAME
SpecularMaterialSource - 3D API specular material source

USAGE
SpecularMaterialSource = int(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable front and back specular color material. See the GL_SPECULAR description on the OpenGL glColorMaterial manual page for details. See the D3DRS_SPECULARMATERIALSOURCE render state description for DirectX 9.
The standard reset callback sets the SpecularMaterialSource state to int(AmbientAndDiffuse).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
10.186 SrcBlendAlpha

NAME

SrcBlendAlpha - 3D API source alpha blending type

USAGE

SrcBlendAlpha = int(blend)

VALID ENUMERANTS

blend: Zero, One, DestColor, InvDestColor, SrcAlpha, InvSrcAlpha, DstAlpha, InvDestAlpha, SrcAlphaSat, SrcColor, InvSrcColor, BlendFactor, InvBlendFactor

DESCRIPTION

Set the Direct3D separate alpha blending source blend type. See the D3DRS_SRCBLENDALPHA render state description for DirectX 9.

The standard reset callback sets the SrcBlendAlpha state to int(One).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, SrcBlend, VertexBlend, BlendOpAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.187 SrcBlend

NAME

SrcBlend - 3D API src blend

USAGE

SrcBlend = int(sfactor)

VALID ENUMERANTS

sfactor: Zero, One, DestColor, InvDestColor, SrcAlpha, InvSrcAlpha, DstAlpha, InvDestAlpha, SrcAlphaSat, SrcColor, InvSrcColor, BlendFactor, InvBlendFactor

DESCRIPTION

Specify the source blend factor. See the sfactor parameter description on the OpenGL glBlendFunc manual page for details. See the D3DRS_SRCBLEND render state description for DirectX 9.

The standard reset callback sets the SrcBlend state to int(One).
OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaBlendEnable, BlendEnable, BlendColor, BlendEquation, BlendEquationSeparate, BlendFunc, BlendFuncSeparate, BlendOp, DestBlend, IndexedVertexBlendEnable, VertexBlend, BlendFuncSeparate, BlendOpAlpha, DestBlendAlpha, SeparateAlphaBlendEnable

10.188 StencilEnable

NAME

StencilEnable - 3D API stencil enable

USAGE

StencilEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable stencil testing. See the GL_STENCIL_TEST description on the OpenGL glEnable manual page for details. See the D3DRS_STENCILENABLE render state description for DirectX 9.

The standard reset callback sets the StencilEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.189 StencilFail

NAME

StencilFail - 3D API stencil fail

USAGE

StencilFail = int(sfail)
VALID ENUMERANTS

sfail: integer value

DESCRIPTION

Specify the action for stencil test failures. See the sfail parameter description on the OpenGL glStencilOp manual page for details. See the D3DRS_STENCILFAIL render state description for DirectX 9.

The standard reset callback sets the StencilFail state to int(GL_KEEP).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.190 StencilFunc

NAME

StencilFunc - 3D API stencil func

USAGE

StencilFunc = int(func) StencilFunc = int3(func, ref, mask)

VALID ENUMERANTS


DESCRIPTION

Specify the stencil test function. See the OpenGL glStencilFunc manual page for details. See the D3DRS_STENCILFUNC render state description for DirectX 9.

The standard reset callback sets the StencilFunc state to int(Always) or int3(Always, 0, 0xFFFFFFFF).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFail, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode
10.191 StencilFuncSeparate

NAME
StencilFuncSeparate - 3D API stencil func separate

USAGE
StencilFuncSeparate = int4(face, func, ref, mask)

VALID ENUMERANTS

DESCRIPTION
Specify front and/or back stencil test functions. See the OpenGL glStencilFuncSeparate manual page for details. See the D3DRS_STENCILFUNC, D3DRS_STENCILREF, D3DRS_STENCILMASK, and D3DRS_CCW_STENCILFUNC render state descriptions for DirectX 9.

The standard reset callback sets the StencilFuncSeparate state to int4(GL_FRONT_AND_BACK, GL_ALWAYS, 0, ~0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 2.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearStencil,StencilEnable,StencilFail,StencilFunc,StencilMask,StencilMaskSeparate,StencilOp,StencilOpSeparate,StencilPass,StencilRef,StencilTestEnable,StencilTestTwoSideEnable,StencilWriteMask,StencilZFail,TwoSidedStencilMode

10.192 StencilMask

NAME
StencilMask - 3D API stencil mask

USAGE
StencilMask = int(mask)

VALID ENUMERANTS
mask: integer value

DESCRIPTION
Specify the stencil plane write mask. See the OpenGL glStencilMask manual page for details. See the D3DRS_STENCILWRITEMASK render state description for DirectX 9.

The standard reset callback sets the StencilMask state to int(~0).
10.193 StencilMaskSeparate

NAME

StencilMaskSeparate - 3D API stencil mask separate

USAGE

StencilMaskSeparate = int2(face, mask)

VALID ENUMERANTS

face: Front, Back, FrontAndBack

mask: integer value

DESCRIPTION

Specify front and/or back stencil plane write mask. See the OpenGL glStencilMaskSeparate manual page for details. See the D3DRS_STENCILMASK render state description for DirectX 9.

The standard reset callback sets the StencilMaskSeparate state to int2(FrontAndBack, ~0).

10.194 StencilOp

NAME

StencilOp - 3D API stencil op

USAGE

StencilOp = int3(sfail, dpfail, dppass)
VALID ENUMERANTS

sfail, dpfail, dppass: Keep, Zero, Replace, Incr, Decr, Invert, IncrWrap, DecrWrap, IncrSat, DecrSat

DESCRIPTION

Specify actions for stencil and depth tests. See the OpenGL glStencilOp manual page for details. See the D3DRS_STENCILFAIL, D3DRS_STENCILZFAIL, and D3DRS_STENCILPASS render state descriptions for DirectX 9.

The standard reset callback sets the StencilOp state to int3(Keep, Keep, Keep).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.195 StencilOpSeparate

NAME

StencilOpSeparate - 3D API stencil op separate

USAGE

StencilOpSeparate = int4(face, sfail, dpfail, dpass)

VALID ENUMERANTS

face: Front, Back, FrontAndBack sfail, dpfail, dppass: Keep, Zero, Replace, Incr, Decr, Invert, IncrWrap, DecrWrap, IncrSat, DecrSat

DESCRIPTION

Specify actions for front and/or stencil and depth tests. See the OpenGL glStencilOpSeparate manual page for details. See the D3DRS_STENCILFAIL, D3DRS_STENCILZFAIL, D3DRS_STENCILPASS, D3DRS_CCW_STENCILFAIL, D3DRS_CCW_STENCILZFAIL, and D3DRS_CCW_STENCILPASS render state descriptions for DirectX 9.

The standard reset callback sets the StencilOpSeparate state to int4(FrontAndBack, Keep, Keep, Keep).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 2.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9
10.196 StencilPass

NAME

StencilPass - 3D API stencil pass

USAGE

StencilPass = int(dppass)

VALID ENUMERANTS

dppass: integer value

DESCRIPTION

Specify action to take when both the stencil and depth tests pass. See the dppass parameter description on the OpenGL glStencilOp manual page for details. See the D3DRS_STENCILPASS render state description for DirectX 9.

The standard reset callback sets the StencilPass state to int(GL_KEEP).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.197 StencilRef

NAME

StencilRef - 3D API stencil ref

USAGE

StencilRef = int(ref)

VALID ENUMERANTS

ref: integer value
DESCRIPTION

Specify the reference value for the stencil test. See the ref parameter description on the OpenGL glStencilFunc manual page for details. See the D3DRS_STENCILREF render state description for DirectX 9.

The standard reset callback sets the StencilRef state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.198 StencilTestEnable

NAME

StencilTestEnable - 3D API stencil test enable

USAGE

StencilTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable stencil testing. See the GL_STENCIL_TEST description on the OpenGL glEnable manual page for details. See the D3DRS_STENCILENABLE render state description for DirectX 9.

The standard reset callback sets the StencilTestEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode
10.199 StencilTestTwoSideEnable

NAME
StencilTestTwoSideEnable - 3D API stencil test two side enable

USAGE
StencilTestTwoSideEnable = bool(enable)

VALID ENUMERANTs
enable: true, false

DESCRIPTION
Enable/disable two-sided stenciling. See the OpenGL EXT_stencil_two_side specification for details. See the D3DRS_TWOSIDEDSTENCILMODE render state description for DirectX 9.
The standard reset callback sets the StencilTestTwoSideEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
Support for extension EXT_stencil_two_side.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilWriteMask, StencilZFail, TwoSidedStencilMode

10.200 StencilWriteMask

NAME
StencilWriteMask - 3D API stencil write mask

USAGE
StencilWriteMask = int(mask)

VALID ENUMERANTs
mask: integer value

DESCRIPTION
Specify the stencil plane write mask. See the OpenGL glStencilMask manual page for details. See the D3DRS_STENCILWRITEMASK render state description for DirectX 9.
The standard reset callback sets the StencilWriteMask state to int(~0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilZFail, TwoSidedStencilMode

10.201 StencilZFail

NAME
StencilZFail - 3D API stencil Z fail

USAGE
StencilZFail = int(dpfail)

VALID ENUMERANTS
dpfail: integer value

DESCRIPTION
Specifies action for when the stencil test passes but the depth test fails. See the dpfail parameter description on the OpenGL glStencilOp manual page for details. See the D3DRS_STENCILZFAIL render state description for DirectX 9.

The standard reset callback sets the StencilZFail state to int(GL_KEEP).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilOpSeparate, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilZFail, TwoSidedStencilMode

10.202 TessellationControlProgram

NAME
TessellationControlProgram - 3D API tessellation control program

USAGE
TessellationControlProgram = compile profile “-po profileOption” main(args);
VALID ENUMERANTS

prog: Null

DESCRIPTION

Compile a tessellation control program. See the specification of the OpenGL tessellation control profile for details. See the specification of the Direct3D 11 hull shaders for details.

The standard reset callback sets the TessellationControlProgram state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 11

SEE ALSO


10.203 TessellationControlShader

NAME

TessellationControlShader - 3D API tessellation control shader

USAGE

TessellationControlShader = compile profile “-po profileOption” main(args);

VALID ENUMERANTS

shader: Null

DESCRIPTION

Compile a tessellation control shader. See the specification of the OpenGL tessellation control profile for details. See the specification of the Direct3D 11 hull shaders for details.

The standard reset callback sets the TessellationControlShader state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 11

SEE ALSO

10.204 TessellationEvaluationProgram

NAME

TessellationEvaluationProgram - 3D API tessellation evaluation program

USAGE

TessellationEvaluationProgram = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

prog: Null

DESCRIPTION

Compile a tessellation evaluation program. See the specification of the OpenGL tessellation evaluation profile for details. See the specification of the Direct3D 11 domain shaders for details.

The standard reset callback sets the TessellationEvaluationProgram state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 11

SEE ALSO

FragmentProgram, GeometryProgram, TessellationControlProgram, VertexProgram, GeometryShader, PixelShader, TessellationControlShader, TessellationEvaluationShader, VertexShader

10.205 TessellationEvaluationShader

NAME

TessellationEvaluationShader - 3D API tessellation evaluation shader

USAGE

TessellationEvaluationShader = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

shader: Null

DESCRIPTION

Compile a tessellation evaluation shader. See the specification of the OpenGL tessellation evaluation profile for details. See the specification of the Direct3D 11 domain shaders for details.

The standard reset callback sets the TessellationEvaluationShader state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 11

SEE ALSO

GeometryShader, PixelShader, TessellationControlShader, VertexShader, FragmentProgram, GeometryProgram, TessellationControlProgram, TessellationEvaluationProgram, VertexProgram

10.206 TexCoordIndex

NAME

TexCoordIndex - 3D API tex coord index

USAGE

TexCoordIndex = int(index)

VALID ENUMERANTS

index: integer value

DESCRIPTION

Specify the index of the texture coordinate set to use with this texture stage. See the D3DTSS_TEXCOORDINDEX texture stage state description for DirectX 9.

The standard reset callback sets the TexCoordIndex state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.207 TexGenQEnable

NAME

TexGenQEnable - 3D API tex gen Q enable

USAGE

TexGenQEnable[i] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL generation of the q texture coordinate. See the GL_TEXTURE_GEN_Q description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenQEnable[i] state to bool(false).
10.208 TexGenQEYEPlane

NAME
	TexGenQEYEPlane - 3D API tex gen Q eye plane

USAGE
	TexGenQEYEPlane[i] = float4(p1, p2, p3, p4)

VALID ENUMERANTS
	p1, p2, p3, p4: floating point values

DESCRIPTION
Specify the texture coordinate generation parameters for the Q eye plane. See the GL_EYE_PLANE description on the OpenGL gTexGen manual page for details.

The standard reset callback sets the TexGenQEYEPlane[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO
	TexGenQEnable, TexGenQMode, TexGenQObjectPlane, TexGenREyePlane, TexGenSEyePlane, TexGenTEyePlane

10.209 TexGenQMode

NAME
	TexGenQMode - 3D API tex gen Q mode

USAGE
	TexGenQMode[i] = int(mode)

VALID ENUMERANTS

mode: ObjectLinear, EyeLinear, SphereMap, ReflectionMap, NormalMap
**DESCRIPTION**

Specify the Q texture coordinate generation mode. See the GL_TEXTURE_GEN_MODE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenQMode[i] state to int(EyeLinear).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

TexGenQEnable, TexGenQEyePlane, TexGenQObjectPlane, TexGenRMode, TexGenSMode, TexGenTMode

### 10.210 TexGenQObjectPlane

**NAME**

TexGenQObjectPlane - 3D API tex gen Q object plane

**USAGE**

TexGenQObjectPlane[i] = float4(p1, p2, p3, p4)

**VALID ENUMERANTS**

p1, p2, p3, p4: floating point values

**DESCRIPTION**

Specify the texture coordinate generation parameters for the Q object plane. See the GL_OBJECT_PLANE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenQObjectPlane[i] state to float4(0, 0, 0, 0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

TexGenQEnable, TexGenQEyePlane, TexGenQMode, TexGenRObjectPlane, TexGenSObjectPlane, TexGenTObjectPlane

### 10.211 TexGenREnable

**NAME**

TexGenREnable - 3D API tex gen R enable

**Usage**

TexGenREnable[i] = bool(enable)

**Valid Enumerants**

enable: true, false

**Description**

Enable/disable OpenGL generation of the r texture coordinate. See the GL_TEXTURE_GEN_R description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenREnable[i] state to bool(false).

**OpenGL Functionality Requirements**

OpenGL 1.0

**Direct3D Functionality Requirements**

Not applicable.

**See Also**

TexGenREyePlane, TexGenRMode, TexGenRObjectPlane, TexGenQEnable, TexGenSEnable, TexGenTEnable

### 10.212 TexGenREyePlane

**Name**

TexGenREyePlane - 3D API tex gen R eye plane

**Usage**

TexGenREyePlane[i] = float4(p1, p2, p3, p4)

**Valid Enumerants**

p1, p2, p3, p4: floating point values

**Description**

Specify the texture coordinate generation parameters for the R eye plane. See the GL_EYE_PLANE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenREyePlane[i] state to float4(0, 0, 0, 0).

**OpenGL Functionality Requirements**

OpenGL 1.0

**Direct3D Functionality Requirements**

Not applicable.

**See Also**

TexGenREnable, TexGenRMode, TexGenRObjectPlane, TexGenQEyePlane, TexGenSEyePlane, TexGenTEyePlane
10.213 TexGenRMode

NAME

TexGenRMode - 3D API tex gen R mode

USAGE

TexGenRMode[i] = int(mode)

VALID ENUMERANTS

mode: ObjectLinear, EyeLinear, SphereMap, ReflectionMap, NormalMap

DESCRIPTION

Specify the R texture coordinate generation mode. See the GL_TEXTURE_GEN_MODE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenRMode[i] state to int(EyeLinear).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

TexGenREnable, TexGenREyePlane, TexGenRObjectPlane, TexGenQMode, TexGenSMode, TexGenTMode

10.214 TexGenRObjectPlane

NAME

TexGenRObjectPlane - 3D API tex gen R object plane

USAGE

TexGenRObjectPlane[i] = float4(p1, p2, p3, p4)

VALID ENUMERANTS

p1, p2, p3, p4: floating point values

DESCRIPTION

Specify the texture coordinate generation parameters for the R object plane. See the GL_OBJECT_PLANE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenRObjectPlane[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
TexGenREnable, TexGenREyePlane, TexGenRMode, TexGenQObjectPlane, TexGenSObjectPlane, TexGenTObjectPlane

10.215 TexGenSEnable

NAME
TexGenSEnable - 3D API tex gen S enable

USAGE
TexGenSEnable[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL generation of the s texture coordinate. See the GL_TEXTURE_GEN_S description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenSEnable[i] state to bool(false).

OPengl FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
TexGenSEyePlane, TexGenSMode, TexGenQObjectPlane, TexGenQEnable, TexGenREnable, TexGenTEnable

10.216 TexGenSEyePlane

NAME
TexGenSEyePlane - 3D API tex gen S eye plane

USAGE
TexGenSEyePlane[i] = float4(p1, p2, p3, p4)

VALID ENUMERANTS
p1, p2, p3, p4: floating point values
DESCRIPTION

Specify the texture coordinate generation parameters for the S eye plane. See the GL_EYE_PLANES description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenSEyePlane[i] state to float4(1, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

TexGenSEnable, TexGenSMo de, TexGenSObjectPlane, TexGenQEyePlane, TexGenREyePlane, TexGenTEyePlane

10.217 TexGenSMo de

NAME

TexGenSMo de - 3D API tex gen S mode

USAGE

TexGenSMo de[i] = int(mode)

VALID ENUMERANTS

mode: ObjectLinear, EyeLinear, SphereMap, ReflectionMap, NormalMap

DESCRIPTION

Specify the S texture coordinate generation mode. See the GL_TEXTURE_GEN_MODE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenSMo de[i] state to int(EyeLinear).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

TexGenSEnable, TexGenSEyePlane, TexGenSObjectPlane, TexGenQMode, TexGenRMode, TexGenTMode

10.218 TexGenSObjectPlane

NAME

TexGenSObjectPlane - 3D API tex gen S object plane
USAGE
TexGenSObjectPlane[i] = float4(p1, p2, p3, p4)

VALID ENUMERANTS
p1, p2, p3, p4: floating point values

DESCRIPTION
Specify the texture coordinate generation parameters for the S object plane. See the GL_OBJECT_PLANE description on the OpenGL glTexGen manual page for details.
The standard reset callback sets the TexGenSObjectPlane[i] state to float4(1, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
TexGenSEnable, TexGenSEyePlane, TexGenSMode, TexGenQObjectPlane, TexGenRObjectPlane, TexGenTObjectPlane

10.219 TexGenTEnable

NAME
TexGenTEnable - 3D API tex gen T enable

USAGE
TexGenTEnable[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL generation of the t texture coordinate. See the GL_TEXTURE_GEN_T description on the OpenGL glEnable manual page for details.
The standard reset callback sets the TexGenTEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
TexGenTEyePlane, TexGenTMode, TexGenTObjectPlane, TexGenQEnable, TexGenREnable, TexGenSEnable
10.220 TexGenTEyePlane

NAME

TexGenTEyePlane - 3D API tex gen T eye plane

USAGE

TexGenTEyePlane[i] = float4(p1, p2, p3, p4)

VALID ENUMERANTS

p1, p2, p3, p4: floating point values

DESCRIPTION

Specify the texture coordinate generation parameters for the T eye plane. See the GL_EYE_PLANE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenTEyePlane[i] state to float4(0, 1, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

TexGenTEnable, TexGenTMode, TexGenTObjectPlane, TexGenQEyePlane, TexGenREyePlane, TexGenSEyePlane

10.221 TexGenTMode

NAME

TexGenTMode - 3D API tex gen T mode

USAGE

TexGenTMode[i] = int(mode)

VALID ENUMERANTS

mode: ObjectLinear, EyeLinear, SphereMap, ReflectionMap, NormalMap

DESCRIPTION

Specify the T texture coordinate generation mode. See the GL_TEXTURE_GEN_MODE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenTMode[i] state to int(EyeLinear).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
10.222 TexGenTObjectPlane

**NAME**

TexGenTObjectPlane - 3D API tex gen T object plane

**USAGE**

TexGenTObjectPlane[i] = float4(p1, p2, p3, p4)

**VALID ENUMERANTS**

p1, p2, p3, p4: floating point values

**DESCRIPTION**

Specify the texture coordinate generation parameters for the T object plane. See the GL_OBJECT_PLANE description on the OpenGL glTexGen manual page for details.

The standard reset callback sets the TexGenTObjectPlane[i] state to float4(0, 1, 0, 0).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

TexGenTEnable, TexGenTEyePlane, TexGenTObjectPlane, TexGenQMode, TexGenRMode, TexGenSMode

10.223 Texture1DEnable

**NAME**

Texture1DEnable - 3D API texture1D enable

**USAGE**

Texture1DEnable[i] = bool(enable)

**VALID ENUMERANTS**

enable: true, false
DESCRIPTION
Enable/disable OpenGL one-dimensional texturing. See the GL_TEXTURE_1D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture1DEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.224 Texture1D

NAME
Texture1D - 3D API texture1D

USAGE
Texture1D[i] = sampler1D(samp)

VALID ENUMERANTS
samp: sampler

DESCRIPTION
Specify the 1D texture to be used. See the GL_TEXTURE_1D description on the OpenGL glBindTexture manual page for details. See the IDirect3DDevice9::SetTexture method for details.

The standard reset callback sets the Texture1D[i] state to sampler1D(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias
10.225 Texture2DEnable

NAME
Texture2DEnable - 3D API texture2D enable

USAGE
Texture2DEnable[i] = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable OpenGL two-dimensional texturing. See the GL_TEXTURE_2D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture2DEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
Texture1DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.226 Texture2D

NAME
Texture2D - 3D API texture2D

USAGE
Texture2D[i] = sampler2D(samp)

VALID ENUMERANTS
samp: sampler

DESCRIPTION
Specify the 2D texture to be used. See the GL_TEXTURE_2D description on the OpenGL glBindTexture manual page for details. See the IDirect3DDevice9::SetTexture method for details.

The standard reset callback sets the Texture2D[i] state to sampler2D(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

### 10.227 Texture3DEnable

**NAME**

Texture3DEnable - 3D API texture3D enable

**USAGE**

Texture3DEnable[i] = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Enable/disable OpenGL three-dimensional texturing. See the GL_TEXTURE_3D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture3DEnable[i] state to bool(false).

**OPENVGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

Texture1DEnable, Texture2DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

### 10.228 Texture3D

**NAME**

Texture3D - 3D API texture3D

**USAGE**

Texture3D[i] = sampler3D(samp)

**VALID ENUMERANTS**

samp: sampler
DESCRIPTION

Specify the 3D texture to be used. See the GL_TEXTURE_3D description on the OpenGL glBindTexture manual page for details. See the IDirect3DDevice9::SetTexture method for details.

The standard reset callback sets the Texture3D[i] state to sampler3D(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.229 TextureCubeMapEnable

NAME

TextureCubeMapEnable - 3D API texture cube map enable

USAGE

TextureCubeMapEnable[i] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL cube-mapped texturing. See the GL_TEXTURE_CUBE_MAP_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TextureCubeMapEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias
10.230 TextureCubeMap

NAME
TextureCubeMap - 3D API texture cube map

USAGE
TextureCubeMap[i] = samplerCUBE(samp)

VALID ENUMERANTS
samp: sampler

DESCRIPTION
Specify the cube map texture to be used. See the GL_TEXTURE_CUBE_MAP description on the OpenGL glBindTexture manual page for details. See the IDirect3DDevice9::SetTexture method for details.
The standard reset callback sets the TextureCubeMap[i] state to samplerCUBE(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Texture1Enable, Texture2Enable, Texture3Enable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.231 TextureEnvColor

NAME
TextureEnvColor - 3D API texture env color

USAGE
TextureEnvColor[i] = float4(r, g, b, a)

VALID ENUMERANTS
r, g, b, a: floating point values

DESCRIPTION
Specify the color used for multiple-texture blending. See the GL_TEXTURE_ENV_COLOR description on the OpenGL glTexCoord manual page for details. See the D3DRS_TEXTUREFACTOR render state description and the D3DTSS_COLORARG2 texture stage description for details.
The standard reset callback sets the TextureEnvColor[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
10.232 TextureEnvMode

NAME
TextureEnvMode - 3D API texture env mode

USAGE
TextureEnvMode[i] = int(mode)

VALID ENUMERANTS
mode: Modulate, Decal, Blend, Replace, Add

DESCRIPTION
Set the texture environment mode. See the GL_TEXTURE_ENV_MODE description on the OpenGL glTexEnv manual page for details. See the D3DTSS_COLOROP texture stage state description for DirectX 9.

The standard reset callback sets the TextureEnvMode[i] state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags

10.233 TextureFactor

NAME
TextureFactor - 3D API texture factor

USAGE
TextureFactor = int(color)
VALID ENUMERANTS

color: integer value

DESCRIPTION

Specify the color used for multiple-texture blending. See the D3DRS_TEXTUREFACTOR render state description for DirectX 9.

The standard reset callback sets the TextureFactor state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.234 TextureMatrix

NAME

TextureMatrix - 3D API texture matrix

USAGE

TextureMatrix = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the texture matrix. See the GL_TEXTURE description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_TEXTUREi transform state type for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the TextureMatrix state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureTransform, TextureTransformFlags
10.235 TextureRectangleEnable

NAME

TextureRectangleEnable - 3D API texture rectangle enable

USAGE

TextureRectangleEnable[i] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Enable/disable OpenGL non-power-of-two texturing. See the OpenGL NV_texture_rectangle specification for details. The standard reset callback sets the TextureRectangleEnable[i] state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.236 TextureRectangle

NAME

TextureRectangle - 3D API texture rectangle

USAGE

TextureRectangle[i] = samplerRECT(samp)

VALID ENUMERANTS

samp: sampler

DESCRIPTION

Specify the rectangle texture to be used. See the GL_TEXTURE_RECTANGLE_NV description in the OpenGL NV_texture_rectangle specification for details. See the IDirect3DDevice9::SetTexture method for details. The standard reset callback sets the TextureRectangle[i] state to samplerRECT(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform, TextureTransformFlags, Texture, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

10.237 TextureTransform

NAME

TextureTransform - 3D API texture transform

USAGE

TextureTransform = float4x4(f1, f2, f3, f4, ...) , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the texture matrix. See the GL_TEXTURE description on the OpenGL glMatrixMode manual page for details. See the description of the D3DTS_TEXTUREi transform state type for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the TextureTransform state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform

10.238 TextureTransformFlags

NAME

TextureTransformFlags - 3D API texture transform flags

USAGE

TextureTransformFlags = int(flags)
VALID ENUMERANTS
flags: Disable, Count1, Count2, Count3, Count4, Projected

DESCRIPTION
Specify how texture coordinates are transformed for this texture stage. See the D3DTSS_TEXTURETRANSFORMFLAGS texture stage state description for DirectX 9.

The standard reset callback sets the TextureTransformFlags state to int(Disable).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable, Texture1D, Texture2D, Texture3D, TextureCubeMap, TextureRectangle, TextureEnvColor, TextureEnvMode, TextureFactor, TextureMatrix, TextureTransform

10.239 TweenFactor

NAME
TweenFactor - 3D API tween factor

USAGE
TweenFactor = float(factor)

VALID ENUMERANTS
factor: floating point value

DESCRIPTION
Specify the tween factor. See the D3DRS_TWEENFACTOR render state description for DirectX 9.

The standard reset callback sets the TweenFactor state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO

10.240 TwoSidedStencilMode

NAME
TwoSidedStencilMode - 3D API two sided stencil mode
**USAGE**

TwoSidedStencilMode = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Enable Direct3D’s two-sided stenciling. See the D3DRS_TWOSIDEDSTENCILMODE render state description for DirectX 9.

The standard reset callback sets the TwoSidedStencilMode state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9

**SEE ALSO**

StencilOpSeparate ClearStencil, StencilEnable, StencilFail, StencilFunc, StencilFuncSeparate, StencilMask, StencilMaskSeparate, StencilOp, StencilPass, StencilRef, StencilTestEnable, StencilTestTwoSideEnable, StencilWriteMask, StencilZFail

### 10.241 VertexBlend

**NAME**

VertexBlend - 3D API vertex blend

**USAGE**

VertexBlend = int(num)

**VALID ENUMERANTS**

num: integer value

**DESCRIPTION**

Specify the number of matrices to use to perform geometry blending. See the D3DRS_VERTEXBLEND render state description for DirectX 9.

The standard reset callback sets the VertexBlend state to int(Disable).

**OPENGL FUNCTIONALITY REQUIREMENTS**

Not applicable.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

DirectX 9
### 10.242 VertexEnvParameter

**NAME**

*VertexEnvParameter* - 3D API vertex env parameter

**USAGE**

VertexEnvParameter[i] = float4(x, y, z, w)

**VALID ENUMERANTS**

x, y, z, w: floating point values

**DESCRIPTION**

Set vertex program environment parameter. See the OpenGL glProgramEnvParameter4fvARB manual page for details. The standard reset callback sets the VertexEnvParameter[i] state to float4(0, 0, 0, 0).

**OPENVGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

*FragmentEnvParameter, FragmentLocalParameter, VertexLocalParameter, PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstant1, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstant1, vertexShaderConstant4*

### 10.243 VertexLocalParameter

**NAME**

*VertexLocalParameter* - 3D API vertex local parameter

**USAGE**

VertexLocalParameter[i] = float4(x, y, z, w)

**VALID ENUMERANTS**

x, y, z, w: floating point values
DESCRIPTION

Set vertex program local parameter. See the OpenGL glProgramLocalParameter4fvARB manual page for details.

The standard reset callback sets the VertexLocalParameter[i] state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4

10.244 VertexProgram

NAME

VertexProgram - 3D API vertex program

USAGE

VertexProgram = compile profile "-po profileOption" main(args);

VALID ENUMERANTS

prog: Null

DESCRIPTION

Compile a vertex program. See the specification of the OpenGL vertex profile for details. See the specification of the Direct3D vertex shaders for details.

The standard reset callback sets the VertexProgram state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

10.245 **VertexProgramPointSizeEnable**

**NAME**

*VertexProgramPointSizeEnable* - 3D API vertex program point size enable

**USAGE**

VertexProgramPointSizeEnable = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Enable/disable using OpenGL shader builtin `gl_PointSize` in vertex shaders. See the `GL_VERTEX_PROGRAM_POINT_SIZE` description on the OpenGL glEnable manual page for details.

The standard reset callback sets the VertexProgramPointSizeEnable state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

*PointScaleEnable, PointScale_A, PointScale_B, PointScale_C, PointSizeMax, PointSizeMin, PointSize*

10.246 **VertexProgramTwoSideEnable**

**NAME**

*VertexProgramTwoSideEnable* - 3D API vertex program two side enable

**USAGE**

VertexProgramTwoSideEnable = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Enable/disable OpenGL vertex program color selection based on the polygon face direction. See the `GL_VERTEX_PROGRAM_TWO_SIDE` description on the OpenGL glEnable manual page for details.

The standard reset callback sets the VertexProgramTwoSideEnable state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
FrontFace

10.247 VertexShaderConstant1

NAME
VertexShaderConstant1 - 3D API vertex shader constant1

USAGE
VertexShaderConstant1[i] = float4(f1, f2, f3, f4)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.248 VertexShaderConstant2

NAME
VertexShaderConstant2 - 3D API vertex shader constant2

USAGE
VertexShaderConstant2[i] = float4x2(f1, f2, f3, f4, f5, f6, f7, f8)

VALID ENUMERANTS
fi: floating point values
DESCRIPTION
Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, VertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.249 VertexShaderConstant3

NAME
VertexShaderConstant3 - 3D API vertex shader constant3

USAGE
VertexShaderConstant3[i] = float4x3(f1, f2, f3, f4, ..., f9, f10, f11, f12)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, VertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter
10.250 vertexShaderConstant4

NAME
vertexShaderConstant4 - 3D API vertex shader constant4

USAGE
vertexShaderConstant4[i] = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstant4, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.251 VertexShaderConstantB

NAME
VertexShaderConstantB - 3D API vertex shader constantB

USAGE
VertexShaderConstantB[i] = bool4(b1, b2, b3, b4)

VALID ENUMERANTS
bi: true, false

DESCRIPTION
Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantB method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.252 VertexShaderConstantF

NAME

VertexShaderConstantF - 3D API vertex shader constantF

USAGE

VertexShaderConstantF[i] = float4(f1, f2, f3, f4)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details. The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantF, VertexShaderConstantI, vertexShaderConstant4, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.253 VertexShaderConstantI

NAME

VertexShaderConstantI - 3D API vertex shader constantI

USAGE

VertexShaderConstantI[i] = int4(i1, i2, i3, i4)
VALID ENUMERANTS

ii: integer values

DESCRIPTION

Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantI method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, FragmentEnvParameter, FragmentLocalParameter, VertexEnvParameter, VertexLocalParameter

10.254 VertexShaderConstant

NAME

VertexShaderConstant - 3D API vertex shader constant

USAGE

VertexShaderConstant[i] = float4x4(f1, f2, f3, f4, ..., f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set a Direct3D vertex shader constant. See the IDirect3DDevice9::SetVertexShaderConstantF method for details.
The standard reset callback does nothing.

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

PixelShaderConstant, PixelShaderConstant1, PixelShaderConstant2, PixelShaderConstant3, PixelShaderConstant4, PixelShaderConstantB, PixelShaderConstantF, PixelShaderConstantI, VertexShaderConstant, VertexShaderConstant1, VertexShaderConstant2, VertexShaderConstant3, VertexShaderConstantB, VertexShaderConstantF, VertexShaderConstantI, ver-
10.255 VertexShader

NAME

VertexShader - 3D API vertex shader

USAGE

VertexShader = compile profile “-po profileOption” main(args);

VALID ENUMERANTS

shader: Null

DESCRIPTION

Compile a vertex shader. See the specification of the OpenGL vertex profile for details. See the specification of the Direct3D vertex shaders for details.

The standard reset callback sets the VertexShader state to program(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO


10.256 ViewTransform

NAME

ViewTransform - 3D API view transform

USAGE

ViewTransform = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS

fi: floating point values

DESCRIPTION

Set the values of the view matrix. See the description of the D3DTS_VIEW transform state type for the IDirect3DDevice9::SetTransform method.

The standard reset callback sets the ViewTransform state to the identity matrix.
OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ModelViewMatrix, ModelViewTransform, ProjectionMatrix, ProjectionTransform, WorldTransform

10.257 WorldTransform

NAME
WorldTransform - 3D API world transform

USAGE
WorldTransform = float4x4(f1, f2, f3, f4, ... , f13, f14, f15, f16)

VALID ENUMERANTS
fi: floating point values

DESCRIPTION
Set the values of the world matrix. See the description of the D3DTS_WORLD transform state type for the IDirect3DDevice9::SetTransform method.
The standard reset callback sets the WorldTransform state to the identity matrix.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ModelViewMatrix, ModelViewTransform, ProjectionMatrix, ProjectionTransform, ViewTransform

10.258 Wrap0

NAME
Wrap0 - 3D API wrap0

USAGE
Wrap0 = int(flags)
VALID ENUMERANTS
flags: integer value
Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 0. See the D3DRS_WRAP0 render state description for DirectX 9.

The standard reset callback sets the Wrap0 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.259 Wrap10

NAME
Wrap10 - 3D API wrap10

USAGE
Wrap10 = int(flags)

VALID ENUMERANTS
flags: integer value
Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 10. See the D3DRS_WRAP10 render state description for DirectX 9.

The standard reset callback sets the Wrap10 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9
SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.260 Wrap11

NAME

Wrap11 - 3D API wrap11

USAGE

Wrap11 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 11. See the D3DRS_WRAP11 render state description for DirectX 9.

The standard reset callback sets the Wrap11 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap12, Wrap13, Wrap14, Wrap15

10.261 Wrap12

NAME

Wrap12 - 3D API wrap12

USAGE

Wrap12 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.
DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 12. See the D3DRS_WRAP12 render state description for DirectX 9.

The standard reset callback sets the Wrap12 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap13, Wrap14, Wrap15

10.262 Wrap13

NAME

Wrap13 - 3D API wrap13

USAGE

Wrap13 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 13. See the D3DRS_WRAP13 render state description for DirectX 9.

The standard reset callback sets the Wrap13 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap14, Wrap15
10.263 Wrap14

NAME

Wrap14 - 3D API wrap14

USAGE

Wrap14 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 14. See the D3DRS_WRAP14 render state description for DirectX 9.

The standard reset callback sets the Wrap14 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap15

10.264 Wrap15

NAME

Wrap15 - 3D API wrap15

USAGE

Wrap15 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 15. See the D3DRS_WRAP15 render state description for DirectX 9.

The standard reset callback sets the Wrap15 state to int(0).
OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14

10.265 Wrap1

NAME
Wrap1 - 3D API wrap1

USAGE
Wrap1 = int(flags)

VALID ENUMERANTS
flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 1. See the D3DRS_WRAP1 render state description for DirectX 9.

The standard reset callback sets the Wrap1 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.266 Wrap2

NAME
Wrap2 - 3D API wrap2
USAGE

Wrap2 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 2. See the D3DRS_WRAP2 render state description for DirectX 9.

The standard reset callback sets the Wrap2 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.267 Wrap3

NAME

Wrap3 - 3D API wrap3

USAGE

Wrap3 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 3. See the D3DRS_WRAP3 render state description for DirectX 9.

The standard reset callback sets the Wrap3 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.
DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.268 Wrap4

NAME
Wrap4 - 3D API wrap4

USAGE
Wrap4 = int(flags)

VALID ENUMERANTS
flags: integer value
Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 4. See the D3DRS_WRAP4 render state description for DirectX 9.
The standard reset callback sets the Wrap4 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.269 Wrap5

NAME
Wrap5 - 3D API wrap5

USAGE
Wrap5 = int(flags)
VALID ENUMERANTS
flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 5. See the D3DRS_WRAP5 render state description for DirectX 9.

The standard reset callback sets the Wrap5 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap6, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.270 Wrap6

NAME
Wrap6 - 3D API wrap6

USAGE
Wrap6 = int(flags)

VALID ENUMERANTS
flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 6. See the D3DRS_WRAP6 render state description for DirectX 9.

The standard reset callback sets the Wrap6 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9
SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap7, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.271 Wrap7

NAME

Wrap7 - 3D API wrap7

USAGE

Wrap7 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION

Set the Direct3D texture wrapping state for texture coordinate 7. See the D3DRS_WRAP7 render state description for DirectX 9.

The standard reset callback sets the Wrap7 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap8, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.272 Wrap8

NAME

Wrap8 - 3D API wrap8

USAGE

Wrap8 = int(flags)

VALID ENUMERANTS

flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.
DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 8. See the D3DRS_WRAP8 render state description for DirectX 9.

The standard reset callback sets the Wrap8 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap9, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15

10.273 Wrap9

NAME
Wrap9 - 3D API wrap9

USAGE
Wrap9 = int(flags)

VALID ENUMERANTS
flags: integer value

Any combination of D3DWRAPCOORD_0, D3DWRAPCOORD_1, D3DWRAPCOORD_2, and D3DWRAPCOORD_3. Setting flags to 0 disables texture wrapping in all directions.

DESCRIPTION
Set the Direct3D texture wrapping state for texture coordinate 9. See the D3DRS_WRAP9 render state description for DirectX 9.

The standard reset callback sets the Wrap9 state to int(0).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
Wrap0, Wrap1, Wrap2, Wrap3, Wrap4, Wrap5, Wrap6, Wrap7, Wrap8, Wrap10, Wrap11, Wrap12, Wrap13, Wrap14, Wrap15
10.274 ZEnable

NAME
ZEnable - 3D API Z enable

USAGE
ZEnable = bool(enable)

VALID ENUMERANTS
enable: true, false

DESCRIPTION
Enable/disable depth testing. See the GL_DEPTH_TEST description on the OpenGL glEnable manual page for details.
See the D3DRS_ZENABLE render state description for DirectX 9.
The standard reset callback sets the ZEnable state to bool(false).

OPENVGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthFunc, DepthMask, DepthRange, DepthTestEnable, SlopScaleDepthBias, ZFunc, ZWriteEnable

10.275 ZFunc

NAME
ZFunc - 3D API Z func

USAGE
ZFunc = int(func)

VALID ENUMERANTS
func: Never, Less, LEqual, LessEqual, Equal, Greater, NotEqual, GEqual, GreaterEqual, Always

DESCRIPTION
Specify the function used for depth buffer comparisons. See the OpenGL glDepthFunc manual page for details. See the D3DRS_ZFUNC render state description for DirectX 9.
The standard reset callback sets the ZFunc state to int(Less).

OPENVGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthTestEnable, DepthFunc, DepthMask, DepthRange, DepthTestEnable, SlopScaleDepthBias, ZEnable, ZWriteEnable

10.276 ZWriteEnable

NAME

ZWriteEnable - 3D API Z write enable

USAGE

ZWriteEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Specify whether the depth buffer is enabled for writing. See the OpenGL glDepthMask manual page for details. See the D3DRS_ZWRITEENABLE render state description for DirectX 9.

The standard reset callback sets the ZWriteEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

ClearDepth, DepthBias, DepthBounds, DepthBoundsEnable, DepthClampEnable, DepthTestEnable, DepthFunc, DepthMask, DepthRange, DepthTestEnable, SlopScaleDepthBias, ZEnable, ZFunc
CGFX_SAMPLER_STATES

11.1 AddressU

NAME
AddressU - 3D API U texture addressing mode

USAGE
AddressU = int(mode)
WrapS = int(mode)

VALID ENUMERANTS
mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION
Set the Direct3D u-direction texture wrapping mode. See the D3DSAMP_ADDRESSU sampler state description for DirectX 9.

Set the wrap parameter for texture coordinate S. See the GL_TEXTURE_WRAP_S description on the OpenGL glTexParameteri manual page for details.

The standard reset callback sets the AddressU or WrapS state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS
Clamp, Repeat, Wrap: OpenGL 1.0
ClampToEdge: OpenGL 1.2
Border, ClampToBorder: OpenGL 1.3
Mirror, MirroredRepeat: OpenGL 1.4
MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce: support for extension EXT_mirror_clamp.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
AddressUi, AddressV, AddressW, WrapS, WrapT, WrapR
11.2 AddressV

NAME

AddressV - 3D API V texture addressing mode

USAGE

AddressV = int(mode)
WrapT = int(mode)

VALID ENUMERANTS

mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION

Set the Direct3D v-direction texture wrapping mode. See the D3DSAMP_ADDRESSV sampler state description for DirectX 9.

Set the wrap parameter for texture coordinate T. See the GL_TEXTURE_WRAP_T description on the OpenGL glTexParameteri manual page for details.

The standard reset callback sets the AddressV or WrapT state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS

Clamp, Repeat, Wrap: OpenGL 1.0
ClampToEdge: OpenGL 1.2
Border, ClampToBorder: OpenGL 1.3
Mirror, MirroredRepeat: OpenGL 1.4
MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce: support for extension EXT_mirror_clamp.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AddressW, AddressU, AddressW, WrapS, WrapT, WrapR

11.3 AddressW

NAME

AddressW - 3D API W texture addressing mode

USAGE

AddressW = int(mode)
WrapR = int(mode)
VALID ENUMERANTS

mode: Repeat, Wrap, Clamp, ClampToEdge, ClampToBorder, Border, MirroredRepeat, Mirror, MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce

DESCRIPTION

Set the Direct3D w-direction texture wrapping mode. See the D3DSAMP_ADDRESSW sampler state description for DirectX 9.

Set the wrap parameter for texture coordinate R. See the GL_TEXTURE_WRAP_R description on the OpenGL glTexParameteri manual page for details.

The standard reset callback sets the AddressW or WrapR state to int(Wrap).

OPENGL FUNCTIONALITY REQUIREMENTS

Clamp, Repeat, Wrap: OpenGL 1.0
ClampToEdge: OpenGL 1.2
Border, ClampToBorder: OpenGL 1.3
Mirror, MirroredRepeat: OpenGL 1.4
MirrorClamp, MirrorClampToEdge, MirrorClampToBorder, MirrorOnce: support for extension EXT_mirror_clamp.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AddressWi, AddressU, AddressV, WrapS, WrapT, WrapR

11.4 BorderColor

NAME

BorderColor - 3D API texture border color

USAGE

BorderColor = float4(r, g, b, a)

VALID ENUMERANTS

r, g, b, a: floating point values

DESCRIPTION

Set the Direct3D sampler border color. See the GL_TEXTURE_BORDER_COLOR description on the OpenGL glTexParameter manual page for details. See the D3DSAMP_BORDERCOLOR sampler state description for DirectX 9.

The standard reset callback sets the BorderColor state to float4(0, 0, 0, 0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

BorderColor

11.5 CompareFunc

NAME

CompareFunc - 3D API compare function

USAGE

CompareFunc = int(func)

VALID ENUMERANTS

func: Never, Less, LEqual, Equal, Greater, NotEqual, GEqual, Always

DESCRIPTION

Specify the texture comparison function. See the GL_TEXTURE_COMPARE_FUNC description on the OpenGL glTexParameter manual page, the GL_ARB_shadow extension, or EXT_shadow_funcs extension for details.

The standard reset callback sets the CompareFunc state to float(LEqual).

OPENGL FUNCTIONALITY REQUIREMENTS

LEqual, GEqual: OpenGL 1.4 or support for extension GL_ARB_shadow.
all others: OpenGL 1.5 or support for extension GL_EXT_shadowfuncs.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

CompareMode, DepthMode

11.6 CompareMode

NAME

CompareMode - 3D API compare mode

USAGE

CompareMode = int(mode)

VALID ENUMERANTS

mode: None, CompareRToTexture
DESCRIPTION
Specify the texture comparison mode. See the GL_TEXTURE_COMPARE_MODE description on the OpenGL glTexParameter manual page or the GL_ARB_shadow extension for details.

The standard reset callback sets the CompareMode state to float(None).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.4 or support for extension GL_ARB_shadow.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
CompareFunc, DepthMode

11.7 DepthMode

NAME
DepthMode - 3D API depth mode

USAGE
DepthMode = int(mode)

VALID ENUMERANTS
mode: Alpha, Intensity, Luminance

DESCRIPTION
Specify the depth texture mode. See the GL_DEPTH_TEXTURE_MODE description on the OpenGL glTexParameter manual page or the ARB_depth_texture extension for details.

The standard reset callback sets the DepthMode state to float(Luminance).

OPENGL FUNCTIONALITY REQUIREMENTS
OpenGL 1.4 or support for extension ARB_depth_texture.

DIRECT3D FUNCTIONALITY REQUIREMENTS
Not applicable.

SEE ALSO
CompareFunc, CompareMode

11.8 GenerateMipmap

NAME
GenerateMipmap - 3D API texture mipmap generation
**USAGE**

GenerateMipmap = bool(enable)

**VALID ENUMERANTS**

enable: true, false

**DESCRIPTION**

Set the OpenGL texture mipmap generation for the given texture. See the GL_GENERATE_MIPMAP description on the OpenGL glTexParameter manual page for details.

For a TextureRectangle texture, this state has no effect.

The standard reset callback sets the GenerateMipmap state to bool(false).

**OPENGL FUNCTIONALITY REQUIREMENTS**

OpenGL 1.4 or support for extension SGIS_generate_mipmap.

**DIRECT3D FUNCTIONALITY REQUIREMENTS**

Not applicable.

**SEE ALSO**

Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

### 11.9 LODBias

**NAME**

LODBias - 3D API mip map lod bias

**DESCRIPTION**

LODBias is an alias for MipMapLodBias.

**SEE ALSO**

MipMapLodBias

### 11.10 MagFilter

**NAME**

MagFilter - 3D API mag filter

**USAGE**

MagFilter = int(type)

**VALID ENUMERANTS**

type: Point, Nearest, Linear
DESCRIPTION
Set the sampler magnification filter type. See the GL_TEXTURE_MAG_FILTER description on the OpenGL glTexParameter manual page for details. See the D3DSAMP_MAGFILTER sampler state description for DirectX 9.
The standard reset callback sets the MagFilter state to int(Point).

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
MagFilteri, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MaxMipLevel, MinFilter, MipFilter, MipMapLodBias

11.11 MaxAnisotropy

NAME
MaxAnisotropy - 3D API max anisotropy

USAGE
MaxAnisotropy = int(d3d_max)
MaxAnisotropy = float(gl_max)

VALID ENUMERANTS

d3d_max: integer value
gl_max: floating point value

DESCRIPTION
Set the maximum anisotropy value. See the D3DSAMP_MAXANISOTROPY sampler state description for DirectX 9. See the description of GL_TEXTURE_MAX_ANISOTROPY_EXT in the OpenGL EXT_texture_filter_anisotropic extension for details.
The standard reset callback sets the MaxAnisotropy state to int(1).

OPENGL FUNCTIONALITY REQUIREMENTS
Support for extension EXT_texture_filter_anisotropic.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
MaxAnisotropyi
11.12 MaxMipLevel

NAME
MaxMipLevel - 3D API max mip level

USAGE
MaxMipLevel = int(d3d_max)
MaxMipLevel = float(gl_max)

VALID ENUMERANTS
max: integer value

DESCRIPTION
Set the Direct3D maximum mipmap lod index. See the D3DSAMP_MAXMIPLEVEL sampler state description for DirectX 9. See the GL_TEXTURE_MAX_LOD description on the OpenGL glTexParameter manual page for details.

The standard reset callback sets the MaxMipLevel state to int(0) for DirectX 9 and float(1000) for OpenGL.

OPENGL FUNCTIONALITY REQUIREMENTS
Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS
DirectX 9

SEE ALSO
MaxMipLeveli, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MinFilter, MipFilter, MipMapLodBias

11.13 MinFilter

NAME
MinFilter - 3D API min filter

USAGE
MinFilter = int(type)

VALID ENUMERANTS
type: Anisotropic, GaussianQuad, Linear, LinearMipMapLinear, LinearMipMapNearest, Nearest, Nearest-MipMapLinear, NearestMipMapNearest, None, Point, PyramidalQuad

Anisotropic, GaussianQuad, None, and PyramidalQuad are only supported on Direct3D.
DESCRIPTION

Set the Direct3D sampler minification filter type. See the D3DSAMP_MINFILTER sampler state description for DirectX 9. See the GL_TEXTURE_MIN_FILTER description on the OpenGL glGetParameter manual page for details.

The standard reset callback sets the MinFilter state to int(Point) on Direct3D and to int(NearestMipMapLinear) on OpenGL.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MinFilteri, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MipFilter, MipMapLodBias

11.14 MinMipLevel

NAME

MinMipLevel - 3D API min mip level

USAGE

MinMipLevel = float(min)

VALID ENUMERANTS

min: floating point value

DESCRIPTION

Set the minimum mipmap lod index. See the GL_TEXTURE_MIN_LOD description on the OpenGL glGetParameter manual page or the description of GL_TEXTURE_MIN_LOD_SGIS in the OpenGL extension SGIS_texture_lod for details.

The standard reset callback sets the MinMipLevel state to float(-1000).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.2 or support for extension SGIS_texture_lod.

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable.

SEE ALSO

GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MinFilter, MipFilter, MipMapLodBias
11.15 MipFilter

NAME
MipFilter - 3D API mip filter

USAGE
MipFilter = int(type)

VALID ENUMERANTS

- type: None, Point, Linear, Anisotropic, PyramidalQuad, GaussianQuad

DESCRIPTION
Set the Direct3D minification mipmap filter. See the D3DSAMP_MIPFILTER sampler state description for DirectX 9.

The standard reset callback sets the MipFilter state to int(None).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MipFilteri, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MinFilter, MipMapLodBias

11.16 MipMapLodBias

NAME
MipMapLodBias - 3D API mip map lod bias

USAGE
MipMapLodBias = float(bias)

VALID ENUMERANTS

- bias: floating point value

DESCRIPTION
Set the Direct3D mipmap lod bias. See the D3DSAMP_MIPMAPLODBIAS sampler state description for DirectX 9. See the GL_TEXTURE_LOD_BIAS_EXT description on the OpenGL glTexParameter manual page for details.

The standard reset callback sets the MipMapLodBias state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0
DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

MipMapLodBiasi, GenerateMipmap, Texture, Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MaxMipLevel, MinFilter, MipFilter

11.17 SRGBTexture

NAME

SRGBTexture - 3D API gamma correction value

USAGE

SRGBTexture = float(tex)

VALID ENUMERANTS

tex: floating point value

DESCRIPTION

Specify whether gamma correction should be used. See the D3DSAMP_SRGBTEXTURE sampler state description for DirectX 9.

The standard reset callback sets the SRGBTexture state to float(0).

OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

11.18 Texture

NAME

Texture - 3D API texture

USAGE

Texture = texture(t)

VALID ENUMERANTS

t: integer value

DESCRIPTION

Specify the texture to be used. See the IDirect3DDevice9::SetTexture method for details.

The standard reset callback sets the Texture state to texture(0).
OPENGL FUNCTIONALITY REQUIREMENTS

Not applicable.

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

Texture1D, Texture1DEnable, Texture2D, Texture2DEnable, Texture3D, Texture3DEnable, TextureCubeMap, TextureCubeMapEnable, TextureEnvMode, TextureFactor, TextureRectangle, TextureRectangleEnable, MagFilter, MipFilter, MipMapLodBias

11.19 WrapR

NAME

WrapR - 3D API W texture addressing mode

DESCRIPTION

WrapR is an alias for AddressW.

SEE ALSO

AddressW

11.20 WrapS

NAME

WrapS - 3D API U texture addressing mode

DESCRIPTION

WrapS is an alias for AddressU.

SEE ALSO

AddressU

11.21 WrapT

NAME

WrapT - 3D API V texture addressing mode

DESCRIPTION

WrapT is an alias for AddressV.

SEE ALSO

AddressV
12.1 Cg 1.2 Runtime API Additions

Version 1.2 of the Cg runtime adds a number of new capabilities to the existing set of functionality from previous releases. These new features include functionality that make it possible to write programs that can run more efficiently on the GPU, techniques that help hide some of the inherent limitations of some Cg profiles on the GPU, and entrypoints that support new language functionality in the Cg 1.2 release.

Parameter Literalization

The 1.2 Cg runtime makes it possible to denote some of the parameters to a program as having a fixed constant value. This feature can lead to substantially more efficient programs in a number of cases. For example, a program might have a block of code that implements functionality that is only used some of the time:

```c
float4 main(uniform float enableDazzle, ...) : COLOR {
    if (enableDazzle) {
        // do lengthy computation
    }
    else {
        // do basic computation
    }
}
```

Some hardware profiles don’t directly support branching (this includes all of the fragment program profiles supported in this release), and have to handle code like the program by effectively following both sides of the if() test. (They still compute the correct result in the end, just not very efficiently.)

However, if the “enableDazzle” parameter is marked as a literal parameter and a value is provided for it, the compiler can generate an optimized version of the program with the knowledge of “enableDazzle”‘s value, just generating GPU code for one of the two cases. This can lead to substantial performance improvements. This feature also makes it easier to write general purpose shaders with a wide variety of supported functionality, while only paying the runtime cost for the functionality provided.

This feature is also useful for parameters with numeric values. For example, consider a shader that implements a diffuse reflection model:

```c
float4 main(uniform float3 lightPos, uniform float3 lightColor, uniform float3 Kd, float3 pos : TEXCOORD0, float3 normal : TEXCOORD1) : COLOR {
    return Kd*lightColor*max(0., dot(normalize(lightPos-pos), normal));
}
```
If the “lightColor” and “Kd” parameters are set to literals, it is possible for the compiler to compute the product “Kd * lightColor” once, rather than once each time the program executes.

Given a parameter handle, the cgSetParameterVariability() entrypoint sets the variability of a parameter:

```c
void cgSetParameterVariability(CGparameter param, CGenum vary);
```

To set it to a literal parameter, the CG_LITERAL enumerant should be passed as the second parameter.

After a parameter has set to be a literal, the following routines should be used to set the parameter’s value.

```c
void cgSetParameter1f(CGparameter param, float x);
void cgSetParameter2f(CGparameter param, float x, float y);
void cgSetParameter3f(CGparameter param, float x, float y, float z);
void cgSetParameter4f(CGparameter param, float x, float y, float z, float w);
void cgSetParameter1d(CGparameter param, double x);
void cgSetParameter2d(CGparameter param, double x, double y);
void cgSetParameter3d(CGparameter param, double x, double y, double z);
void cgSetParameter4d(CGparameter param, double x, double y, double z, double w);
```

After a parameter has been set to be a literal, or after the value of a literal parameter has been changed, the program must be compiled and loaded into the GPU, regardless of whether it had already been compiled. This issue is discussed further in the section on program recompilation below.

**Array Size Specification**

The Cg 1.2 language also adds support for “unsized array” variables; programs can be written to take parameters that are arrays with an indeterminate size. The actual size of these arrays is then set via the Cg runtime. This feature is useful for writing general-purpose shaders with a minimal performance penalty.

For example, consider a shader that computes shading given some number of light sources. If the information about each light source is stored in a struct LightInfo, the shader might be written as:

```c
float4 main(LightInfo lights[], ...) : COLOR
{
    float4 color = float4(0, 0, 0, 1);
    for (i = 0; i < lights.length; ++i)
    {
        // add lights[i]'s contribution to color
    }
    return color;
}
```

The runtime can then be used to set the length of the lights[] array (and then to initialize the values of the LightInfo structures.) As with literal parameters, the program must be recompiled and reloaded after a parameter’s array size is set or changes.
These two entrypoints set the size of an unsized array parameter referenced by the given parameter handle. To set the size of a multidimensional unsized array, all of the dimensions’ sizes must be set simultaneously, by providing them all via the pointer to an array of integer values.

```c
void cgSetArraySize(CGparameter param, int size);
void cgSetMultiDimArraySize(CGparameter param, const int *sizes);
```

XXX what happens if these are called with an already-sized array?? XXX

To get the size of an array parameter, the `cgGetArraySize()` entrypoint can be used.

```c
int cgGetArraySize(CGparameter param, int dimension);
```

### Program Recompilation at Runtime

The Cg 1.2 runtime environment will allow automatic and manual recompilation of programs. This functionality is useful for multiple reasons:

* **Changing variability of parameters**
  Parameters may be changed from uniform variability to literal variability as described above.

* **Changing value of literal parameters**
  Changing the value of a literal parameter will require recompilation since the value is used at compile time.

* **Resizing parameter arrays**
  Changing the length of a parameter array may require recompilation depending on the capabilities of the profile of the program.

* **Binding sub-shader parameters**
  Sub-shader parameters are structures that overload methods that need to be provided at compile time; they are described below. Binding such parameters to program parameters will require recompilation.

Recompilation can be executed manually by the application using the runtime or automatically by the runtime.

The entry point:

```c
void cgCompileProgram(CGprogram program);
```

causes the given program to be recompiled, and the function:

```c
CGbool cgIsProgramCompiled(CGprogram program);
```

returns a boolean value indicating whether the current program needs recompilation.

By default, programs are automatically compiled when `cgCreateProgram()` or `cgCreateProgramFromFile()` is called. This behavior can be controlled with the entry point:

```c
void cgSetAutoCompile(CGcontext ctx, CGenum flag);
```

Where flag is one of the following three enumerants:

* **CG_COMPILE_MANUAL**
  With this method the application is responsible for manually recompiling a program. It may check to see if a program requires recompilation with the entry point `cgIsProgramCompiled()`. `cgCompileProgram()` can then be used to force compilation.

* **CG_COMPILE_IMMEDIATE**
**CG_COMPILE_IMMEDIATE** will force recompilation automatically and immediately when a program enters an uncompiled state.

* **CG_COMPILE_LAZY**
  
  This method is similar to **CG_COMPILE_IMMEDIATE** but will delay program recompilation until the program object code is needed. The advantage of this method is the reduction of extraneous recompilations. The disadvantage is that compile time errors will not be encountered when the program is enters the uncompiled state but will instead be encountered at some later time.

For programs that use features like unsized arrays that can not be compiled until their array sizes are set, it is good practice to change the default behavior of compilation to **CG_COMPILE_MANUAL** so that cgCreateProgram() or cgCreateProgramFromFile() do not unnecessarily encounter and report compilation errors.

**Shared Parameters (context global parameters)**

Version 1.2 of the runtime introduces parameters that may be shared across programs in the same context via a new binding mechanism. Once shared parameters are constructed and bound to program parameters, setting the value of the shared parameter will automatically set the value of all of the program parameters they are bound to.

Shared parameters belong to a **CGcontext** instead of a **CGprogram**. They may be created with the following new entry points:

- `CGparameter cgCreateParameter(CGcontext ctx, CGtype type);`
- `CGparameter cgCreateParameterArray(CGcontext ctx, CGtype type, int length);`
- `CGparameter cgCreateParameterMultiDimArray(CGcontext ctx, CGtype type, int dim, const int *lengths);`

They may be deleted with:

```
void cgDestroyParameter(CGparameter param);
```

After a parameter has been created, its value should be set with the cgSetParameter*() routines described in the literalization section above.

Once a shared parameter is created it may be associated with any number of program parameters with the call:

```
void cgConnectParameter(CGparameter from, CGparameter to);
```

where “from” is a parameter created with one of the cgCreateParameter() calls, and “to” is a program parameter.

Given a program parameter, the handle to the shared parameter that is bound to it (if any) can be found with the call:

```
CGparameter cgGetConnectedParameter(CGparameter param);
```

It returns NULL if no shared parameter has been connected to “param”.

There are also calls that make it possible to find the set of program parameters to which a given shared parameter has been connected to. The entry point:

```
int cgGetNumConnectedToParameters(CGparameter param);
```

returns the total number of program parameters that “param” has been connected to, and the entry point:

```
CGparameter cgGetConnectedToParameter(CGparameter param, int index);
```

can be used to get CGparameter handles for each of the program parameters to which a shared parameter is connected.

A shared parameter can be unbound from a program parameter with:

```
void cgDisconnectParameter(CGparameter param);
```
The context in which a shared parameter was created can be returned with:

```c
CGcontext cgGetParameterContext(CGparameter param);
```

And the entrypoint:

```c
CGbool cgIsParameterGlobal(CGparameter param);
```

can be used to determine if a parameter is a shared (global) parameter.

**Shader Interface Support**

From the runtime’s perspective, shader interfaces are simply struct parameters that have a `CGtype` associated with them. For example, if the following Cg code is included in some program source compiled in the runtime:

```c
interface FooInterface
{
    float SomeMethod(float x);
}

struct FooStruct : FooInterface
{
    float SomeMethod(float x);
    {
        return(Scale * x);
    }

    float Scale;
};
```

The named types `FooInterface` and `FooStruct` will be added to the context. Each one will have a unique `CGtype` associated with it. The `CGtype` can be retrieved with:

```c
CGtype cgGetNamedUserType(CGprogram program, const char *name);
int cgGetNumUserTypes(CGprogram program);
CGtype cgGetUserType(CGprogram program, int index);
CGbool cgIsParentType(CGtype parent, CGtype child);
CGbool cgIsInterfaceType(CGtype type);
```

Once the `CGtype` has been retrieved, it may be used to construct an instance of the struct using `cgCreateParameter`. It may then be connected to a program parameter of the parent type (in the above example this would be `FooInterface`) using `cgConnectParameter`.

Calling `cgGetParameterType` on such a parameter will return the `CG_STRUCT` to keep backwards compatibility with code that recurses parameter trees. In order to obtain the enumerant of the named type the following entry point should be used:

```c
CGtype cgGetParameterNamedType(CGparameter param);
```

The parent types of a given named type may be obtained with the following entry points:

```c
int cgGetNumParentTypes(CGtype type);
CGtype cgGetParentType(CGtype type, int index);
```

If Cg source modules with differing definitions of named types are added to the same context, an error will be thrown.

XXX update for new scoping/context/program local definitions stuff XXX

**Updated Parameter Management Routines**

XXX where should these go?
Some entrypoints from before have been updated in backwards compatible ways

```c
CGparameter cgGetFirstParameter(CGprogram program, CGenum name_space);
CGparameter cgGetFirstLeafParameter(CGprogram program, CGenum name_space);
```

like cgGetNamedParameter, but limits search to the given name_space (CG_PROGRAM or CG_GLOBAL)...

```c
CGparameter cgGetNamedProgramParameter(CGprogram program, CGenum name_space,
                                        const char *name);
```

### 12.2 Cg

**NAME**

Cg - A multi-platform, multi-API C-based programming language for GPUs

**DESCRIPTION**

Cg is a high-level programming language designed to compile to the instruction sets of the programmable portions of GPUs. While Cg programs have great flexibility in the way that they express the computations they perform, the inputs, outputs, and basic resources available to those programs are dictated by where they execute in the graphics pipeline. Other documents describe how to write Cg programs. This document describes the library that application programs use to interact with Cg programs. This library and its associated API is referred to as the Cg runtime.

**DOCUMENTATION ORGANIZATION**

- Cg Topics
- Cg Language Specification
- Cg Commands
- Cg Core Runtime API
- Cg OpenGL Runtime API
- Cg Direct3D11 Runtime API
- Cg Direct3D10 Runtime API
- Cg Direct3D9 Runtime API
- Cg Direct3D8 Runtime API
- Cg Profiles
- Cg Standard Library Routines
- CgFX States
- CgFX Sampler States

**SEE ALSO**

*Cg Language Specification, cgc, cgCreateContext, cgDestroyContext*
12.3 glut

ABSTRACT

GLUT provides a cross-platform window system API for writing OpenGL examples and demos. For this reason, the Cg examples packaged with the Cg Toolkit rely on GLUT.

WINDOWS INSTALLATION

The Cg Toolkit installer for Windows provides a pre-compiled 32-bit (and 64-bit if selected) versions of GLUT. GLUT is provided both as a Dynamic Link Library (DLL) and a static library.

The GLUT DLL is called glut32.dll and requires linking against glut32.lib. These 32-bit versions are typically installed at:

- c:\Program Files\NVIDIA Corporation\Cg\bin\glut32.dll
- c:\Program Files\NVIDIA Corporation\Cg\lib\glut32.lib

The 64-bit (x64) versions are installed at:

- c:\Program Files\NVIDIA Corporation\Cg\bin.x64\glut32.dll
- c:\Program Files\NVIDIA Corporation\Cg\lib.x64\glut32.lib

As with any DLL in Windows, if you link your application with the GLUT DLL, running your application requires that glut32.dll can be found when executing GLUT.

Alternatively you can link statically with GLUT. This can easily be done by defining the GLUT_STATIC_LIB preprocessor macro before including GLUT's <GL/glut.h> header file. This is typically done by adding the -DGLUT_STATIC_LIB option to your compiler command line. When defined, a #pragma in <GL/glut.h> requests the linker link against glutstatic.lib instead of glut32.lib.

The 32-bit and 64-bit versions of the GLUT static library are installed at:

- c:\Program Files\NVIDIA Corporation\Cg\lib\glutstatic.lib
- c:\Program Files\NVIDIA Corporation\Cg\lib.x64\glutstatic.lib

SEE ALSO

TBD

12.4 Mac OS X

ABSTRACT

The Cg runtime is available as a framework for Apple OS X versions 10.5 (Leopard) and onwards.
ISSUES

APPLE APP STORE DEPLOYMENT

The Apple App store won’t accept binaries that include Power PC support. It is necessary to remove the Power PC support from the Cg framework for App Store purposes.

Using the Apple lipo tool manipulating universal files:

1. Check for Cg framework architecture support

   $ lipo -i /Library/Frameworks/Cg.framework/Cg
   Architectures in the fat file: /Library/Frameworks/Cg.framework/Cg are: ppc7400 i386 x86_64
   $

2. Remove the ppc7400 support

   $ lipo -remove ppc7400 /Library/Frameworks/Cg.framework/Cg -output Cg
   $

3. Validate the result

   $ lipo -i Cg
   Architectures in the fat file: Cg are: i386 x86_64
   $

The resulting Intel-only Cg framework binary ought to be acceptable to the Apple App Store.

SEE ALSO

TBD

12.5 Trace

TOPIC

Trace - API Trace for Cg, CgGL, OpenGL and GLUT

INTRODUCTION

The NVIDIA Cg Toolkit provides trace wrapper libraries for logging API calls to Cg, CgGL, OpenGL and GLUT libraries. The log includes nested function call information and function parameter and return values. The logged data can be useful for debugging, trouble shooting and reporting issues.

The libraries are experimental and require some software development expertise.

REQUIREMENTS AND LIMITATIONS

Trace supports Cg version 2.2.010 (October 2009) and onwards. Cg, CgGL, OpenGL, and GLUT trace libraries are included.

CgD3D9, CgD3D10 and CgD3D10 are not currently supported.

Cg and GLUT are supported for OSX, but currently not OpenGL.

Trace captures calls and parameters, but not the contents of files referenced by calls such as cgCreateProgramFromFile.

Archive the logs, .cg and .cgfx files together into a .zip or .tgz for later reference.
BUILDING

The trace library source code, makefile and Visual Studio projects are located in the examples/Tools/trace directory of the Cg Toolkit installation. Pre-built binaries are also included.

The trace library components are as follows.

```
trace.c  }
traceOutput.c }  ----> trace library
traceFilter.c }
traceTime.c  }
b64.c  }
traceCg.c  }  ----> Cg wrapper library
traceCgGL.c }  ----> CgGL wrapper library
traceGLUT.c }  ----> GLUT wrapper library
traceGL.c  }
traceGLX.c }  ----> GL wrapper library
traceWGL.c  }
```

DEPLOYMENT

ENVIRONMENT VARIABLES

The `CG_TRACE_FILE` and `CG_TRACE_ERROR` environment variables specify paths to the trace log and error log. The two file names can be the same. Otherwise `stdout` and `stderr` are used.

The `CG_TRACE_CG_LIBRARY` and `CG_TRACE_CGGL_LIBRARY` environment variables specify the path of the target Cg and CgGL libraries. The `CG_TRACE_GL_LIBRARY` environment variable specifies the path of the target OpenGL library. The `CG_TRACE_GLUT_LIBRARY` environment variable specifies the path of the target OpenGL library.

Otherwise, default system locations are used.

The `CG_TRACE_ENABLE` environment variable enables or disables trace output. By default trace output is enabled.

The `CG_TRACE_BASE64` environment variable specifies base64 encoding for binary data such as float, double and byte arrays. A zero integer value disables base64 encoding for floats, doubles and byte arrays. By default base64 encoding is enabled.

The `CG_TRACE_BLOB_LIMIT` environment variable limits the size of raw data logged. Just the pointer value is stored for parameters that exceed the limit. This variable is typically set to avoid capturing texture image data. By default there is no limit and all parameter data is logged.

The `CG_TRACE_TIMESTAMP` environment variable specifies timestamp logging for each call. Any non-zero integer value enables timestamp tracing. By default timestamp logging is disabled.

Recommended settings:

```
CG_TRACE_FILE traceLog.txt
CG_TRACE_ERROR traceError.txt
```

WINDOWS

Select a trace directory to copy the trace libraries to. This can be the same as the application directory. The directory of the target .exe is recommended.

Copy `trace.dll` to the trace directory. The API-specific trace libraries depend on `trace.dll`. The other trace libraries are optional.

Optionally copy trace `cg.dll` and `cgGL.dll` to the trace directory. The `CG_BIN_PATH` (32-bit) or `CG_BIN64_PATH` (64-bit) environment variables are used unless `CG_TRACE_CG_LIBRARY` or `CG_TRACE_CGGL_LIBRARY` are defined.
Optionally copy trace `opengl32.dll` to the trace directory.

Optionally copy trace `glut32.dll` to the trace directory.

**LINUX and SOLARIS**

Set the `LD_LIBRARY_PATH` environment variable to the directory containing the trace libraries. The API-specific trace libraries depend on `libtrace.so`. The other trace libraries are optional.

**OSX**

Set the `DYLD_LIBRARY_PATH` environment variable to the directory containing the trace libraries. These are installed to `/Developer/NVIDIA/Cg/examples/Tools/trace` by default. The API-specific trace libraries depend on `libtrace.dylib`. The Cg and GLUT trace frameworks are both optional. The symlinks `libCg.dylib` and `libGLUT.dylib` specify the full path of the native frameworks.

**CREDITS AND LICENCES**

The core trace library uses the base64 C implementation of Base64 Content-Transfer-Encoding standard (also known as RFC1113) by Bob Trower, Trantor Standard Systems Inc.

The Cg trace library uses the uthash C hash table implementation by Troy D. Hanson.

### 12.6 win64

**TOPIC**

**win64** - using Cg with 64-bit Windows

**ABSTRACT**

The Cg Toolkit for Windows installs versions of the Cg compiler and runtime libraries for both 32-bit (x86) and 64-bit (x64) compilation. This topic documents how to use Cg for 64-bit Windows.

**64-BIT INSTALLATION**

The Cg Toolkit installer (CgSetup.exe) installs the 32-bit version of the Cg compiler and the Cg runtime libraries by default. To install the 64-bit support, you must check the component labeled “Files to run and link 64-bit (x64) Cg-based applications” during your installation.

If you’ve forgotten to install the 64-bit component, you can re-run the Cg Toolkit installer and check the 64-bit component.

**EXAMPLES**

The Cg Toolkit includes Visual Studio .NET 2003 projects intended to compile 64-bit versions of the Cg Toolkit examples.

These project files match the pattern `*_x64.vcproj`

The solution files that collect these projects matches the pattern `*_x64.sln`

To use these project files with Visual Studio .NET 2003, you must also install the latest Windows Platform SDK to obtain 64-bit compiler tools and libraries.

Once the Platform SDK is installed, from the Start menu navigate to start a Windows shell for the 64-bit Windows Build Environment. This shell is started with the correct environment variables (Path, Include, and Lib) for the 64-bit compiler tools and libraries.
Now run devenv.exe with the /useenv command line option that forces Visual Studio to pick up Path, Include, and Lib settings from the shell’s environment. When the Visual Studio IDE appears, select File->Open->Project... and locate one of the *\_x64.sln files for the Cg examples. These are usually under:

c:\Program Files\NVIDIA Corporation\Cg\examples

When you open a *\_x64.vcproj solution, it references a number of *\_x64.vcproj projects. These have a “Debug x64” and “Release x64” configuration to build.

**HINTS**

Remember to link with BufferOverflowU.lib because of the /GS option to help detect string overflow runtime errors because Microsoft has enabled this option by default in its 64-bit compilers. See:

http://support.microsoft.com/?id=894573

**IA64 SUPPORT**

The Cg Toolkit does not provide 64-bit support for Intel’s Itanium architecture.

**SEE ALSO**

TBD
CHAPTER
THIRTEEN

CG TOOLKIT RELEASES

13.1 cg_2_1_0009

NAME

cg_2_1_0009 - Cg Toolkit 2.1 August 2008 (beta)

SUMMARY

• New translation profiles for DirectX10 Shader Model 4
• Shader source virtual file system for compilation
• Callback for providing shader source included by the compiler
• Improved handling of generic attributes by the GLSL profile
• Performance improvements and bug fixes
• New examples for OpenGL and D3D10

DOWNLOAD

• Windows 32/64-bit installer for Windows XP, Vista and Win7.
• Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
• Linux 32-bit tgz tarball and rpm for RedHat.
• Linux 64-bit tgz tarball and rpm for RedHat.
• Solaris 32-bit tarball.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.2 cg_2_1_0012

NAME

cg_2_1_0012 - Cg Toolkit 2.1 October 2008
SUMMARY

- Added more ‘clip’ functions for improved compatibility with HLSL
- TEXCOORD8-15 now available in vs/ps 3.0 profiles
- Solaris release in Sun package format
- Performance improvements and bug fixes
- Fixed sampler initialization by the GLSL profile
- Fixed handling of integer parameters values
- Fixed handling of interpolation modifiers

DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball and rpm for RedHat.
- Linux 64-bit tgz tarball and rpm for RedHat.
- Solaris 32-bit tarball.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.3 cg_2_1_0016

NAME

cg_2_1_0016 - Cg Toolkit 2.1 November 2008

SUMMARY

- Fixed crash when cgGL routines were called without a current GL context
- Fixed a bug in compiling CgFX files with multiple geometry shaders
- Fixed D3D10 runtime’s handling of the parameter buffer (cbuffer)
- D3D10 runtime now correctly deals with true integer parameters
- Shared parameter arrays were not being updated correctly
- Corrected some minor issues in the Sun package file

DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball and rpm for RedHat.
- Linux 64-bit tgz tarball and rpm for RedHat.
- Solaris 32-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.4 cg_2_1_0017

NAME

cg_2_1_0017 - Cg Toolkit 2.1 February 2009

SUMMARY

- Fixed a potential corruption issue when creating programs of type CG_OBJECT
- Fixed a problem in parsing of #included shader source files
- Repaired cgGLSetParameter functions for varying inputs of vertex programs

DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball and rpm for RedHat.
- Linux 64-bit tgz tarball and rpm for RedHat.
- Solaris 32-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.5 cg_2_2_0004

NAME

cg_2_2_0004 - Cg Toolkit 2.2 February 2009 (beta)

SUMMARY

- Added DirectX10 and GLSL geometry profiles (gs_4_0 AND glslg)
- Support for “latest” profile keyword in CgFX compile statements
- Additional API routines
- Support for pack_matrix() pragma
- Arrays of shaders can now be used in CgFX files
- Libraries for 64 bit Solaris development
- Migrated the OpenGL examples onto GLEW
- New examples
- Updated reference manual pages for new profiles and entry points

DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball and rpm for RedHat.
- Linux 64-bit tgz tarball and rpm for RedHat.
• Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.6 cg_2_2_0006

NAME
cg_2_2_0006 - Cg Toolkit 2.2 April 2009

SUMMARY

• Added DirectX10 and GLSL geometry profiles (gs_4_0 AND glslg)
• Support for “latest” profile keyword in CgFX compile statements
• Additional API routines
• Support for pack_matrix() pragma
• Arrays of shaders can now be used in CgFX files
• Libraries for 64 bit Solaris development
• Migrated the OpenGL examples onto GLEW
• New examples
• Updated reference manual pages for new profiles and entry points

DOWNLOAD

• Windows 32/64-bit installer for Windows XP, Vista and Win7.
• Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
• Linux 32-bit tgz tarball and rpm for RedHat.
• Linux 64-bit tgz tarball and rpm for RedHat.
• Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.7 cg_2_2_0010

NAME
cg_2_2_0010 - Cg Toolkit 2.2 October 2009
SUMMARY

• Allow compiler options in effect compile statements. e.g. VertexProgram = compile vp40 ‘-po PosInv=1’ shader();
• Better performance when running on multicore CPUs
• Choosing the “latest” profile is now deferred until effect validation
• Improved the inverse matrix computation in cgGLSetStateMatrixParameter
• Better memory behavior when a program is repeatedly recompiled
• Fixed an issue when using PSIZE semantic with ps_3_0 and ps_4_0 profiles
• cgCombinePrograms now works with CG_OBJECT programs
• cgGetNextProgram was always returning 0
• Fixed a problem with effect parameters and cgGLGetTextureEnum
• Allow comments prior to the shader version in D3D asm blocks of an effect
• HLSL10: Mark globally scoped temporaries ‘static’ to keep them out of constant buffers
• HLSL10: Allow any semantic for varyings provided the semantics match between stages
• HLSL10: Fix handling of TEXUNITn semantic
• HLSL10: Added support for arrays of samplers
• HLSL10: Empty structs for uniform parameters crashed the D3D compiler
• Fixed a problem when connecting parameters of type string
• Corrected issues in the fp20 profile on Solaris
• Now using MesaGLUT-7.5 for GLUT on Windows

DOWNLOAD

• Windows 32/64-bit installer for Windows XP, Vista and Win7.
• Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
• Linux 32-bit tgz tarball and rpm for RedHat.
• Linux 64-bit tgz tarball and rpm for RedHat.
• Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.8 cg_2_2_0017

NAME

cg_2_2_0017 - Cg Toolkit 2.2 February 2010

SUMMARY

• Require EXT_gpu_shader4 in GLSL when using bit shift/mask instructions
• Modified example gs_simple to explicitly use the GLSL profiles if supported
• HLSL semantic VFACE is now accepted as an alias for semantic FACE
• Improved our handling of extensions on older versions of OpenGL
• Various performance improvements
• Enhanced cgfxcat to work for program files as well as effect files
• Fixed some compiler crashes with malformed shaders
• Fixed a crash in cgGetParameterBufferIndex and cgGetParameterBufferOffset
• Fixed a bug in cgGetPassProgram for combined programs
• Fixed a problem with geometry shaders on Solaris

DOWNLOAD

• Windows 32/64-bit installer for Windows XP, Vista and Win7.
• Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
• Linux 32-bit tgz tarball and rpm for RedHat.
• Linux 64-bit tgz tarball and rpm for RedHat.
• Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.9 cg_3_0_0007

NAME

cg_3_0_0007 - Cg Toolkit 3.0 July 2010

SUMMARY

• OpenGL GPU Program5 profiles
• DirectX11 Shader Model 5 profiles
• Support for tessellation programs for OpenGL and DirectX 11 hardware such as NVIDIA’s new Fermi GPU architecture
• Numerous examples for new Direct3D and OpenGL capabilities
• Support for up to 32 texture units
• Unbind routines for D3D programs
• CgFX buffer routines
• Dependent parameter routines for CgFX shader arrays
• Shadow versions of texBLAHproj functions in the hlsI10f profile
• Improved evaluation engine for expressions in CgFX files
• Updated reference manual for new profiles and entry points

DOWNLOAD

• Windows 32/64-bit installer for Windows XP, Vista and Win7.
• Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
• Linux 32-bit tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
• Linux 64-bit tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
• Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013
13.10  cg_3_0_0015

NAME

cg_3_0_0015 - Cg Toolkit 3.0 November 2010

SUMMARY

- Vertex shaders aren’t disabled by tessellation shader state assignments
- DirectX feature levels 9 and 10 now work with cgD3D11
- cgCombinePrograms respects the auto compile flag
- The hls10g profile now supports structures in AttribArray
- cgCopyProgram works once again!
- Fixed a bug when compound-assignment operators (+=, *=, etc.) were used on column major matrices
- tex2Dsize will now work for shadow samplers
- Resolved a problem with geometry shaders on OSX
- Corrected the OSX Framework executable path
- Fixed crash when connecting API created parameters to precompiled (CG_OBJECT) program parameters
- No longer try to use typedef keyword in generated GLSL code
- Fixed a problem in the modf routine
- Various documentation updates

DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
- Linux 64-bittgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
- Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.11  cg_3_0_0016

NAME

cg_3_0_0016 - Cg Toolkit 3.0 February 2011

SUMMARY

- Improved DX11 tessellation support
- Resolved an issue with nearly identical user defined types
- Resolved an issue with default values from unreferenced uniform parameters
- Support setting matrices beyond 96 float constants in the vp30 profile
- Application supplied compilation options now override those set by cgGLSetOptimalOptions
- Improved support for ‘const’ variables in the GLSL profiles
- Added sampler state documentation
DOWNLOAD

- Mac OS X ppc/i386/x86_64 dmg for Tiger, Leopard and Snow Leopard.
- Linux 32-bit tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
- Linux 64-bit tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
- Solaris 32/64-bit package.

SEE ALSO

2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
3.0.0007, 3.0.0015, 3.0.0016
3.1.0010, 3.1.0013

13.12 cg_3_1_0010

NAME

cg_3_1_0010 - Cg Toolkit 3.1 February 2012

SUMMARY

Improvements

- Added Cg language support for uniform buffers.
- Added OpenGL Unified Buffer Object (UBO) support for buffers.
- Added OpenGL GLSL version 110 and 120 translation support.
- New tessellation examples added.
- New uniform buffer examples added.
- VC10 projects added for examples.

Documentation

Note: The Cg Users Manual has not been updated for this release.

- Updated reference manual for new entry points.
- Updated Cg standard library documentation.
- Release history documentation added.

Bug Fixes

- Fixed buffer emulation in DX9, arbvp1 and arbfp1 profiles.
- Fixed f3dtext2D standard library issue.
- Added samplerRBUF to standard library.
- Fixed issues with implicit cast of int to uint.
- Fixed cgc compiler issue for $COL0.
- Fixed cgGetParameterResourceIndex to return the proper resource index.
Fixed var bindings info in *GLSL* and *fp20* profiles.
Improved the DX10 and DX11 examples.
Added support for non-uniform matrix argument in the *glslf* profile.
Increased *GLSL* output color array to 8.
Added **CENTROID**, **FLAT** and **NOPERSPECTIVE** semantic support to *gp4* and *gp5* profiles.
Fixed an issue for int and uint in *GLSL* profile versions before 130.
Fixed unsigned integer uniform usage in *GLSL* profiles.
Fixed **SAMPLEPOS** usage in the *gp5fp* profile.
Fixed the tex2dbias usage in the *arbfp1* profile.

**New API**

Below is the complete list of the new entry points in Cg 3.1. See the Cg Reference Manual for further details.

```c

cgGetFirstUniformBufferParameter
cgGetNamedEffectUniformBuffer
cgGetNamedProgramUniformBuffer
cgGetNamedUniformBufferParameter
cgGetProfileSibling
cgGetProgramOutputVertices
cgGetUniformBufferBlockName
cgGetUniformBufferParameter
cgIsBuffer
cgSetProgramOutputVertices
cgSetUniformBufferParameter
cgGLCreateBufferFromObject
cgGLDetectGLSLVersion
cgGLGetContextGLSLVersion
cgGLGetContextOptimalOptions
cgGLGetGLSLVersion
cgGLGetGLSLVersionString
cgGLSetContextGLSLVersion
cgGLSetContextOptimalOptions
cgD3D10CreateBufferFromObject
cgD3D10CreateBuffer
cgD3D10GetBufferObject
cgD3D11CreateBufferFromObject
cgD3D11CreateBuffer
cgD3D11GetBufferObject
```
COMPATIBILITY NOTES

Although the 3.1 release of Cg is generally compatible with previous releases, several improvements and other changes may affect existing applications. This section details these potential compatibility issues.

Cg 3.1 supports defining constant blocks with the uniform keyword. Use of a BUFFER semantic on simple variables and structs has been deprecated. By default cgc will now issue a warning when the BUFFER semantic is used (and an error when BUFFER is used on a struct with a GLSL profile). The -no_uniform_blocks compiler flag can be used to disable these warnings and errors, but it also removes support for the new uniform keyword method of defining constant blocks.

There aren’t any other known compatibility issues with programs written against Cg 3.0. For programs written against Cg 2.2 or earlier, refer to the Compatibility Notes section of the release notes for Cg 3.0.

Deprecated

- Manual pages have been removed.
- D3D8 support has been removed.
- Mac OS X 10.4 support has been removed.

DOWNLOAD

- Windows x86/x64 installer for Windows XP, Vista and Win7.
- Mac OS X ppc/i386/x64 dmg for Leopard, Snow Leopard and Lion.
- Linux x86 tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.
- Linux x86-64 tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.

SEE ALSO

- 2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
- 2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
- 3.0.0007, 3.0.0015, 3.0.0016
- 3.1.0010, 3.1.0013

13.13 cg_3_1_0013

NAME

cg_3_1_0013 - Cg Toolkit 3.1 April 2012

SUMMARY

The Cg Toolkit allows developers to write and run Cg programs using a wide variety of hardware and OS platforms and graphics APIs. Originally released in December 2002, the Toolkit now supports over 30 different DirectX and OpenGL profile targets. It provides a compiler for the Cg language, runtime libraries to use with the OpenGL and DirectX graphics APIs, support for CgFX effect files, example applications, and extensive documentation.

CONTENTS

Cg Runtime libraries

The Cg core runtime library for managing parameters and loading programs.

The CgGL runtime library for OpenGL based applications.
The CgD3D9 runtime library for DirectX 9 based applications.
The CgD3D10 runtime library for DirectX 10 based applications.
The CgD3D11 runtime library for DirectX 11 based applications.

Supported Profiles

OpenGL

- \textit{gp5tcp} NV\_tessellation\_program5 control program.
- \textit{gp5tep} NV\_tessellation\_program5 evaluation program.
- \textit{gp5gp} NV\_geometry\_program5.
- \textit{gp5vp} NV\_vertex\_program5.
- \textit{gp5fp} NV\_fragment\_program5.
- \textit{gp4gp} NV\_geometry\_program4.
- \textit{gp4vp} NV\_vertex\_program4.
- \textit{gp4fp} NV\_fragment\_program4.
- \textit{glslg} OpenGL Shading Language (GLSL) for OpenGL 2.0 geometry shader.
- \textit{glslv} OpenGL Shading Language (GLSL) for OpenGL 2.0 vertex shader.
- \textit{glslf} OpenGL Shading Language (GLSL) for OpenGL 2.0 fragment shader.
- \textit{arbvp1} ARB\_vertex\_program 1.0.
- \textit{arbfp1} ARB\_fragment\_program 1.0.
- \textit{vp40} ARB\_vertex\_program + NV\_vertex\_program2 option.
- \textit{fp40} ARB\_fragment\_program + NV\_fragment\_program2 option.
- \textit{vp30} NV\_vertex\_program 2.0.
- \textit{fp30} NV\_fragment\_program 1.0.
- \textit{vp20} NV\_vertex\_program 1.0.
- \textit{fp20} NV\_register\_combiners and NV\_texture\_shader.

DirectX 11.0

- \textit{ds\_5\_0} HLSL11 Domain Shader.
- \textit{hs\_5\_0} HLSL11 Hull Shader.
- \textit{gs\_5\_0} HLSL11 Geometry Shader.
- \textit{vs\_5\_0} HLSL11 Vertex Shader.
- \textit{ps\_5\_0} HLSL11 Fragment Shader.

DirectX 10.0

- \textit{gs\_4\_0} HLSL10 Geometry Shader.
- \textit{vs\_4\_0} HLSL10 Vertex Shader.
- \textit{ps\_4\_0} HLSL10 Fragment Shader.

DirectX 9.0c
**hlslv** HLSL9 Vertex Shader.

**hlslf** HLSL9 Fragment Shader.

**vs_3_0** Vertex Shader 3.0.

**ps_3_0** Pixel Shader 3.0.

**DirectX 9**

**vs_2_x** Extended Vertex Shader 2.0.

**ps_2_x** Extended Pixel Shader 2.0.

**vs_2_0** Vertex Shader 2.0.

**ps_2_0** Pixel Shader 2.0.

**vs_1_1** Vertex Shader 1.1.

**ps_1_3** Pixel Shader 1.3.

**ps_1_2** Pixel Shader 1.2.

**ps_1_1** Pixel Shader 1.1.

**IMPROVEMENTS AND BUG FIXES**

**Bug Fixes**

Improved GLSL support for clip semantics.

Fixed a runtime bug concerning gp4 and gp5 geometry programs.

**DOWNLOAD**

Windows x86/x86-64 installer for Windows XP, Vista and Win7.

Mac OS X ppc/i386/x86_64 dmg for Leopard, Snow Leopard and Lion.

Linux x86 tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.

Linux x86-64 tgz tarball, rpm for RedHat and deb for Debian and Ubuntu.

**LICENSE**

The docs directory contains a file **license.pdf** providing a non-exclusive, world-wide, royalty free license for redistributing Cg with your applications. See this license for details.

**Notice**

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS”. NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication or otherwise under any patent or patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. NVIDIA Corporation products are not authorized for use as critical components in life support devices or systems without express written approval of NVIDIA Corporation.
Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation. Microsoft, Windows, the Windows logo, and DirectX are registered trademarks of Microsoft Corporation. OpenGL is a trademark of SGI. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright © 2004-2012 NVIDIA Corporation.

All rights reserved.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com

SEE ALSO

 2.1.0009, 2.1.0012, 2.1.0016, 2.1.0017
 2.2.0004, 2.2.0006, 2.2.0010, 2.2.0017
 3.0.0007, 3.0.0015, 3.0.0016
 3.1.0010, 3.1.0013
INDEX

A
abs, 685
acos, 686
AddressU, 979
AddressUi, 813
AddressV, 979
AddressVi, 813
AddressW, 980
AddressWi, 814
all, 687
AlphaArg0, 814
AlphaArg1, 815
AlphaArg2, 816
AlphaBlendEnable, 816
AlphaFunc, 817
AlphaOp, 817
AlphaRef, 818
AlphaTestEnable, 819
Ambient, 820
AmbientMaterialSource, 819
any, 688
arbfp1, 41
arbvp1, 42
asin, 688
atan, 691
atan2, 690
AutoNormalEnable, 820

B
bitCount, 692
bitfieldExtract, 693
bitfieldInsert, 694
bitfieldReverse, 695
BlendColor, 821
BlendEnable, 822
BlendEquation, 822
BlendEquationSeparate, 823
BlendFunc, 823
BlendFuncSeparate, 824
BlendOp, 825
BlendOpAlpha, 825
BorderColor, 981
BorderColori, 826
BumpEnvLOffset, 827
BumpEnvLScale, 827
BumpEnvMat00, 828
BumpEnvMat01, 828
BumpEnvMat10, 829
BumpEnvMat11, 829

C
ceil, 696
Cg, 996
Cg 1.2 Runtime API Additions, 991
Cg Language Specification, 3
cg_2_1_0009, 1003
cg_2_1_0012, 1003
cg_2_1_0016, 1004
cg_2_1_0017, 1005
cg_2_2_0004, 1005
cg_2_2_0006, 1006
cg_2_2_0010, 1006
cg_2_2_0017, 1007
cg_3_0_0007, 1008
cg_3_0_0015, 1008
cg_3_0_0016, 1009
cg_3_1_0010, 1010
cg_3_1_0013, 1012
cgAddStateEnumerant, 137
cge, 27
cgCallStateResetCallback, 138
cgCallStateSetCallback, 138
cgCallStateValidateCallback, 139
cgCombinePrograms, 140
cgCombinePrograms2, 141
cgCombinePrograms3, 142
cgCombinePrograms4, 143
cgCombinePrograms5, 145
cgCompileProgram, 146
cgConnectParameter, 147
cgCopyEffect, 149
cgCopyProgram, 149
cgCreateArraySamplerState, 150
Cg 3.1 Reference Manual, Release 3.1

cgCreateArrayState, 151
cgCreateBuffer, 152
cgCreateContext, 154
cgCreateEffect, 154
cgCreateEffectAnnotation, 156
cgCreateEffectFromFile, 157
cgCreateEffectParameter, 158
cgCreateEffectParameterArray, 159
cgCreateEffectParameterMultiDimArray, 160
cgCreateObj, 161
cgCreateObjFromFile, 163
cgCreateParameter, 164
cgCreateParameterAnnotation, 165
cgCreateParameterArray, 166
cgCreateParameterMultiDimArray, 167
cgCreatePass, 168
cgCreatePassAnnotation, 169
cgCreateProgram, 170
cgCreateProgramAnnotation, 171
cgCreateProgramFromEffect, 172
cgCreateProgramFromFile, 173
cgCreateSamplerState, 175
cgCreateSamplerStateAssignment, 176
cgCreateState, 177
cgCreateStateAssignment, 178
cgCreateStateAssignmentIndex, 179
cgCreateTechnique, 180
cgCreateTechniqueAnnotation, 181
cgD3D10BindProgram, 629
cgD3D10CreateBuffer, 630
cgD3D10CreateBufferFromFile, 631
cgD3D10GetBufferByIndex, 632
cgD3D10GetBufferObject, 633
cgD3D10GetCompiledProgram, 633
cgD3D10GetDevice, 634
cgD3D10GetIASignatureByPass, 635
cgD3D10GetLastError, 636
cgD3D10GetLatestGeometryProfile, 637
cgD3D10GetLatestPixelProfile, 638
cgD3D10GetLatestVertexProfile, 638
cgD3D10GetManageTextureParameters, 639
cgD3D10GetOptimalOptions, 640
cgD3D10GetProgramErrors, 641
cgD3D10GetSamplerStateParameter, 642
cgD3D10GetTextureParameter, 643
cgD3D10LoadProgram, 643
cgD3D10UnbindProgram, 653
cgD3D10UnbindProgram, 653
cgD3D11BindProgram, 657
cgD3D11CreateBuffer, 658
cgD3D11CreateBufferFromFile, 659
cgD3D11GetBufferByIndex, 660
cgD3D11GetBufferObject, 660
cgD3D11GetCompiledProgram, 661
cgD3D11GetDevice, 662
cgD3D11GetIASignatureByPass, 663
cgD3D11GetLastError, 664
cgD3D11GetLatestDomainProfile, 665
cgD3D11GetLatestGeometryProfile, 665
cgD3D11GetLatestHullProfile, 666
cgD3D11GetLatestPixelProfile, 667
cgD3D11GetLatestVertexProfile, 668
cgD3D11GetManageTextureParameters, 669
cgD3D11GetOptimalOptions, 670
cgD3D11GetProgramErrors, 671
cgD3D11IsProfileSupported, 671
cgD3D11IsProfileSupported, 672
cgD3D11LoadProgram, 673
cgD3D11RegisterStates, 674
cgD3D11SetDevice, 675
cgD3D11SetManageTextureParameters, 676
cgD3D11SetSamplerStateParameter, 677
cgD3D11SetTextureParameter, 678
cgD3D11SetTextureSamplerStateParameter, 679
cgD3D11TranslateCGerror, 680
cgD3D11TranslateCGerror, 681
cgD3D11TranslateHRESULT, 681
cgD3D11TypeToSize, 682
cgD3D11UnbindProgram, 683
cgD3D11UnbindProgram, 683
cgD3D9BindProgram, 693
cgD3D9EnableDebugTracing, 694
cgD3D9EnableParameterShadowing, 694
cgD3D9SetDevice, 695
cgD3D9SetDevice, 696
cgD3D9SetManageTextureParameters, 697
cgD3D9SetManageTextureParameters, 698
cgD3D9GetDevice, 699
cgD3D9GetVertexDeclaration, 700
cgD3D9IsParameterShadowingEnabled, 701
cgD3D9IsParameterShadowingEnabled, 702
cgD3D9IsProfileSupported, 703
cgD3D9IsProfileSupported, 704
cgD3D9ResourceToDeclUsage, 705
cgD3D9SetManageTextureParameters, 706
cgD3D9SetManageTextureParameters, 707
cgD3D9SetDevice, 708
cgD3D9SetManageTextureParameters, 709
cgD3D9SetManageTextureParameters, 710
Index
cgGetNamedState, 264

cgGetNamedStateAssignment, 265

cgGetNamedStructParameter, 266

cgGetNamedSubParameter, 266

cgGetNamedTechnique, 267

cgGetNamedTechniqueAnnotation, 268

cgGetNamedUniformBufferParameter, 269

cgGetNamedUserType, 270

cgGetNextAnnotation, 271

cgGetNextEffect, 272

cgGetNextLeafParameter, 273

cgGetNextParameter, 274

cgGetNextPass, 275

cgGetNextProgram, 276

cgGetNextState, 277

cgGetNextStateAssignment, 278

cgGetNextTechnique, 279

cgGetNumConnectedToParameters, 280

cgGetNumDependentAnnotationParameters, 281

cgGetNumDependentProgramArrayStateAssignmentParameters, 282

cgGetNumDependentStateAssignmentParameters, 283

cgGetNumParentTypes, 284

cgGetNumProgramDomains, 285

cgGetNumStateEnumerants, 286

cgGetNumSupportedProfiles, 287

cgGetParameterBaseResource, 288

cgGetParameterBaseType, 289

cgGetParameterBufferIndex, 290

cgGetParameterBufferOffset, 291

cgGetParameterClass, 292

cgGetParameterClassEnum, 293

cgGetParameterClassString, 294

cgGetParameterColumns, 295

cgGetParameterContext, 296

cgGetParameterDefaultValue, 297

cgGetParameterDefaultValuedc, 298

cgGetParameterDefaultValueedr, 299

cgGetParameterDefaultValuefc, 300

cgGetParameterDefaultValuefr, 301

cgGetParameterDefaultValueic, 302

cgGetParameterDefaultValueir, 303

cgGetParameterDefaultValue, 304

cgGetParameterDefaultValue, 305

cgGetParameterDefaultValue, 306

cgGetParameterDefaultValue, 307

cgGetParameterDefaultValue, 308

cgGetParameterDefaultValue, 309

cgGetParameterDefaultValue, 310

cgGetParameterDefaultValue, 311

cgGetParameterDefaultValue, 312

cgGetParameterDefaultValue, 313

cgGetParameterDefaultValue, 314

cgGetParameterResourceType, 315

cgGetParameterRows, 316

cgGetParameterSemantic, 317

cgGetParameterSettingMode, 318

cgGetParameterType, 319

cgGetParameterValue, 320

cgGetParameterValueude, 321

cgGetParameterValuef, 322

cgGetParameterValue, 323

cgGetParameterValuef, 324

cgGetParameterValueic, 325

cgGetParameterValueic, 326

cgGetParameterValueic, 327

cgGetParameterValueic, 328

cgGetParameterValueic, 329

cgGetParameterValueic, 330

cgGetParameterValueic, 331

cgGetParameterValueic, 332

cgGetParameterValueic, 333

cgGetParameterValueic, 334

cgGetParameterValueic, 335

cgGetParameterValueic, 336

cgGetParameterValueic, 337

cgGetParameterValueic, 338

cgGetParameterValueic, 339

cgGetParameterValueic, 340

cgGetParameterValueic, 341

cgGetParameterValueic, 342

cgGetParameterValueic, 343

cgGetParameterValueic, 344

cgGetParameterValueic, 345

cgGetParameterValueic, 346

cgGetParameterValueic, 347

cgGetParameterValueic, 348

cgGetParameterValueic, 349

cgGetParameterValueic, 350

cgGetParameterValueic, 351

cgGetParameterValueic, 352

cgGetParameterValueic, 353

cgGetParameterValueic, 354

cgGetParameterValueic, 355

cgGetParameterValueic, 356

cgGetParameterValueic, 357

gGetParameterValueic, 358

cgGetParameterValueic, 359

cgGetStateContext, 360

cgGetStateEnumerant, 361

cgGetStateEnumerantValue, 362

cgGetStateLatestProfile, 363

cgGetStateName, 364

cgGetStateResetCallback, 365

cgGetStateSetCallback, 366

cgGetProfileDomain, 334

cgGetProfileProperty, 335

cgGetProfileSibling, 337

cgGetProfileString, 338

cgGetProgramBuffer, 339

cgGetProgramBufferMaxIndex, 340

cgGetProgramBufferMaxSize, 340

cgGetProgramContext, 341

cgGetProgramDomain, 342

cgGetProgramDomainProfile, 343

cgGetProgramDomainProgram, 344

cgGetProgramInput, 345

cgGetProgramOptions, 346

cgGetProgramOutput, 347

cgGetProgramOutputVertices, 348

cgGetProgramProfile, 349

cgGetProgramStateAssignmentValue, 350

cgGetProgramString, 351

cGetResource, 352

cGetResourceString, 353

cGetSamplerStateAssignmentParameter, 354

cGetSamplerStateAssignmentState, 354

cGetSamplerStateAssignmentValue, 355

cGetSemanticCasePolicy, 356

cGetStateAssignmentIndex, 357

cGetStateAssignmentPass, 358

cGetStateAssignmentState, 358

cGetStateAssignm
cgGLSetParameterArray1f, 574
cgGLSetParameterArray2d, 575
cgGLSetParameterArray2f, 576
cgGLSetParameterArray3d, 577
cgGLSetParameterArray3f, 578
cgGLSetParameterArray4d, 579
cgGLSetParameterArray4f, 580
cgGLSetParameterPointer, 584
cgGLSetStateMatrixParameter, 585
cgGLSetTextureParameter, 587
cgGLSetupSampler, 588
cgGLUnbindProgram, 589
cgGLUnloadProgram, 590
cginfo, 40
cgIsAnnotation, 382
cgIsBuffer, 383
cgIsContext, 383
cgIsEffect, 384
cgIsInterfaceType, 385
cgIsParameter, 386
cgIsParameterGlobal, 387
cgIsParameterReferenced, 387
cgIsParameterUsed, 388
cgIsParentType, 390
cgIsPass, 390
cgIsProfileSupported, 391
cgIsProgram, 392
cgIsProgramCompiled, 393
cgIsState, 394
cgIsStateAssignment, 395
cgIsTechnique, 395
cgIsTechniqueValidated, 396
cgMapBuffer, 397
cgResetPassState, 398
cgSetArraySize, 399
cgSetAutoCompile, 400
cgSetBoolAnnotation, 402
cgSetBoolAssignment, 403
cgSetBoolStateAssignment, 404
cgSetBufferData, 405
cgSetBufferSubData, 406
cgSetCompilerIncludeCallback, 407
cgSetCompilerIncludeFile, 407
cgSetCompilerIncludeString, 408
cgSetContextBehavior, 409
cgSetEffectName, 411
cgSetEffectParameterBuffer, 412
cgSetErrorCallback, 414
cgSetErrorHandler, 415
cgSetFloatAnnotation, 416
cgSetFloatArrayStateAssignment, 417
cgSetFloatStateAssignment, 418
cgSetIntAnnotation, 419
cgSetIntArrayStateAssignment, 419
cgSetIntStateAssignment, 420
cgSetLastListing, 421
cgSetLockingPolicy, 422
cgSetMatrixParameter, 423
cgSetMatrixParameterdc, 424
cgSetMatrixParameterdr, 425
cgSetMatrixParameterfc, 426
cgSetMatrixParameterfr, 427
cgSetMatrixParameteric, 428
cgSetMatrixParameterir, 429
cgSetMultiDimArraySize, 430
cgSetParameter, 431
cgSetParameter1d, 433
cgSetParameter1dv, 434
cgSetParameter1f, 435
cgSetParameter1fv, 436
cgSetParameter1i, 436
cgSetParameter1iv, 437
cgSetParameter2d, 438
cgSetParameter2dv, 439
cgSetParameter2f, 440
cgSetParameter2fv, 441
cgSetParameter2i, 442
cgSetParameter2iv, 443
cgSetParameter3d, 444
cgSetParameter3dv, 445
cgSetParameter3f, 446
cgSetParameter3fv, 447
cgSetParameter3i, 448
cgSetParameter3iv, 449
cgSetParameter4d, 450
cgSetParameter4dv, 451
cgSetParameter4f, 452
cgSetParameter4fv, 453
cgSetParameter4i, 454
cgSetParameter4iv, 455
cgSetParameterSemantic, 456
cgSetParameterSettingMode, 457
cgSetParameterValue, 459
cgSetParameterValue, 459
cgSetParameterValuedc, 460
cgSetParameterValueDr, 461
cgSetParameterValuefc, 462
cgSetParameterValuefr, 464
cgSetParameterValueic, 465
cgSetParameterValueir, 466
cgSetParameterVariability, 467
cgSetPassProgramParameters, 469
cgSetPassState, 469
cgSetProgramBuffer, 470
cgSetProgramOutputVertices, 471
cgSetProgramProfile, 472
cgSetProgramStateAssignment, 473
cgSetSamplerState, 474
cgSetSamplerStateAssignment, 475
Cg 3.1 Reference Manual, Release 3.1

cgSetSemanticCasePolicy, 476

cgSetStateCallbacks, 477

cgSetStateLatestProfile, 478

cgSetStringAnnotation, 479

cgSetStringParameterValue, 480

cgSetStringStateAssignment, 481

cgSetTextureStateAssignment, 481

cgSetUniformBufferParameter, 483

cgUnmapBuffer, 483

cgUpdatePassParameters, 484

cgUpdateProgramParameters, 485

cgValidateTechnique, 486

clamp, 697

ClearColor, 830

ClearDepth, 831

ClearStencil, 831

clip, 698

Clipping, 832

ClipPlane, 833

ClipPlaneEnable, 832

ColorArg0, 833

ColorArg1, 834

ColorArg2, 835

ColorLogicOpEnable, 835

ColorMask, 836

ColorMaterial, 836

ColorMatrix, 837

ColorOp, 837

ColorTransform, 838

ColorVertex, 839

ColorWriteEnable, 841

ColorWriteEnable1, 839

ColorWriteEnable2, 840

ColorWriteEnable3, 840

CompareFunc, 982

CompareMode, 982

cos, 700

cosh, 699

cross, 701

CullFace, 842

CullFaceEnable, 842

CullMode, 843

D

ddx, 702

ddy, 703

degrees, 704

DepthBias, 843

DepthBounds, 844

DepthBoundsEnable, 844

DepthClampEnable, 845

DepthFunc, 846

DepthMask, 846

DepthMode, 983

DepthRange, 847

DepthTestEnable, 847

DestBlend, 849

DestBlendAlpha, 848

determinant, 705

DiffuseMaterialSource, 849

distance, 707

DitherEnable, 850

dot, 708

ds_5_0, 44

E

EmissiveMaterialSource, 850

exp, 710

exp2, 709

F

faceforward, 711

FillMode, 851

findLSB, 712

findMSB, 713

floatToIntBits, 714

floatToRawIntBits, 715

floor, 716

fmod, 717

FogColor, 852

FogCoordSrc, 852

FogDensity, 853

FogDistanceMode, 853

FogEnable, 854

FogEnd, 855

FogMode, 855

FogStart, 856

FogTableMode, 856

FogVertexMode, 857

fp20, 45

fp30, 52

fp40, 54

frac, 718

FragmentEnvParameter, 858

FragmentLocalParameter, 858

FragmentProgram, 859

frexp, 719

FrontFace, 859

fwidth, 720

G

GenerateMipmap, 983

GeometryProgram, 860

GeometryShader, 861

glsl, 59

glslf, 56

glslg, 58

glslv, 60
glut, 996
gp4, 77
gp4fp, 62
gp4gp, 69
gp4vp, 78
gp5, 85
gp5fp, 84
gp5gp, 84
gp5tcp, 86
gp5tep, 87
gp5vp, 89
gs_4_0, 90
gs_5_0, 92

H
hlsl10, 93
hlsl11, 94
hlslf, 94
hlslv, 96
hs_5_0, 98

I
IndexedVertexBlendEnable, 861
intBitsToFloat, 721
inverse, 722
isfinite, 723
isinf, 724
isnan, 725

L
LastPixel, 862
ldexp, 726
length, 727
lerp, 728
LightAmbient, 862
LightAttenuation0, 863
LightAttenuation1, 864
LightAttenuation2, 864
LightConstantAttenuation, 865
LightDiffuse, 866
LightDirection, 866
LightEnable, 867
LightFalloff, 867
Lighting, 869
LightingEnable, 868
LightLinearAttenuation, 869
LightModelAmbient, 870
LightModelColorControl, 871
LightModelLocalViewerEnable, 871
LightModelTwoSideEnable, 872
LightPhi, 872
LightPosition, 873
LightQuadraticAttenuation, 874
LightRange, 874
LightSpecular, 875
LightSpotCutoff, 876
LightSpotDirection, 876
LightSpotExponent, 877
LightTheta, 877
LightType, 878
LineSmoothEnable, 879
LineStipple, 880
LineStippleEnable, 879
LineWidth, 880
lit, 729
LocalViewer, 881
LODBias, 984
log, 733
log10, 731
log2, 732
LogicOp, 882
LogicOpEnable, 881

M
Mac OS X, 997
MagFilter, 984
MagFilteri, 883
MaterialAmbient, 883
MaterialDiffuse, 884
MaterialEmission, 884
MaterialEmissive, 885
MaterialPower, 886
MaterialShininess, 886
MaterialSpecular, 887
max, 734
MaxAnisotropy, 985
MaxAnisotropyi, 887
MaxMipLevel, 985
MaxMipLeveli, 888
min, 735
MinFilter, 986
MinFilteri, 889
MipFilter, 987
MipFilteri, 889
MipMapLodBias, 988
MipMapLodBiasi, 990
ModelViewMatrix, 890
ModelViewTransform, 891
modf, 736
mul, 737
MultiSampleAntialias, 892
MultisampleEnable, 892
MultiSampleMask, 893

N
normalize, 741
NormalizeEnable, 893
NormalizeNormals, 894
# Index

## P
- pack, 742
- PatchSegments, 895
- PixelShader, 900
- PixelShaderConstant, 899
- PixelShaderConstant1, 895
- PixelShaderConstant2, 896
- PixelShaderConstant3, 896
- PixelShaderConstant4, 897
- PixelShaderConstantB, 898
- PixelShaderConstantF, 898
- PixelShaderConstantI, 899
- PointDistanceAttenuation, 901
- PointFadeThresholdSize, 901
- PointScale_A, 902
- PointScale_B, 902
- PointScale_C, 903
- PointScaleEnable, 903
- PointSize, 905
- PointSizeMax, 904
- PointSizeMin, 905
- PointSmoothEnable, 906
- PointSpriteCoordOrigin, 906
- PointSpriteCoordReplace, 907
- PointSpriteEnable, 907
- PointSpriteRMode, 908
- PolygonMode, 909
- PolygonOffset, 910
- PolygonOffsetFillEnable, 909
- PolygonOffsetLineEnable, 910
- PolygonOffsetPointSizeEnable, 911
- PolygonSmoothEnable, 911
- PolygonStippleEnable, 912
- pow, 743
- ProjectionMatrix, 913
- ProjectionTransform, 913
- ps_1_1, 100
- ps_1_2, 101
- ps_1_3, 102
- ps_2_0, 102
- ps_2_sw, 105
- ps_2_x, 107
- ps_3_0, 109
- ps_4_0, 111
- ps_5_0, 113
- rsqrt, 749

## S
- SampleAlphaToCoverageEnable, 916
- SampleAlphaToOneEnable, 916
- SampleCoverageEnable, 917
- Sampler, 917
- saturate, 750
- Scissor, 918
- ScissorTestEnable, 918
- SeparateAlphaBlendEnable, 919
- ShadeMode, 920
- ShadeModel, 920
- sign, 751
- sin, 754
- sincos, 752
- sinh, 753
- SlopeScaleDepthBias, 921
- smoothstep, 755
- SpecularEnable, 921
- SpecularMaterialSource, 922
- sqrt, 756
- SrcBlend, 923
- SrcBlendAlpha, 922
- SRGBTexture, 989
- StencilEnable, 924
- StencilFail, 924
- StencilFunc, 925
- StencilFuncSeparate, 925
- StencilMask, 926
- StencilMaskSeparate, 927
- StencilOp, 927
- StencilOpSeparate, 928
- StencilPass, 929
- StencilRef, 929
- StencilTestEnable, 930
- StencilTestTwoSideEnable, 930
- StencilWriteMask, 931
- StencilZFail, 932
- step, 757

## T
- tan, 759
- tanh, 758
- TessellationControlProgram, 932
- TessellationControlShader, 933
- TessellationEvaluationProgram, 933
- TessellationEvaluationShader, 934
- tex1D, 760
- tex1DARRAY, 771
- tex1DARRAYbias, 767
- tex1DARRAYcmpbias, 768
- tex1DARRAYclmplod, 768
- tex1DARRAYfetch, 769

## R
- radians, 745
- RangeFogEnable, 914
- reflect, 745
- refract, 746
- RescaleNormalEnable, 914
- ResultArg, 915
- round, 747

Index | 1025
tex1DARRAYlod, 770
tex1DARRAYproj, 772
tex1DARRAYsize, 773
tex1Dbias, 762
tex1Dcmppbias, 762
tex1Dcmplod, 763
tex1Dfetch, 764
tex1Dlod, 765
tex1Dproj, 765
tex1Dszie, 766
tex2D, 774
tex2DARRAY, 782
tex2DARRAYbias, 780
tex2DARRAYfetch, 781
tex2DARRAYlod, 781
tex2DARRAYproj, 784
tex2DARRAYsize, 784
tex2Dbias, 773
tex2Dcmppbias, 775
tex2Dcmplod, 776
tex2Dfetch, 777
tex2Dlod, 777
tex2DARRAYfetch, 786
tex2DARRAYsize, 787
tex2DMfetch, 785
tex2DMsize, 786
tex2Dproj, 778
tex2Dszie, 779
tex3D, 790
tex3Dbias, 788
tex3Dfetch, 789
tex3Dlod, 789
tex3Dproj, 791
tex3Dszie, 792
texBUF, 793
texBUFsize, 793
TexCoordIndex, 935
texCUBE, 798
texCUBEARRAY, 796
texCUBEARRAYbias, 794
texCUBEARRAYlod, 795
texCUBEARRAYsize, 796
texCUBbias, 797
texCUBElod, 798
texCUBEproj, 799
texCUBEsize, 800
TexGenQEnable, 935
TexGenQEyePlane, 936
TexGenQMode, 936
TexGenQObjectPlane, 937
TexGenREnable, 937
TexGenREyePlane, 938
TexGenRMode, 938
TexGenRObjectPlane, 939
TexGenSEnable, 940
TexGenSEyePlane, 940
TexGenSMode, 941
TexGenSObjectPlane, 941
TexGenTEnable, 942
TexGenTEyePlane, 942
TexGenTMode, 943
TexGenTObjectPlane, 944
texRBUF, 801
texRBUFsize, 802
texRECT, 805
texRECTbias, 802
texRECTfetch, 803
texRECTlod, 804
texRECTproj, 806
texRECTsize, 807
Texture, 989
Texture1D, 945
Texture1DEnable, 944
Texture2D, 946
Texture2DEnable, 945
Texture3D, 947
Texture3DEnable, 947
TextureCubeMap, 948
TextureCubeMapEnable, 948
TextureEnvColor, 949
TextureEnvMode, 950
TextureFactor, 950
TextureMatrix, 951
TextureRectangle, 952
TextureRectangleEnable, 951
TextureTransform, 953
TextureTransformFlags, 953
Trace, 998
transpose, 807
trunc, 808
TweenFactor, 954
TwoSidedStencilMode, 954
unpack, 809
V
VertexBlend, 955
VertexEnvParameter, 956
VertexLocalParameter, 956
VertexProgram, 957
VertexProgramPointSizeEnable, 957
VertexProgramTwoSideEnable, 958
VertexShader, 964
VertexShaderConstant, 963
VertexShaderConstant1, 959
VertexShaderConstant2, 959
VertexShaderConstant3, 960
vertexShaderConstant4, 960
VertexShaderConstantB, 961
VertexShaderConstantF, 962
VertexShaderConstantI, 962
ViewTransform, 964
vp20, 116
vp30, 117
vp40, 118
vs_1_1, 121
vs_2_0, 121
vs_2_sw, 124
vs_2_x, 126
vs_3_0, 129
vs_4_0, 131
vs_5_0, 133

W
win64, 1000
WorldTransform, 965
Wrap0, 965
Wrap1, 970
Wrap10, 966
Wrap11, 967
Wrap12, 967
Wrap13, 968
Wrap14, 968
Wrap15, 969
Wrap2, 970
Wrap3, 971
Wrap4, 972
Wrap5, 972
Wrap6, 973
Wrap7, 974
Wrap8, 974
Wrap9, 975
WrapR, 990
WrapS, 990
WrapT, 990

Z
ZEnable, 975
ZFunc, 976
ZWriteEnable, 977