NAME

Cg − An 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 Core Runtime API
Cg OpenGL Runtime API
Cg Direct3D9 Runtime API
Cg Standard Libary Routines
CgFX States

SEE ALSO

Cg_language, cgCreateContext, cgDestroyContext

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:

```
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 improvments. 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:

```
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:

```
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.

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);

void cgSetParameter1fv(CGparameter param, const float *v); void cgSetParameter2fv(CGparameter param, const float *v); void cgSetParameter3fv(CGparameter param, const float *v); void cgSetParameter4fv(CGparameter param, const float *v); void cgSetParameter1dv(CGparameter param, const double *v); void cgSetParameter2dv(CGparameter param, const double *v); void cgSetParameter3dv(CGparameter param, const double *v); void cgSetParameter4dv(CGparameter param, const double *v);

void cgSetMatrixParameterdr(CGparameter param, const double *matrix); void cgSetMatrixParameterfr(CGparameter param, const float *matrix); void cgSetMatrixParameterdc(CGparameter param, const double *matrix); void cgSetMatrixParameterfc(CGparameter param, const float *matrix);

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:

```
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.

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.

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. See the Sub-Shaders entry elsewhere in this document for more information.

Recompilation can be executed manually by the application using the runtime or automatically by the

runtime.

The entry point:

void cgCompileProgram(CGprogram program);

causes the given program to be recompiled, and the function:

CGbool cgIsProgramCompiled(CGprogram program);

reurns 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 controled with the entry point :

```
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 conencted 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:

CGcontext cgGetParameterContext(CGparameter param);

And the entrypoint:

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 :

```
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 :

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 bound to a program parameter of the parent type (in the above

example this would be FooInterface) using cgBindParameter().

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 :

CGtype cgGetParameterNamedType(CGparameter param);

The parent types of a given named type may be obtained with the following entry points :

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 wheer should these go?

Some entrypoints from before have been updated in backwards compatible ways

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)...

CGparameter cgGetNamedProgramParameter(CGprogram program, CGenum name_space, const char *name);

TOPIC

glut − using Cg with the OpenGL Utility Toolkit (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

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

Cg Language Specification

This is version 2.0 of the Cg Language specification. This language specification describes version 2.0 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 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. Marks this declaration as solely a declaration and not a definition. There must be a non-`extern` declaration elsewhere in the program. 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` Only usable on a function definition. Tells the compiler that it should always inline calls to the function if at all possible. 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 Only usable on global variables. Marks the variable as 'private' to the program, and not visible externally. Cannot be combined with `uniform` or `varying` Only usable on parameter and `varying` declarations. Marks the parameter or varying as an output from the function or program 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. 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. 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:

```
functionName(void)
```

With no parameters at all:

```
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 the Overload resolution entry elsewhere in this document 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:

```
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 that `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:

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

  For example, implementations must predefine 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 predefine type identifiers (in the global scope) to represent these types:

```
packed TYPE1 TYPE1x1[1];
packed TYPE2 TYPE1x2[1];
packed TYPE3 TYPE1x3[1];
packed TYPE4 TYPE1x4[1];
packed TYPE1 TYPE2x1[2];
packed TYPE2 TYPE2x2[2];
packed TYPE3 TYPE2x3[2];
packed TYPE4 TYPE2x4[2];
packed TYPE1 TYPE3x1[3];
packed TYPE2 TYPE3x2[3];
packed TYPE3 TYPE3x3[3];
packed TYPE4 TYPE3x4[3];
packed TYPE1 TYPE4x1[4];
packed TYPE2 TYPE4x2[4];
packed TYPE3 TYPE4x3[4];
packed TYPE4 TYPE4x4[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 a *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 the unsized array entry elsewhere in this document 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`
- **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.

Target type \ Source type	Scalar	Vector	Matrix	Struct	Array
Scalar	A	W	W	E(3)	-
Vector	A	A/W(1)	W(2)	E(3)	E(6)
Matrix	A	W(2)	A/W(1)	E(3)	E(7)
Struct	E	E(4)	E(4)	E(4/5)	E(4)
Array	-	E(6)	E(7)	E(3)	E(6)

```
      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 `cfloat` 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 `cfloat` and the other is *floating*, the `cfloat` 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. 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

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[2][0], A[2][1], A[2][2], A[2][3] },
               { A[3][0], A[3][1], A[3][2], A[3][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:
    a. 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.
    b. 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.
    c. 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.
    d. 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
    e. Fail.
6. Choose a function based on profile:
    a. If there is at least one function with a profile that exactly matches the compilation profile, discard all functions that don't exactly match.
    b. 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 chose 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>] [=
```

- 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>._m<row><col>[_m<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>[_<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. Matrix with `n` rows and `m` columns Vector with `n` elements Unary vector negate Unary matrix negate Componentwise * Componentwise / Componentwise % Componentwise + Componentwise − Componentwise * Componentwise / Componentwise % Componentwise + 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*
virtual
void
volatile
while
```

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 is documented in ‘‘spec_stdlib.txt’’.

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 `POSITION` (or `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 `#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:

    ```
    OUT_POSITION = mul(MVP, IN_POSITION)
    ```

    where:

    OUT_POSITION
    : is a variable (or structure element) of type `float4` with an output binding semantic of `POSITION` or `HPOS`.

    IN_POSITION
    : is a variable (or structure element) of type `float4` with an input binding semantic of `POSITION`.

    MVP
    : is a uniform variable (or structure element) of type `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 `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

Name	Meaning	Type	Default Value
POSITION	Homogeneous clip-space position; fed to rasterizer.	float4	Undefined
PSIZE	Point size	float	Undefined

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

Name	Meaning	Type	Default Value
COLOR	RGBA output color	float4	Undefined
COLOR0	Same as COLOR		
DEPTH	Fragment depth value (in range [0,1])	float	Interpolated depth from rasterizer (in range [0,1])

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:

```
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.

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, cgGetStateName

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

NAME

cgCallStateSetCallback − calls the state setting callback function for a state assignment

SYNOPSIS

```
#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

NAME

cgCallStateValidateCallback − calls the state validation callback function for a state assignment

SYNOPSIS

```
#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

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, NUL
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NUL

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, cgCreateProgram, cgGLBindProgram, cgD3D9BindProgram, cgD3D8BindProgram

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, NUL
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NUL

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

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, NUL
CGprogram p2 = cgCreateProgram(context, CG_SOURCE, fSrc, fProfile, fEntryName, NUL
CGprogram p3 = cgCreateProgram(context, CG_SOURCE, gSrc, gProfile, gEntryName, NUL

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

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

```
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

NAME

cgConnectParameter − connect two parameters

SYNOPSIS

```
#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** :

```
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 orignal 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

```
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

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**.

EXAMPLES

to-be-written

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

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

to-be-written

ERRORS

CG_INVALID_PROGRAM_HANDLE_ERROR is generated if **program** is not a valid program handle.

HISTORY

cgCopyProgram was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram, cgDestroyProgram

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

context	The context in which to define the sampler state.
name	The name of the new sampler state.
type	The type of the new sampler 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

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, cgGetStateName, cgGetStateType, cgIsState, cgSetStateCallbacks, cgGLRegisterStates

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

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

NAME

cgCreateBuffer − create a buffer object managed by the runtime

SYNOPSIS

```
#include <Cg/cg.h>

CGbuffer cgCreateBuffer( CGcontext context, int size, const void *data, CGbufferus
```

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 inital 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 any 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, initalData, 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

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

to-be-written

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

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 any error occurs. cgGetLastListing can be called to retrieve any warning or error messages from the compilation process.

DESCRIPTION

cgCreateEffect generates a new **CGeffect** object and adds it to the specified Cg context.

EXAMPLES

```
char *effectSource = ...;
CGcontext context = cgCreateContext();
CGeffect effect  = cgCreateEffect(context,
                                  effectSource,
                                  NULL);
```

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

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 any 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

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

context 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 any error occurs. cgGetLastListing can be called to retrieve any warning or error messages from the compilation process.

DESCRIPTION

cgCreateEffectFromFile generates a new **CGeffect** object and adds it to the specified Cg context.

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

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

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 specificed effect.

EXAMPLES

```
CGeffect effect = cgCreateEffect( ... );
CGparameter array = cgCreateEffectParameterArray( effect, "myFloatArray", 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

cgCreateEffectParameterArray was introduced in Cg 1.5.

SEE ALSO

cgCreateEffectParameter, 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, "myFloatMultiArra
```

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

NAME

cgCreateObj − *to-be-written*

SYNOPSIS

```
#include <Cg/cg.h>

CGobj cgCreateObj( CGcontext context,
                   CGenum program_type,
                   const char * source,
                   CGprofile profile,
                   const char ** args );
```

PARAMETERS

context *to-be-written*

program_type
to-be-written

source *to-be-written*

profile *to-be-written*

args *to-be-written*

RETURN VALUES

Returns *to-be-written*

DESCRIPTION

cgCreateObj does *to-be-written*

EXAMPLES

to-be-written

ERRORS

to-be-written

HISTORY

cgCreateObj was introduced in Cg 2.0.

SEE ALSO

cgCreateObjFromFile, cgDestroyObj

NAME

cgCreateObjFromFile − *to-be-written*

SYNOPSIS

```
#include <Cg/cg.h>

CGobj cgCreateObjFromFile( CGcontext context,
                           CGenum program_type,
                           const char * source_file,
                           CGprofile profile,
                           const char ** args );
```

PARAMETERS

context *to-be-written*

program_type
to-be-written

source_file
to-be-written

profile *to-be-written*

args *to-be-written*

RETURN VALUES

Returns *to-be-written*

DESCRIPTION

cgCreateObjFromFile does *to-be-written*

EXAMPLES

to-be-written

ERRORS

to-be-written

HISTORY

cgCreateObjFromFile was introduced in Cg 2.0.

SEE ALSO

cgCreateObj, cgDestroyObj

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

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 any error occurs.

DESCRIPTION

cgCreateParameterAnnotation adds a new annotation to the specified parameter.

EXAMPLES

```
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

NAME

cgCreateParameterArray − creates a parameter array

SYNOPSIS

```
#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

```
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

NAME

cgCreateParameterMultiDimArray − creates a multi-dimensional parameter array

SYNOPSIS

```
#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

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 any 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

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 any 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

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 any 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

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 any 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

NAME

cgCreateProgramFromEffect − create a program object from an effect

SYNOPSIS

```
#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 any 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

NAME

cgCreateProgramFromFile − create a program object from a file

SYNOPSIS

```
#include <Cg/cg.h>

CGprogram cgCreateProgramFromFile( CGcontext context,
                                   CGenum program_type,
                                   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 any 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

NAME

cgCreateSamplerState − create a sampler state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateSamplerState( CGcontext context,
                              const char * name,
                              CGtype type );
```

PARAMETERS

context 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, cgGetStateName, cgGetStateType, cgIsState, cgCreateSamplerStateAssignment, cgGLRegisterStates

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

NAME

cgCreateState − create a state definition

SYNOPSIS

```
#include <Cg/cg.h>

CGstate cgCreateState( CGcontext context,
                       const char * name,
                       CGtype type );
```

PARAMETERS

context The context in which to define the new state.

name 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

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

cgCreateState was introduced in Cg 1.4.

SEE ALSO

cgCreateArrayState, cgGetStateContext, cgGetStateName, cgGetStateType, cgIsState, cgSetStateCallbacks, cgGLRegisterStates

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

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:

```
/* 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

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 any 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

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 any error occurs.

DESCRIPTION

cgCreateTechniqueAnnotation adds a new annotation to the technique.

EXAMPLES

```
/* 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

NAME

cgDestroyBuffer − delete a buffer

SYNOPSIS

```
#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

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

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

NAME

cgDestroyObj − *to-be-written*

SYNOPSIS

```
#include <Cg/cg.h>

void cgDestroyObj( CGobj obj );
```

PARAMETERS

obj *to-be-written*

RETURN VALUES

None.

DESCRIPTION

cgDestroyObj does *to-be-written*

EXAMPLES

to-be-written

ERRORS

to-be-written

HISTORY

cgDestroyObj was introduced in Cg 2.0.

SEE ALSO

cgCreateObj, cgCreateObjFromFile

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

NAME

cgDestroyProgram − destroy a program

SYNOPSIS

```
#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

NAME

cgDisconnectParameter − disconnects two parameters

SYNOPSIS

```
#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

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 evalutated.
buf	Buffer in which to store the results of program evaluation.
ncomps	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

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

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

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

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

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) */
if (cgIsArray(param)) {
  int dim = cgGetArrayDimension(param);
  int elements = cgGetArraySize(param, 0);
  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, cgGetMatrixSize, cgGetTypeSizes

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 **pararm** 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

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 the type of the inner most 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 inner most 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

NAME

cgGetAutoCompile − get the auto-compile enumerant for a context

SYNOPSIS

```
#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

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

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

NAME

cgGetBooleanAnnotationValues − deprecated

DESCRIPTION

cgGetBooleanAnnotationValues is deprecated. Use cgGetBoolAnnotationValues instead.

SEE ALSO

cgGetBoolAnnotationValues

NAME

cgGetBufferSize − get the size of a buffer

SYNOPSIS

```
#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

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

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

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

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

```
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

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

NAME

cgGetDependentStateAssignmentParameter − get one of the parameters that a state assignment's value depends on

SYNOPSIS

```
#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.

Returns **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

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

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

NAME

cgGetEffectParameterBySemantic − get the a parameter in an effect via its semantic

SYNOPSIS

```
#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

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

NAME

cgGetEnumString − get the name string associated with an enumerant

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

NAME

cgGetError − get error condition

SYNOPSIS

```
#include <Cg/cg.h>

CGerror cgGetError( void );
```

PARAMETERS

None.

RETURN VALUES

Returns the last error condition that has occured.

Returns **CG_NO_ERROR** if no error has occurred.

DESCRIPTION

cgGetError returns the last error condition that has occured. 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

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

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

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

NAME

cgGetFirstDependentParameter − get the first dependent parameter from a parameter

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetFirstDependentParameter( CGparameter param );
```

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

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 cgGetNextEffect for more information.

EXAMPLES

```
/* one or more effects have previously been loaded into context */
CGeffect effect = cgGetFirstEffect( context );
```

ERRORS

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

NAME

cgGetFirstEffectAnnotation − get the first annotation in an effect

SYNOPSIS

```
#include <Cg/cg.h>

CGannotation cgGetFirstEffectAnnotation( CGeffect effect );
```

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 **cgGetFirstEffectAnnotation**. The rest of the effect's annotations can be discovered by iterating through them using cgGetNextAnnotation.

EXAMPLES

```
CGannotation ann = cgGetFirstEffectAnnotation( effect );
while( ann )
{
   /* do something with ann */
   ann = cgGetNextAnnotation( ann );
}
```

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

NAME

cgGetFirstEffectParameter − get the first parameter in an effect

SYNOPSIS

```
#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

```
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

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 occured since **cgGetFirstError** was last called.

Returns **CG_NO_ERROR** if no error has occurred.

DESCRIPTION

cgGetFirstError returns the first error condition that has occured since **cgGetFirstError** was previously called.

EXAMPLES

```
CGerror firstError = cgGetFirstError();
```

ERRORS

None.

HISTORY

cgGetFirstError was introduced in Cg 1.4.

SEE ALSO

cgSetErrorHandler, cgGetError

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

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

```
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

NAME

cgGetFirstParameter − get the first parameter in a program

SYNOPSIS

```
#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 **NULL** if **program** is invalid or if **program** does not have any parameters.

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

```
CGparameter param = cgGetFirstParameter( program, CG_GLOBAL );
while ( param )
{
  /* Do stuff with leaf */
  param = cgGetNextParameter( param );
}
```

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

cgGetFirstParameter was introduced in Cg 1.1.

SEE ALSO

cgGetNextParameter

NAME

cgGetFirstParameterAnnotation − get the first annotation of a parameter

SYNOPSIS

```
#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

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

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

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

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

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

NAME

cgGetFirstSamplerStateAssignment − get the first state assignment in a sampler_state block

SYNOPSIS

```
#include <Cg/cg.h>

CGstateassignment cgGetFirstSamplerStateAssignment( CGparameter param );
```

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

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 cgGetNextState 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

NAME

cgGetFirstStateAssignment − get the first state assignment in a pass

SYNOPSIS

```
#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

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

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

to-be-written

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

cgGetNextTechnique, cgGetNamedTechnique, cgIsTechnique

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

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

NAME

cgGetFloatStateAssignmentValues − get a float-valued state assignment's values

SYNOPSIS

```
#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.

HISTORY

cgGetFloatStateAssignmentValues was introduced in Cg 1.4.

SEE ALSO

cgGetStateAssignmentState, cgGetStateType, cgGetIntStateAssignmentValues, cgGetBoolStateAssignmentValues, cgGetStringStateAssignmentValue, cgGetProgramStateAssignmentValue, cgGetSamplerStateAssignmentValue, cgGetTextureStateAssignmentValue

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

NAME

cgGetIntStateAssignmentValues − get an int-valued state assignment's values

SYNOPSIS

```
#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

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 ocurred 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

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 currrent 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

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

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

cgGetParameterRows, cgGetParameterColumns, cgGetParameterValues

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

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

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

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

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

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

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

cgGetArrayTotalSize, cgGetArrayDimension, cgGetArrayParameter, cgGetTypeSizes

NAME

cgGetNamedEffect − get an effect from a context by name

SYNOPSIS

```
#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

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

NAME

cgGetNamedEffectParameter − get an effect parameter by name

SYNOPSIS

```
#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

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;
};

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

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

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

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

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

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

NAME

cgGetNamedSamplerState − get a sampler state by name

SYNOPSIS

```
#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

NAME

cgGetNamedSamplerStateAssignment − get a sampler state assignment by name

SYNOPSIS

```
#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

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

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

NAME

cgGetNamedStructParameter − get 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

NAME

cgGetNamedSubParameter − gets a ‘‘shallow’’ or ‘‘deep’’ parameter from an aggregate parameter (ie struct, array, etc.)

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNamedSubParameter( CGparameter param,
                                    const char * name );
```

PARAMETERS

param Aggregate parameter.

name Name of the parameter inside the aggregate parameter (param) being requested.

RETURN VALUES

Returns the named parameter.

Returns **NULL** if **param** has no parameter corresponding to **name**.

DESCRIPTION

cgGetNamedSubParameter is a generalized parameter getter function that will retrieve parameters, including deep parameters, of an aggregate parameter type such as a structure or an array.

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

None.

HISTORY

cgGetNamedSubParameter was introduced in Cg 1.5.

SEE ALSO

cgGetNamedParameter, cgGetNamedStructParameter, cgGetArrayParameter

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

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

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 constuct 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

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

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 **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

```
CGeffect effect = cgGetFirstEffect( context );
while( effect )
{
  /* do something with effect */
  effect = cgGetNextEffect( effect );
}
```

ERRORS

CG_INVALID_EFFECT_HANDLE_ERROR is generated if **effect** is not a valid effect.

HISTORY

cgGetNextEffect was introduced in Cg 1.4.

SEE ALSO

cgGetFirstEffect

NAME

cgGetNextLeafParameter − get the next leaf parameter in a program or effect

SYNOPSIS

```
#include <Cg/cg.h>

CGparameter cgGetNextLeafParameter( CGparameter param );
```

PARAMETERS

param The current leaf parameter.

RETURN VALUES

Returns the next leaf **CGparameter** object.

Returns **NULL** if **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 cgGetFirstLeafEffectParameter.

EXAMPLES

```
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

NAME

cgGetNextParameter − iterate through a program's or effect's parameters

SYNOPSIS

```
#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

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

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

```
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

NAME

cgGetNextState − iterate through states in a context

SYNOPSIS

```
#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

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

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 )
}
```

ERRORS

CG_INVALID_TECHNIQUE_HANDLE_ERROR is generated if **tech** is not a valid technique.

HISTORY

cgGetNextTechnique was introduced in Cg 1.4.

SEE ALSO

cgGetFirstTechnique, cgGetNamedTechnique

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 **NULL** 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

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

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 mangement 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

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:

```
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

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

```
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

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

cgGetUserType, cgGetNamedUserType

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

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

NAME

cgGetParameterBufferIndex − get buffer index by parameter

SYNOPSIS

```
#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.

EXAMPLES

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

NAME

cgGetParameterBufferOffset − get a parameter's buffer offset

SYNOPSIS

```
#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.

EXAMPLES

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

cgGetParameterBufferIndex, cgGetParameterResourceSize

NAME

cgGetParameterClass − get a parameter's class

SYNOPSIS

```
#include <Cg/cg.h>

CGparameterclass cgGetParameterClass( CGparameter param );
```

PARAMETERS

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

cgGetParameterType, cgGetType, cgGetTypeString

NAME

cgGetParameterColumns − get number of parameter columns

SYNOPSIS

```
#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

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

NAME

cgGetParameterDirection − get a program parameter's direction

SYNOPSIS

```
#include <Cg/cg.h>

CGenum cgGetParameterDirection( CGparameter param );
```

PARAMETERS

param 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

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

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

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

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

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

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

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

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 **param**.

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 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

cgGetParameterResourceIndex was introduced in Cg 1.1.

SEE ALSO

cgGetParameterResource, cgGetResource, cgGetResourceString

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 *to-be-written*

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

NAME

cgGetParameterResourceType − get a parameter's resource type

SYNOPSIS

```
#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

cgGetParameterResource, cgGetParameterResourceIndex, cgGetParameterResourceSize, cgGetResource, cgGetResourceString

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

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

cgGetParameterResource, cgGetParameterResourceIndex, cgGetParameterName, cgGetParameterType

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, cgUpdateProgramParameters

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_STRUCT** if the parameter is a struct and **CG_ARRAY** if the parameter is an array. 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

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 **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 **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, 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 **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

The **cgGetParameterValue** functions were introduced in Cg 1.4.

SEE ALSO

cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgSetParameterValue

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

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

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

cgGetParameterValuedr was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

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:

```
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

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

NAME

cgGetParameterValuefr − get the value of any numeric parameter

SYNOPSIS

```
#include <Cg/cg.h>

int cgGetParameterValuefr( 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

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 **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

cgGetParameterValuefr was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

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

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

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

cgGetParameterValueir was introduced in Cg 1.4.

SEE ALSO

cgGetParameterValue, cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize

NAME

cgGetParameterValues − get a program parameter's values

SYNOPSIS

```
#include <Cg/cg.h>

const double * cgGetParameterValues( CGparameter param,
                                     CGenum value_type,
                                     int * nvalues );
```

PARAMETERS

param The program parameter.

value_type

Determines what type of value to return. Valid enumerants are :

- **CG_CONSTANT**
  Returns the constant values for parameters that have constant variability. See cgGetParameterVariability for more information.
- **CG_DEFAULT**
  Returns the default values for a uniform parameter.
- **CG_CURRENT**
  Returns the current values for a uniform parameter.

nvalues Pointer to integer that will be initialized to store the number of values returned.

RETURN VALUES

Returns a pointer to an array of **double** values. The number of values in the array is returned via the **nvalues** parameter.

Returns **NULL** if no values are available, and **nvalues** will be **0**.

DESCRIPTION

cgGetParameterValues allows the application to retrieve default, current, or constant values from uniform parameters.

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 **nvalues** is **NULL**.

CG_INVALID_ENUMERANT_ERROR is generated if **value_type** is not **CG_CONSTANT**, **CG_DEFAULT**, or **CG_CURRENT**.

HISTORY

cgGetParameterValues was introduced in Cg 1.1.

SEE ALSO

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

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

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

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

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_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

NAME

cgGetProfileDomain − get the domain of a profile enumerant

SYNOPSIS

```
#include <Cg/cg.h>

CGdomain cgGetProfileDomain( CGprofile profile );
```

PARAMETERS

profile The profile enumerant.

RETURN VALUES

Returns:

CG_UNKNOWN_DOMAIN CG_FIRST_DOMAIN CG_VERTEX_DOMAIN CG_FRAGMENT_DOMAIN

DESCRIPTION

cgGetProfileDomain returns which type of domain the given profile belongs to.

EXAMPLES

```
CGdomain domain = cgGetProfileDomain(CG_PROFILE_PS_3_0);
/* domain == CG_FRAGMENT_DOMAIN */
```

ERRORS

None.

HISTORY

cgGetProfileDomain was introduced in Cg 1.5.

SEE ALSO

cgGetNumProgramDomains, cgGetProgramDomainProfile

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 named associated with a profile enumerant.

EXAMPLES

```
static void dumpCgProgramInfo(CGprogram program)
{
  const char* p = cgGetProfileString(cgGetProgramProfile(program));
  if ( p ) {
    printf(" Profile: %s\n", cgGetProfileString(cgGetProgramProfile(program)));
  }
  /* ... */
}
```

ERRORS

None.

HISTORY

cgGetProfileString was introduced in Cg 1.1.

SEE ALSO

cgGetProfile, cgGetProgramProfile

NAME

cgGetProgramBuffer − get buffer associated with a buffer index

SYNOPSIS

```
#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 any 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

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

NAME

cgGetProgramBufferMaxIndex − get the maximum index of a buffer for a given profile

SYNOPSIS

```
#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 any error occurs.

DESCRIPTION

cgGetProgramBufferMaxIndex returns the maximum buffer index for a **profile**. **cgGetProgramBufferMaxIndex** will return **0** if an invalid profile is passed.

EXAMPLES

int size = cgGetProgramBufferMaxIndex(CG_PROFILE_GPU_VP);

ERRORS

none.

HISTORY

cgGetProgramBufferMaxIndex was introduced in Cg 2.0.

SEE ALSO

cgSetProgramBuffer, cgGetParameterBufferIndex, cgCreateBuffer

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 any 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

cgSetProgramBuffer, cgGetParameterBufferIndex, cgCreateBuffer

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

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.

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++) {
  cgGLEnableProfile( cgGetProgramDomainProfile(glslComboProgram, i) );
}

/* This will enable the profile for the first program domain in glslComboProgram *
cgGLEnableProfile( cgGetProgramDomainProfile(glslComboProgram, 0) );
```

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

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**.

Returns **CG_UNKNOWN** if the input is unknown.

DESCRIPTION

cgGetProgramInput returns the program input enumerant.

EXAMPLES

```
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;
    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

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

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**.

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

```
void printProgramOutput(CGprogram program)
{
  char * output = NULL;
  switch(cgGetProgramOutput(program))
  {
    case CG_POINT_OUT:
        output = "point";
        break;
    case CG_LINE_OUT:
        output = "line";
        break;
    case CG_TRIANGLE_OUT:
        output = "triangle";
        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

NAME

cgGetProgramProfile − get a program's profile

SYNOPSIS

```
#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

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

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

```
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

NAME

cgGetResource − get the resource enumerant assigned to a resource name

SYNOPSIS

```
#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

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 named 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

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

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

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

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

cgGetSemanticCasePolicy returns an enumerant indicating the current semantic case policy for the library. See cgSetSemanticCasePolicy for more information.

EXAMPLES

```
CGenum currentSemanticCasePolicy = cgGetSemanticCasePolicy();
```

ERRORS

None.

HISTORY

cgGetSemanticCasePolicy was introduced in Cg 2.0.

SEE ALSO

cgSetSemanticCasePolicy, cgGetParameterSemantic

NAME

cgGetStateAssignmentIndex − 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

cgGetStateAssignmentIndex 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); }
```

cgGetStateAssignmentIndex 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

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

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

NAME

cgGetStateContext − get a state's context

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetStateContext( CGstate state );
```

PARAMETERS

state The state.

RETURN VALUES

Returns the context for the state.

Returns **NULL** if **state** is invalid.

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

NAME

cgGetStateEnumerantName − get a state enumerant name by value

SYNOPSIS

```
#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

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 any 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

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

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

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

NAME

cgGetStateType − returns the type of a given state

SYNOPSIS

```
#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

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 validateting 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

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 associtated 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

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

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 **ann** is not a valid annotation.

CG_INVALID_PARAMETER_ERROR is generated if **nvalues** is **NULL**.

HISTORY

cgGetStringAnnotationValues was introduced in Cg 2.0.

SEE ALSO

cgGetAnnotationType, cgGetStringAnnotationValue, cgGetBoolAnnotationValues, cgGetFloatAnnotationValues, cgGetIntAnnotationValues

NAME

cgGetStringParameterValue − get the value of a string parameter

SYNOPSIS

```
#include <Cg/cg.h>

const char * cgGetStringParameterValue( CGparameter param );
```

PARAMETERS

param The parameter whose value will be retrieved.

RETURN VALUES

Returns a pointer to the string contained by a string parameter.

Returns **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 **param** is not a valid parameter.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if **param** is not string-typed.

HISTORY

cgGetStringParameterValue was introduced in Cg 1.4.

SEE ALSO

cgSetStringParameterValue

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

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

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

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

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

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

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

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

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 named 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

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 **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.

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

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

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, assuming create succeeded */

cgDestroyContext(context);
cgIsContext(context);          /* returns CG_FALSE */
```

ERRORS

None.

HISTORY

cgIsContext was introduced in Cg 1.1.

SEE ALSO

cgCreateContext, cgDestroyContext

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

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

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

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

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 definintely 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

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 definintely 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 technqiue'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 the **param** handle 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

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**.

Returns **CG_FALSE** otherwise.

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

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

cgCreatePass, cgGetFirstPass, cgGetNamedPass, cgGetNextPass, cgGetPassName, cgGetPassTechnique

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

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

NAME

cgIsState − determine if a state handle references a valid state

SYNOPSIS

```
#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

NAME

cgIsStateAssignment − determine if a state assignment handle references a valid Cg state assignment

SYNOPSIS

```
#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

```
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

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

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

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|cgUnmapBuffer** when it's done updating or querying the buffer.

EXAMPLES

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_READ_ONLY**, **CG_WRITE_ONLY**, or **CG_READ_WRITE**.

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

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

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 resiable 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, or if **param** is not an array.

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

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

cgCompileProgram, cgIsProgramCompiled

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 **ann** is not a valid annotation.

CG_INVALID_PARAMETER_TYPE_ERROR is generated if **ann** is not an annotation of bool type.

CG_ARRAY_SIZE_MISMATCH_ERROR is generated if **ann** is not a scalar.

HISTORY

cgSetBoolAnnotation was introduced in Cg 1.5.

SEE ALSO

cgGetBoolAnnotationValues, cgSetIntAnnotation, cgSetFloatAnnotation, cgSetStringAnnotation

NAME

cgSetBoolArrayStateAssignment − set a bool-valued state assignment array

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgSetBoolArrayStateAssignment( CGstateassignment sa,
                                      const CGbool * vals );
```

PARAMETERS

sa A handle to a state assignment array of type **CG_BOOL**.

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

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 **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.

HISTORY

cgSetBoolArrayStateAssignment was introduced in Cg 1.5.

SEE ALSO

cgGetBoolStateAssignmentValues, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

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.

SEE ALSO

cgGetBoolStateAssignmentValues, cgSetBoolArrayStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

NAME

cgSetBufferData − resize and completely update a buffer object

SYNOPSIS

```
#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

NAME

cgSetBufferSubData − partially update a Cg buffer object

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetBufferSubData( CGbuffer buffer,
                         int offset,
                         int size,
                         const void * data );
```

PARAMETERS

buffer	The buffer which will be updated.
offset	The offset in bytes within the buffer where the partial update begins.
size	Specifies a new size for the buffer object. Zero means no bytes are updated.
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

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.

HISTORY

cgSetBufferSubData was introduced in Cg 2.0.

SEE ALSO

cgCreateBuffer, cgGLCreateBuffer, cgSetBufferData, cgMapBuffer, cgUnmapBuffer

NAME

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

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 occurrs. 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

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 occurrs. The callback function is passed:

context
The context in which the error occured. 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 **cgSetErrorHandler**. This pointer can be used to make arbitrary application-side information available to the error handler.

To disable the callback function, specify a **NULL** callback function pointer via **cgSetErrorHandler**.

EXAMPLES

```
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]);
  ...
}
```

ERRORS

to-be-written

HISTORY

cgSetErrorHandler was introduced in Cg 1.4.

SEE ALSO

cgGetErrorHandler, cgGetError, cgGetErrorString, cgGetFirstError

NAME

cgSetFloatAnnotation − set the value of a float annotation

SYNOPSIS

```
#include <Cg/cg.h>

CGbool cgSetFloatAnnotation( CGannotation ann,
                             float 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

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

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, cgSetBoolStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

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

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

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, cgSetIntStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetProgramStateAssignment, cgSetSamplerStateAssignment, cgSetStringStateAssignment, cgSetTextureStateAssignment

NAME

cgSetIntStateAssignment − set the value of an int state assignment

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

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 compiler, 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 **cgSetLastListing** 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

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 mulitiple 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

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 );
```

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

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, cgGetParameterValues

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

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, cgGetParameterValues

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, cgGetParameterValues

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, cgGetParameterValues

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

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 :

```
/* ... */

float4 main(float4 myarray[][][])
{
  /* ... */
}
```

You can set the sizes of each dimension of the **myarray** array parameter like so :

```
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, or if **param** is not an array.

CG_ARRAY_PARAM_ERROR is generated if **param** is not an array parameter.

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

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.

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

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

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

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

NAME

cgSetParameter1fv − sets the value of scalar and vector parameters

SYNOPSIS

```
#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

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

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

NAME

cgSetParameter2d − set the value of scalar and vector parameters

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetParameter2d( CGparameter param,
                       double x,
                       double y );
```

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

cgGetParameterValue, cgGetParameterValues

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** 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

cgSetParameter2dv was introduced in Cg 1.2.

SEE ALSO

cgGetParameterValue

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

cgGetParameterResource, cgGetParameterResourceIndex, cgGetParameterName, cgGetParameterType

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.

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, cgD3D9BindProgram, or cgD3D8BindProgram).

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 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.

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 **parameterSettingMode** is not **CG_IMMEDIATE_PARAMETER_SETTING** or **CG_DEFERRED_PARAMETER_SETTING**.

HISTORY

cgSetParameterSettingMode was introduced in Cg 2.0.

SEE ALSO

cgGetParameterSettingMode, cgUpdateProgramParameters, cgSetParameter, cgGLBindProgram, cgD3D9BindProgram, cgD3D8BindProgram

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;
```

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

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:

```
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

NAME

cgSetParameterValuedr − set the value of any numeric parameter

SYNOPSIS

```
#include <Cg/cg.h>

void cgSetParameterValuedr( 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

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

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

NAME

cgSetParameterValuefr − set the value of any numeric parameter

SYNOPSIS

```
#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:

```
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

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:

```
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

NAME

cgSetParameterValueir − set the value of any numeric parameter

SYNOPSIS

```
#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

cgSetParameterValueir 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

cgSetParameterValueir was introduced in Cg 1.4.

SEE ALSO

cgSetParameterValue, cgGetParameterRows, cgGetParameterColumns, cgGetArrayTotalSize, cgGetParameterValue

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. However parameters of **CG_UNIFORM** and **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 a not a numeric parameter.

HISTORY

cgSetParameterVariability was introduced in Cg 1.2.

SEE ALSO

cgGetParameterVariability

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

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

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, cgD3D8BindProgram

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

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

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

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_SAMP
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

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 coversion will be done to the semantic strings.

EXAMPLES

```
cgSetSemanticCasePolicy(CG_UNCHANGED_CASE_POLICY);  /* return original semantic st
```

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

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

NAME

cgSetStringAnnotation − set the value of a string annotation

SYNOPSIS

```
#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

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 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

NAME

cgSetStringParameterValue − set the value of a string parameter

SYNOPSIS

```
#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

NAME

cgSetStringStateAssignment − set the value of a string state assignment

SYNOPSIS

```
#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

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_SAMP
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.

SEE ALSO

cgGetTextureStateAssignmentValue, cgSetSamplerStateAssignment, cgSetBoolArrayStateAssignment, cgSetBoolStateAssignment, cgSetFloatArrayStateAssignment, cgSetFloatStateAssignment, cgSetIntArrayStateAssignment, cgSetIntStateAssignment, cgSetProgramStateAssignment, cgSetStringStateAssignment

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

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, cgSetParameter, cgGLBindProgram, cgD3D9BindProgram, cgD3D8BindProgram

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

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

NAME

cgGLCreateBuffer − create an OpenGL buffer object

SYNOPSIS

```
#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 inital 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

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.

HISTORY

cgGLCreateBuffer was introduced in Cg 2.0.

SEE ALSO

cgCreateBuffer, cgGLGetBufferObject

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

to-be-written

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **param** is not a varying parameter.

HISTORY

cgGLDisableClientState was introduced in Cg 1.1.

SEE ALSO

cgGLEnableClientState

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 approriate 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

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

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

NAME

cgGLEnableClientState − enables a vertex attribute in the OpenGL state

SYNOPSIS

```
#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

to-be-written

ERRORS

CG_INVALID_PARAMETER_ERROR is generated if **param** is not a varying parameter.

HISTORY

cgGLEnableClientState was introduced in Cg 1.1.

SEE ALSO

cgGLDisableClientState

NAME

cgGLEnableProfile − enable a profile within OpenGL

SYNOPSIS

```
#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 approriate 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

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

NAME

cgGLEnableTextureParameter − enables the texture unit associated with a texture parameter

SYNOPSIS

```
#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

NAME

cgGLGetBufferObject − get OpenGL buffer object for a buffer

SYNOPSIS

```
#include <Cg/cgGL.h>

GLuint CGGLENTRY 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

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**

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.

cgGLGetLatestProfile can be used in conjuction 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.

EXAMPLES

```
/* Output information about available profiles */
printf("vertex profile:   %s\n", cgGetProfileString(cgGLGetLatestProfile(CG_GL_VER
printf("geometry profile: %s\n", cgGetProfileString(cgGLGetLatestProfile(CG_GL_GEO
printf("fragment profile: %s\n", cgGetProfileString(cgGLGetLatestProfile(CG_GL_FRA
```

ERRORS

CG_INVALID_ENUMERANT_ERROR is generated if **profileClass** is not **CG_GL_VERTEX**, **CG_GL_GEOMETRY** or **CG_GL_FRAGMENT**.

HISTORY

cgGLGetLatestProfile was introduced in Cg 1.1.

CG_GL_GEOMETRY support was introduced in Cg 2.0.

SEE ALSO

cgGLSetOptimalOptions, cgCreateProgram

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

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

NAME

cgGLGetMatrixParameterArray − get the values from an matrix array parameter

SYNOPSIS

```
#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

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

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

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

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

cgGLGetMatrixParameterArrayfc was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

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 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

cgGLGetMatrixParameterArrayfr was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameter, cgGLSetParameterArray

NAME

cgGLGetMatrixParameterdc − get the values from a matrix parameter

SYNOPSIS

```
#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 **double**s 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

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 **double**s 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

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 **float**s 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

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 **float**s 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

cgGLGetMatrixParameterfr retrieves the values from a matrix parameter using row-major ordering.

cgGLGetMatrixParameterfr 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

cgGLGetMatrixParameterfr was introduced in Cg 1.1.

SEE ALSO

cgGLGetMatrixParameterArray, cgGLSetMatrixParameterArray, cgGLSetParameter

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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

to-be-written

ERRORS

CG_INVALID_CONTEXT_HANDLE_ERROR is generated if **context** is not a valid context.

HISTORY

cgGLRegisterStates was introduced in Cg 1.4.

SEE ALSO

cgCreateState, cgSetPassState, cgResetPassState, cgCallStateValidateCallback

NAME

cgGLSetDebugMode − control whether the cgGL runtime calls **glGetError**

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetDebugMode( CGbool debug );
```

PARAMETERS

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

cgSetErrorHandler, cgGetError

NAME

cgGLSetManageTextureParameters − set the manage texture parameters flag for a context

SYNOPSIS

```
#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

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

NAME

cgGLSetMatrixParameterArray − set the value of an array matrix parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

/* TYPE is float or double */

void cgGLSetMatrixParameterArray{fd}{rc}( CGparameter param,
                                          long offset,
                                          long nelements,
                                          const TYPE * 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

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

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 **double**s 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

NAME

cgGLSetMatrixParameterArraydr − set the values of a matrix array parameter

SYNOPSIS

```
#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 **double**s 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

NAME

cgGLSetMatrixParameterArrayfc − set the values of a matrix array parameter

SYNOPSIS

```
#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 **float**s 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

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**.

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

cgGLSetMatrixParameterArrayfr sets the value of a matrix array parameter from an array of **float**s 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

cgGLSetMatrixParameterArrayfr was introduced in Cg 1.1.

SEE ALSO

cgGLSetMatrixParameter, cgGLGetMatrixParameterArray

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 **double**s 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

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 **double**s 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

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 **float**s 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

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 **float**s 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

NAME

cgGLSetOptimalOptions − set the implicit compiler optimization options for a profile

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.

EXAMPLES

to-be-written

ERRORS

CG_INVALID_PROFILE_ERROR is generated if **profile** is not a supported OpenGL profile.

HISTORY

cgGLSetOptimalOptions was introduced in Cg 1.1.

SEE ALSO

cgCreateProgram

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,
                            TYPE x );

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.

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 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

NAME

cgGLSetParameter1d − set the values of a scalar or vector parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetParameter1d( 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

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

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 the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter1d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

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

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 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

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

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 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

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

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 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

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

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 the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter2d was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

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

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 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

NAME

cgGLSetParameter2f − set the values of a scalar or vector parameter

SYNOPSIS

```
#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

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 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

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

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 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

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

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 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

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

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 the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter3dv was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

NAME

cgGLSetParameter3f − set the values of a scalar or vector parameter

SYNOPSIS

```
#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

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 the parameter fails to set for any other reason.

HISTORY

cgGLSetParameter3f was introduced in Cg 1.1.

SEE ALSO

cgGLGetParameter, cgGLSetParameterArray, cgGLSetMatrixParameter, cgGLSetMatrixParameterArray, cgGLSetTextureParameter, cgGLBindProgram

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

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 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

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 );
```

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

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 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

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 parameter, the appropriate immediate mode OpenGL entry point will be called. However, the cgGLGetParameter functions only work 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_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, cgGLSetTextureParameter, cgGLBindProgram

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

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 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

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

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 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

NAME

cgGLSetParameterArray − set the values of an array parameter

SYNOPSIS

```
#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 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

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

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

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 **param** is not a valid parameter.

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

NAME

cgGLSetParameterArray2f − set the values of an array parameter

SYNOPSIS

```
#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 **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

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

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

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

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

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**.

v The array of values used to set the parameter. This must be a contiguous set of **4 * 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

NAME

cgGLSetParameterPointer − sets a varying parameter with an attribute array

SYNOPSIS

```
#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

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

NAME

cgGLSetTextureParameter − sets the value of a texture parameter

SYNOPSIS

```
#include <Cg/cgGL.h>

void cgGLSetTextureParameter( CGparameter param,
                              GLuint texobj );
```

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

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

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

to-be-written

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

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 intialized 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

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

NAME

cgD3D9EnableParameterShadowing − enable or disable parameter shadowing for a program

SYNOPSIS

```
#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

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

NAME

cgD3D9GetLastError − get the last D3D error that occurred

SYNOPSIS

```
#include <Cg/cgD3D9.h>

HRESULT cgD3D9GetLastError( 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 **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

```
HRESULT lastError = cgD3D9GetLastError();
```

ERRORS

None.

HISTORY

cgD3D9GetLastError was introduced in Cg 1.1.

SEE ALSO

cgD3D9TranslateHRESULT

NAME

cgD3D9GetLatestPixelProfile − get the latest supported pixel shader version

SYNOPSIS

```
#include <Cg/cgD3D9.h>

CGprofile cgD3D9GetLatestPixelProfile( 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

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.

EXAMPLES

```
CGprofile bestPixelProfile = cgD3D9GetLatestPixelProfile();
```

ERRORS

None.

HISTORY

cgD3D9GetLatestPixelProfile was introduced in Cg 1.1.

SEE ALSO

cgD3D9GetLatestVertexProfile

NAME

cgD3D9GetLatestVertexProfile − get the latest supported vertex shader version

SYNOPSIS

```
#include <Cg/cgD3D9.h>

CGprofile cgD3D9GetLatestVertexProfile( 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

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.

EXAMPLES

```
CGprofile bestVertexProfile = cgD3D9GetLatestVertexProfile();
```

ERRORS

None.

HISTORY

cgD3D9GetLatestVertexProfile was introduced in Cg 1.1.

SEE ALSO

cgD3D9GetLatestPixelProfile

NAME

cgD3D9GetManageTextureParameters − get the manage texture parameters flag from a context

SYNOPSIS

```
#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

```
CGbool manage = cgD3D9GetManageTextureParameters( pCtx );
if( manage )
   doSomething();
```

ERRORS

None.

HISTORY

cgD3D9GetManageTextureParameters was introduced in Cg 1.5.

SEE ALSO

cgD3D9SetManageTextureParameters

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

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

NAME

cgD3D9GetVertexDeclaration − get the default vertex declaration stream

SYNOPSIS

```
#include <Cg/cgD3D9.h>

CGbool cgD3D9GetVertexDeclaration( CGprogram program,
                                   D3DVERTEXELEMENT9 decl[MAXD3DDECLLENGTH] );
```

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 ***D3DDECL_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, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0 },
  { 0, 16, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_COLOR, 0 },
  { 0, 32, D3DDECLTYPE_FLOAT4, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0 },
  D3DDECL_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

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

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 **profile** is supported by the cgD3D9 library.

Returns **CG_FALSE** otherwise.

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

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

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 CGprogram using a vertex profile
// pixelProg is a CGprogram 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

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

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.

SEE ALSO

cgCreateState, cgSetPassState, cgResetPassState, cgCallStateValidateCallback

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

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

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
cgD3D9SetManageTextureParmeters( pCtx, CG_TRUE );
```

ERRORS

None.

HISTORY

cgD3D9SetManageTextureParameters was introduced in Cg 1.5.

SEE ALSO

cgD3D9GetManageTextureParameters, cgD3D9BindProgram

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

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* intialized elswhere
...
cgD3D9SetTexture(param, tex);
```

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

cgD3D9SetTexture was introduced in Cg 1.1.

SEE ALSO

cgD3D9SetSamplerState, cgD3D9SetTextureWrapMode

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

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

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]
...
// intialize 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

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]
...
// intialize the data for each parameter
D3DXVECTOR3 paramData(1,2,3);
float arrayData[2][2][2] =
{
    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

NAME

cgD3D9SetUniformMatrix − set the values of a uniform matrix parameter

SYNOPSIS

```
#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

```
// matrixParam is a CGparameter handle of type float3x2
// arrayParam is a CGparameter handle of type float4x4[2]
...
// intialize 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);
// 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

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]
...
// intialize 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

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

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, cgGetErrorString, cgSetErrorCallback

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

NAME

cgD3D9UnloadAllPrograms − unload all D3D programs

SYNOPSIS

```
#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

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

```
// 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

NAME

cgD3D9ValidateVertexDeclaration − validate a custom D3D9 vertex declaration stream

SYNOPSIS

```
#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 declaraton already setup

CGbool ret = cgD3D9ValidateVertexDeclaration( program, Decl );
if( ret == CG_TRUE )
    printf( "Vertex declaration not compatable with the program's varying paramete
```

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

NAME

cgD3D8BindProgram − activate a program with D3D

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8BindProgram( CGprogram prog );
```

PARAMETERS

program A **CGprogram** handle, the program to activate with D3D.

RETURN VALUES

cgD3D8BindProgram returns **D3D_OK** if the function succeeds.

If the function fails due to a D3D call, that D3D failure code is returned.

DESCRIPTION

cgD3D8BindProgram does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8BindProgram was introduced in Cg 1.1.

SEE ALSO

cgD3D9BindProgram

NAME

cgD3D8EnableDebugTracing − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

void cgD3D8EnableDebugTracing( CGbool enable );
```

PARAMETERS

to-be-written

to-be-written

RETURN VALUES

None.

or

cgD3D8EnableDebugTracing returns *to-be-written*

DESCRIPTION

cgD3D8EnableDebugTracing does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8EnableDebugTracing was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8EnableParameterShadowing − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetTextureWrapMode( CGparameter param,
                                  DWORD       value );
```

PARAMETERS

to-be-written

to-be-written

RETURN VALUES

None.

or

cgD3D8EnableParameterShadowing returns *to-be-written*

DESCRIPTION

cgD3D8EnableParameterShadowing does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8EnableParameterShadowing was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetDevice − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

IDirect3DDevice8* cgD3D8GetDevice();
```

PARAMETERS

to-be-written

to-be-written

RETURN VALUES

None.

or

cgD3D8GetDevice returns *to-be-written*

DESCRIPTION

cgD3D8GetDevice does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8GetDevice was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetLastError − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8GetLastError();
```

PARAMETERS

None

RETURN VALUES

None.

```
or
```

cgD3D8GetLastError returns *to-be-written*

DESCRIPTION

cgD3D8GetLastError does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8GetLastError was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetLatestPixelProfile − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

CGprofile cgD3D8GetLatestPixelProfile();
```

PARAMETERS

to-be-written

to-be-written

RETURN VALUES

None.

or

cgD3D8GetLatestPixelProfile returns *to-be-written*

DESCRIPTION

cgD3D8GetLatestPixelProfile does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8GetLatestPixelProfile was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetLatestVertexProfile − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

CGprofile cgD3D8GetLatestVertexProfile();
```

PARAMETERS

None

RETURN VALUES

None.

```
or
```

cgD3D8GetLatestVertexProfile returns *to-be-written*

DESCRIPTION

cgD3D8GetLatestVertexProfile does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8GetLatestVertexProfile was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetOptimalOptions − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

char const* cgD3D8GetOptimalOptions( CGprofile profile );
```

PARAMETERS

profile Cg profile for which to get optimal options.

RETURN VALUES

None.

```
or
```

cgD3D8GetOptimalOptions returns *to-be-written*

DESCRIPTION

cgD3D8GetOptimalOptions does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8GetOptimalOptions was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8GetVertexDeclaration − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

cgD3D8GetVertexDeclaration prototype goes here.
CGbool cgD3D8GetVertexDeclaration( CGprogram prog,
                                   DWORD     decl[MAX_FVF_DECL_SIZE] );
```

PARAMETERS

to-be-written

to-be-written

RETURN VALUES

None.

or

cgD3D8GetVertexDeclaration returns *to-be-written*

DESCRIPTION

cgD3D8GetVertexDeclaration does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8GetVertexDeclaration was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8IsParameterShadowingEnabled − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

CGbool cgD3D8IsParameterShadowingEnabled( CGprogram prog );
```

PARAMETERS

prog Cg program for which to query if parameter shadowing is enabled.

RETURN VALUES

None.

```
or
```

cgD3D8IsParameterShadowingEnabled returns *to-be-written*

DESCRIPTION

cgD3D8IsParameterShadowingEnabled does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8IsParameterShadowingEnabled was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8IsProgramLoaded − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

CGbool cgD3D8IsProgramLoaded( CGprogram prog );
```

PARAMETERS

prog Cg program handle.

RETURN VALUES

None.

```
or
```

cgD3D8IsProgramLoaded returns *to-be-written*

DESCRIPTION

cgD3D8IsProgramLoaded does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8IsProgramLoaded was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8LoadProgram − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8LoadProgram( CGprogram    prog,
                           CGbool       paramShadowing,
                           DWORD        assemFlags,
                           DWORD        vshaderUsage,
                           const DWORD* vertexDecl );
```

PARAMETERS

prog Cg program handle.

paramShadowing
Boolean for whether parameter shadowing should occur.

assemFlags
Flags passed to the assembler.

vsharedUsage
to-be-written

vertexDecl
to-be-written

RETURN VALUES

None.

or

cgD3D8LoadProgram returns *to-be-written*

DESCRIPTION

cgD3D8LoadProgram does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8LoadProgram was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8ResourceToInputRegister − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

DWORD cgD3D8ResourceToInputRegister( CGresource resource );
```

PARAMETERS

resource *to-be-written*

RETURN VALUES

None.

or

cgD3D8ResourceToInputRegister returns *to-be-written*

DESCRIPTION

cgD3D8ResourceToInputRegister does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8ResourceToInputRegister was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetDevice − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetDevice( IDirect3DDevice8* pDevice );
```

PARAMETERS

pDevice *to-be-written*

RETURN VALUES

None.

```
or
```

cgD3D8SetDevice returns *to-be-written*

DESCRIPTION

cgD3D8SetDevice does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8SetDevice was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetTexture − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetTexture( CGparameter           param,
                          IDirect3DBaseTexture8* tex );
```

PARAMETERS

param Cg parameter handle.

tex *to-be-written*

RETURN VALUES

None.

```
or
```

cgD3D8SetTexture returns *to-be-written*

DESCRIPTION

cgD3D8SetTexture does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8SetTexture was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetTextureStageState − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetTextureStageState( CGparameter              param,
                                    D3DTEXTURESTAGESTATETYPE type,
                                    DWORD                    value );
```

PARAMETERS

param Cg parameter handle.

type *to-be-written*

value *to-be-written*

RETURN VALUES

None.

or

cgD3D8SetTextureStageState returns *to-be-written*

DESCRIPTION

cgD3D8SetTextureStageState does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8SetTextureStageState was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetTextureWrapMode − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetTextureWrapMode( CGparameter param,
                                  DWORD       value );
```

PARAMETERS

param Cg parameter handle.

value *to-be-written*

RETURN VALUES

None.

```
or
```

cgD3D8SetTextureWrapMode returns *to-be-written*

DESCRIPTION

cgD3D8SetTextureWrapMode does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8SetTextureWrapMode was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetUniform − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetUniform( CGparameter param,
                          const void* floats );
```

PARAMETERS

param Cg parameter handle.

floats *to-be-written*

RETURN VALUES

None.

```
or
```

cgD3D8SetUniform returns *to-be-written*

DESCRIPTION

cgD3D8SetUniform does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8SetUniform was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetUniformArray − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetUniformArray( CGparameter param,
                               DWORD       offset,
                               DWORD       numItems,
                               const void* values );
```

PARAMETERS

param Cg parameter handle.

offset *to-be-written*

numItems
to-be-written

values *to-be-written*

RETURN VALUES

None.

or

cgD3D8SetUniformArray returns *to-be-written*

DESCRIPTION

cgD3D8SetUniformArray does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8SetUniformArray was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetUniformMatrix − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetUniformMatrix( CGparameter      param,
                                const D3DMATRIX* matrix );
```

PARAMETERS

param Cg parameter handle.

matrix *to-be-written*

RETURN VALUES

None.

```
or
```

cgD3D8SetUniformMatrix returns *to-be-written*

DESCRIPTION

cgD3D8SetUniformMatrix does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8SetUniformMatrix was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8SetUniformMatrixArray − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8SetUniformMatrixArray( CGparameter      param,
                                     DWORD            offset,
                                     DWORD            numItems,
                                     const D3DMATRIX* matrices );
```

PARAMETERS

param Cg parameter handle.

offset *to-be-written*

numItems
to-be-written

matrices *to-be-written*

RETURN VALUES

None.

or

cgD3D8SetUniformMatrixArray returns *to-be-written*

DESCRIPTION

cgD3D8SetUniformMatrixArray does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8SetUniformMatrixArray was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8TranslateCGerror − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

const char* cgD3D8TranslateCGerror( CGerror error );
```

PARAMETERS

error Cg error code.

RETURN VALUES

None.

or

cgD3D8TranslateCGerror returns *to-be-written*

DESCRIPTION

cgD3D8TranslateCGerror does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8TranslateCGerror was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8TranslateHRESULT − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

const char* cgD3D8TranslateHRESULT( HRESULT hr );
```

PARAMETERS

hr *to-be-written*

RETURN VALUES

None.

or

cgD3D8TranslateHRESULT returns *to-be-written*

DESCRIPTION

cgD3D8TranslateHRESULT does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8TranslateHRESULT was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8TypeToSize − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

DWORD cgD3D8TypeToSize( CGtype type );
```

PARAMETERS

type Cg type enumerant.

RETURN VALUES

None.

```
or
```

cgD3D8TypeToSize returns *to-be-written*

DESCRIPTION

cgD3D8TypeToSize does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8TypeToSize was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8UnloadProgram − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

HRESULT cgD3D8UnloadProgram( CGprogram prog );
```

PARAMETERS

prog Cg program handle.

RETURN VALUES

None.

```
or
```

cgD3D8UnloadProgram returns *to-be-written*

DESCRIPTION

cgD3D8UnloadProgram does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

```
or
```

to-be-written

HISTORY

cgD3D8UnloadProgram was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

NAME

cgD3D8ValidateVertexDeclaration − *to-be-written*

SYNOPSIS

```
#include <Cg/cgD3D8.h>

CGbool cgD3D8ValidateVertexDeclaration( CGprogram    prog,
                                        const DWORD* decl );
```

PARAMETERS

prog Cg program handle.

decl *to-be-written*

RETURN VALUES

None.

or

cgD3D8ValidateVertexDeclaration returns *to-be-written*

DESCRIPTION

cgD3D8ValidateVertexDeclaration does *to-be-written*

EXAMPLES

```
// example code to-be-written
```

ERRORS

None.

or

to-be-written

HISTORY

cgD3D8ValidateVertexDeclaration was introduced in Cg 1.1.

SEE ALSO

function1text, function2text

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.

PROFILE OPTIONS

NumTemps=n
: Number of temporaries to use (from 12 to 32).

MaxInstructionSlots=n
: Maximum allowable (static) instructions. Not an issue for NVIDIA GPUs.

NoDependentReadLimit=b
: Boolean for whether a read limit exists.

NumTexInstructions=n
: Maximum number of texture instructions to generate. Not an issue for NVIDIA GPUs, but important for ATI GPUs (set it to 32).

NumMathInstructions=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).

MaxDrawBuffers=n
: Maximum draw buffers for use with **ARB_draw_buffers**. Set to 1 for NV3x GPUs. Use to 4 for NV4x or ATI GPUs.

MaxLocalParams=n
: Maximum allowable local parameters.

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

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.

PROFILE OPTIONS

NumTemps=n
: Number of temporaries to use (from 12 to 32).

MaxInstructions=n
: Maximum allowable (static) instructions.

MaxLocalParams=n
: Maximum allowable local parameters.

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

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

```
http://www.opengl.org/registry/specs/NV/register_combiners.txt
http://www.opengl.org/registry/specs/NV/register_combiners2.txt
http://www.opengl.org/registry/specs/NV/texture_shader.txt
http://www.opengl.org/registry/specs/NV/texture_shader2.txt
```

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.

Binding Semantics Name	Corresponding Data
COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0

```
TEX1                            Input texture coordinate sets 1
TEXCOORD1

TEX2                            Input texture coordinate sets 2
TEXCOORD2

TEX3                            Input texture coordinate sets 3
TEXCOORD3

FOGP                            Input fog color (XYZ) and factor (W)
FOG
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

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)
tex2Dproj(sampler2D, 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.

Texture lookup function	Texture coordinate swizzle
tex1Dproj	.xw/.ra
tex2Dproj	.xyw/.rga
texRECTproj	.xyw/.rga
tex3Dproj	.xyzw/.rgba
texCUBEproj	.xyzw/.rgba

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

```
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

```
tex3D_dp3x3(sampler3D tex,
            float3 str,
            float4 intermediate_coord1,
            float4 intermediate_coord2,
            float4 prevlookup)
```

```
texCUBE_dp3x3(samplerCUBE tex,
              float3 str,
              float4 intermediate_coord1,
              float4 intermediate_coord2,
              float4 prevlookup)
```

Performs the following

```
float3 newst = float3(dot(intermediate_coord1.xyz, prevlookup.xyz),
                      dot(intermediate_coord2.xyz, prevlookup.xyz),
                      dot(str, prevlookup.xyz));
return 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' NV_texture_shader instruction combinations.

texCUBE_reflect_dp3x3

```
texCUBE_reflect_dp3x3(uniform samplerCUBE tex,
                      float4 strq,
                      float4 intermediate_coord1,
                      float4 intermediate_coord2,
                      float4 prevlookup)
```

Performs the following

```
float3 E = float3(intermediate_coord2.w, intermediate_coord1.w, strq.w);
float3 N = float3(dot(intermediate_coord1.xyz, prevlookup.xyz),
                  dot(intermediate_coord2.xyz, prevlookup.xyz),
                  dot(strq.xyz, prevlookup.xyz));
return texCUBE(tex, 2 * dot(N, E) / 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' NV_texture_shader instruction combination.

texCUBE_reflect_eye_dp3x3

```
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));
return texCUBE(tex, 2 * dot(N, E) / 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, 'intermediate_coord2' are texture coordinates associated with the 'n-1' texture unit and 'eye' is the eye-ray vector. This function can be used generate the 'dot_product_reflect_cube_map_const_eye' NV_texture_shader instruction combination.

tex_dp3x2_depth

```
tex_dp3x2_depth(float3 str,
                float4 intermediate_coord,
                float4 prevlookup)
```

Performs the following

```
float z = dot(intermediate_coord.xyz, prevlookup.xyz);
float w = dot(str, prevlookup.xyz);
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.

EXAMPLES

The following examples illustrate how a developer can use Cg to achieve NV_texture_shader/NV_register_combiners functionality.

Example 1

```
 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 * (tex2D(normalMap, IN.texCoord1.xy) - 0.5);
    float3 light_vector = 2 * (IN.color.rgb - 0.5);
    float4 dot_result = saturate(dot(light_vector, normal.xyz).xxxx);
    return dot_result * diffuseTexColor;
}
```

Example 2

```
struct VertexOut {
    float4 texCoord0 : TEXCOORD0;
    float4 texCoord1 : TEXCOORD1;
    float4 texCoord2 : TEXCOORD2;
    float4 texCoord3 : TEXCOORD3;
};
```

```
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;
}
```

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

```
http://www.opengl.org/registry/specs/NV/fragment_program.txt
```

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 *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 **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.

```
Binding Semantics Name          Corresponding Data

COLOR                           Input primary color
COLOR0
COL
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:

```
Binding Semantic Name           Corresponding Data

TEXUNIT0                        Texture unit 0
TEXUNIT1                        Texture unit 1
...
TEXUNIT15                       Texture unit 15
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

DEPTH                           Output depth (float)
DEPR
```

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives *are* supported.

NAME

fp40 − OpenGL fragment profile for NV4x (GeForce 6xxx and 7xxx Series, NV4x-based Quadro FX, etc.)

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

```
http://www.opengl.org/registry/specs/NV/fragment_program2.txt
```

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 **fp30** profile correspond to the respectively named varying output semantics of the **vp30** profile.

Binding Semantics Name	Corresponding Data
COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0
TEX1 TEXCOORD1	Input texture coordinate sets 1

```
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

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:

```
Binding Semantic Name           Corresponding Data

TEXUNIT0                        Texture unit 0
TEXUNIT1                        Texture unit 1
...
TEXUNIT15                       Texture unit 15
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

DEPTH                           Output depth (float)
DEPR
```

STANDARD LIBRARY ISSUES

Functions that compute partial derivatives *are* supported.

NAME

glslf − OpenGL fragment profile for the OpenGL Shading Lanauge (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 OpenGL support for **OpenGL 2.0**.

PROFILE OPTIONS

None.

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

Binding Semantics Name	Corresponding Data	GLSL Equivalent
COLOR COLOR0 COL0 COL	Primary color (float4)	gl_Color
COLOR1 COL1	Secondary color (float4)	gl_SecondaryColor
TEXCOORD TEXCOORD# TEX#	Texture coordinate set 0 Texture coordinate set #	gl_TexCoord[0] gl_TexCoord[#]

UNIFORM INPUT SEMANTICS

The Cg profiles for GLSL provide access to all the uniform constants and variables documented in Section 7.4 (Built-in Constants) and 7.5 (Built-in Uniform State) respectively of the OpenGL Shading Language specification found at:

```
http://www.opengl.org/documentation/glsl/
http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf
```

Example:

```
glslf void main(float4 color : COLOR,
                out float4 ocol : COLOR)
{
  ocol.xyz = mul(gl_NormalMatrix, color.xyz);
  ocol.w = 1;
}
```

OUTPUT SEMANTICS

The following standard fragment output semantics are supported:

Binding Semantics Name	Corresponding Data	GLSL Equivalent
COLOR COLOR0 COL0 COL	Output color (float4)	gl_FragColor
DEPTH DEPR	Output depth (float)	gl_FragDepth

STANDARD LIBRARY ISSUES

Fragment program Cg standard library routines are available.

SEE ALSO

The glslv profile is a similar translation profile for GLSL but for vertex shaders.

The **glslf** profile is similar to the DirectX 9 hlslf profile that translates Cg into Microsoft's High Level Shader Language (HLSL) for pixel shaders.

NAME

glslv − OpenGL vertex profile for the OpenGL Shading Lanauge (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

None.

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

Binding Semantics Name	Corresponding Data	GLSL Equivalent
POSITION ATTR0	Object-space position	gl_Vertex
NORMAL ATTR2	Object-space normal	gl_Normal
COLOR COLOR0 ATTR3 DIFFUSE	Primary color (float4)	gl_Color
COLOR1 SPECULAR ATTR4	Secondary color (float4)	gl_SecondaryColor
FOGCOORD ATTR5	Fog coordinate	gl_FogCoord
TEXCOORD# ATTR8 ATTR9 ATTR10 ATTR11 ATTR12 ATTR13 ATTR14 ATTR15	Texture coordinate set # Texture coordinate set 0 Texture coordinate set 1 Texture coordinate set 2 Texture coordinate set 3 Texture coordinate set 4 Texture coordinate set 5 Texture coordinate set 6 Texture coordinate set 7	gl_MultiTexCoord#

UNIFORM INPUT SEMANTICS

The Cg profiles for GLSL provide access to all the uniform constants and variables documented in Section 7.4 (Built-in Constants) and 7.5 (Built-in Uniform State) respectively of the OpenGL Shading Language specification found at:

```
http://www.opengl.org/documentation/glsl/
http://www.opengl.org/registry/doc/GLSLangSpec.Full.1.20.8.pdf
```

Example:

```
glslv void main(float4 position : POSITION,
                out float4 opos : POSITION)
{
  opos = mul(gl_ModelViewMatrix, position);
}
```

OUTPUT SEMANTICS

Binding Semantics Name	Corresponding Data	GLSL Equivalent
POSITION HPOS	Clip-space position	gl_Position
COLOR COLOR0 COL0 COL	Front primary color	gl_FrontColor
COLOR1 COL1	Front secondary color	gl_FrontSecondaryColor
BCOL0	Back primary color	gl_BackColor
BCOL1	Back secondary color	gl_BackSecondaryColor
CLPV	Clip vertex	gl_ClipVertex
TEXCOORD# TEX#	Texture coordinate set #	gl_TexCoord[#}
FOGC FOG	Fog coordinate	gl_FogFragCoord
PSIZE PSIZ	Point size	gl_PointSize

STANDARD LIBRARY ISSUES

Vertex program Cg standard library routines are available.

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

The glslf profile is a similar translation profile for GLSL but for fragment shaders.

The **glslv** profile is similar to the Direct3D hlslv profile that translates Cg into Microsoft's High Level

Shading Language (HLSL) for vertex shaders.

NAME

gp4 − OpenGL profiles for G8x (GeForce 8xxx Series, G8x-based Quadro FX, etc.)

SYNOPSYS

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 **EXT_gpu_program4** these profiles are named the **gp4** profiles. For more details refer to each profile documentation.

SEE ALSO

gp4vp, gp4gp, gp4fp.de Sh

\$1

NAME

gp4fp − OpenGL fragment profile for G8x (GeForce 8xxx Series, G8x-based Quadro FX, etc.)

SYNOPSIS

```
gp4fp
```

DESCRIPTION

This OpenGL profile corresponds to the per-fragment functionality introduced by the GeForce 8800 and other G8x-based NVIDIA GPUs.

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 vertex domain-specific aspects documented in the **NV_program_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).

3D API DEPENDENCIES

Requires OpenGL support for the **NV_gpu_program4** extension. This extension was introduced by the GeForce 6800 and other G8x-based GPUs.

OpenGL Extension Specifications

Programmability:

```
http://www.opengl.org/registry/specs/NV/gpu_program4.txt
http://www.opengl.org/registry/specs/NV/fragment_program4.txt
```

New texture samplers:

```
http://www.opengl.org/registry/specs/EXT/texture_array.txt
http://www.opengl.org/registry/specs/EXT/texture_buffer_object.txt
```

Draw buffers:

```
http://www.opengl.org/registry/specs/ARB/draw_buffers.txt
http://www.opengl.org/registry/specs/ATI/draw_buffers.txt
```

PROFILE OPTIONS

Common GP4 Options

fastimul
: Assume integer multiply inputs have at most 24 significant bits. Example: ‘‘−po fastimul’’

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

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

NAME

gp4gp − OpenGL geometry profile for G8x (GeForce 8xxx Series, G8x-based Quadro FX, etc.)

SYNOPSIS

```
gp4gp
```

DESCRIPTION

This OpenGL profile corresponds to the per-primitive functionality introduced by the GeForce 8800 and other G8x-based NVIDIA GPUs.

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_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 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).

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.

OpenGL Extension Specifications

Programmability:

```
http://www.opengl.org/registry/specs/NV/gpu_program4.txt
http://www.opengl.org/registry/specs/NV/geometry_program4.txt
```

New texture samplers:

```
http://www.opengl.org/registry/specs/EXT/texture_array.txt
http://www.opengl.org/registry/specs/EXT/texture_buffer_object.txt
```

PROFILE OPTIONS

Common GP4 Options

fastimul
: Assume integer multiply inputs have at most 24 significant bits. Example: ‘‘−po fastimul’’

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 proporational 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

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

NAME

gp4vp − OpenGL vertex profile for G8x (GeForce 8xxx Series, G8x-based Quadro FX, etc.)

SYNOPSIS

```
gp4vp
```

DESCRIPTION

This OpenGL profile corresponds to the per-vertex functionality introduced by the GeForce 8800 and other G8x-based NVIDIA GPUs.

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.

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.

Programmability:

```
http://www.opengl.org/registry/specs/NV/gpu_program4.txt
http://www.opengl.org/registry/specs/NV/vertex_program4.txt
```

New texture samplers:

```
http://www.opengl.org/registry/specs/EXT/texture_array.txt
http://www.opengl.org/registry/specs/EXT/texture_buffer_object.txt
```

PROFILE OPTIONS

Common GP4 Options

fastimul
: Assume integer multiply inputs have at most 24 significant bits. Example: ‘‘−po fastimul’’

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

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:

```
http://msdn2.microsoft.com/en-us/library/bb509561.aspx
```

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.

Binding Semantics Name	Corresponding Data
COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0
TEX1 TEXCOORD1	Input texture coordinate sets 1

```
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:

```
Binding Semantic Name           Corresponding Data

TEXUNIT0                        Texture unit 0
TEXUNIT1                        Texture unit 1
...
TEXUNIT15                       Texture unit 15
```

OUTPUT SEMANTICS

```
COLOR                           Output color (float4)
COLOR0
COL0
COL

DEPTH                           Output depth (float)
DEPR
```

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

The **hlslf** profile is similar to the OpenGL glslf profile that translates Cg into the OpenGL Shading Language (GLSL) for fragment shaders.

NAME

hlslv − Translation profile to DirectX 9's High Level Shader Language for pixel 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:

```
http://msdn2.microsoft.com/en-us/library/bb509561.aspx
```

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.

Binding Semantics Name	Corresponding Data
POSITION	Object-space position
NORMAL	Object-space normal
COLOR COLOR0 DIFFUSE	Primary color (float4)
COLOR1 SPECULAR	Secondary color (float4)
FOGCOORD	Fog coordinate
TEXCOORD#	Texture coordinate set #

UNIFORM INPUT SEMANTICS

to-be-written

OUTPUT SEMANTICS

Binding Semantics Name	Corresponding Data
POSITION HPOS	Clip-space position
COLOR COLOR0 COL0 COL	Front primary color
COLOR1 COL1	Front secondary color
TEXCOORD# TEX#	Texture coordinate set #
FOGC FOG	Fog coordinate
PSIZE PSIZ	Point size

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

The hlslf profile is a similar translation profile for HLSL but for pixel (fragment) shaders.

The **hlslv** profile is similar to the OpenGL glslv profile that translates Cg into the OpenGL Shading Language (GLSL) for vertex shaders.

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:

```
http://msdn2.microsoft.com/en-us/library/bb219842.aspx
```

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

The OpenGL fp20 profile is rougly equivalent but more powerful than the **ps_1_1** profile.

The Direct3D ps_1_2 and ps_1_3 profiles extend the **ps_1_1** profile.

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:

```
http://msdn2.microsoft.com/en-us/library/bb219842.aspx
```

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

The OpenGL fp20 profile is rougly equivalent but more powerful than the **ps_1_2** profile.

The Direct3D ps_1_3 profile extends the **ps_1_2** profile.

NAME

ps_1_3 − Direct3D Shader Model 1.1 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:

```
http://msdn2.microsoft.com/en-us/library/bb219842.aspx
```

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

The OpenGL fp20 profile is rougly equivalent to the **ps_1_3** profile.

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:

```
http://msdn2.microsoft.com/en-us/library/bb219843.aspx
```

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.

Binding Semantics Name	Corresponding Data
COLOR	Input primary color
COLOR0	
COL	
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:

```
Binding Semantic Name           Corresponding Data

TEXUNIT0                        Texture unit 0
TEXUNIT1                        Texture unit 1
...
TEXUNIT15                       Texture unit 15
```

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

The OpenGL arbfp1 profile is rougly equivalent to the **ps_2_0** profile.

The Direct3D ps_2_x profile extends the **ps_2_0**.

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:

```
http://msdn2.microsoft.com/en-us/library/bb219844.aspx
```

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:

```
D3DD3DPSHADERCAPS2_0_ARBITRARYSWIZZLE
D3DD3DPSHADERCAPS2_0_GRADIENTINSTRUCTIONS
D3DD3DPSHADERCAPS2_0_PREDICATION
D3DD3DPSHADERCAPS2_0_NODEPENDENTREADLIMIT
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_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.

```
Binding Semantics Name          Corresponding Data
```

COLOR COLOR0 COL COL0	Input primary color
COLOR1 COL1	Input secondary color
TEX0 TEXCOORD0	Input texture coordinate sets 0
TEX1 TEXCOORD1	Input texture coordinate sets 1
TEX2 TEXCOORD2	Input texture coordinate sets 2
TEX3 TEXCOORD3	Input texture coordinate sets 3
TEX4 TEXCOORD4	Input texture coordinate sets 4
TEX5 TEXCOORD5	Input texture coordinate sets 5
TEX6 TEXCOORD6	Input texture coordinate sets 6
TEX7 TEXCOORD7	Input texture coordinate sets 7
FOGP FOG	Input fog color (XYZ) and factor (W)

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

Binding Semantic Name	Corresponding Data
TEXUNIT0	Texture unit 0
TEXUNIT1	Texture unit 1
...	
TEXUNIT15	Texture unit 15

OUTPUT SEMANTICS

COLOR COLOR0 COL0 COL	Output color (float4)

`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.

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:

```
http://msdn2.microsoft.com/en-us/library/bb219845.aspx
```

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.

```
Binding Semantics Name          Corresponding Data

COLOR                           Input primary color
COLOR0
COL
COL0

COLOR1                          Input secondary color
COL1

TEX0                            Input texture coordinate sets 0
TEXCOORD0
```

TEX1 TEXCOORD1	Input texture coordinate sets 1
TEX2 TEXCOORD2	Input texture coordinate sets 2
TEX3 TEXCOORD3	Input texture coordinate sets 3
TEX4 TEXCOORD4	Input texture coordinate sets 4
TEX5 TEXCOORD5	Input texture coordinate sets 5
TEX6 TEXCOORD6	Input texture coordinate sets 6
TEX7 TEXCOORD7	Input texture coordinate sets 7
FOGP FOG	Input fog color (XYZ) and factor (W)

UNIFORM INPUT SEMANTICS

Sixteen texture units are supported:

Binding Semantic Name	Corresponding Data
TEXUNIT0	Texture unit 0
TEXUNIT1	Texture unit 1
...	
TEXUNIT15	Texture unit 15

OUTPUT SEMANTICS

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

STANDARD LIBRARY ISSUES

This profile may have limits on the number of dependent texture fetches.

SEE ALSO

The OpenGL fp40 profile is rougly equivalent to the **ps_3_0** profile.

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

```
http://www.opengl.org/registry/specs/NV/vertex_program.txt
http://www.opengl.org/registry/specs/NV/vertex_program1_1.txt
```

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

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

```
http://www.opengl.org/registry/specs/NV/vertex_program2.txt
```

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

NAME

vp40 − OpenGL vertex profile for NV4x (GeForce 6xxx and 7xxx Series, NV4x-based Quadro FX, etc.)

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

```
http://www.opengl.org/registry/specs/NV/vertex_program3.txt
http://www.opengl.org/registry/specs/ARB/vertex_program.txt
```

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, modulos 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.

Binding Semantics Name	Corresponding Data
POSITION, ATTR0	Input Vertex, Generic Attribute 0
BLENDWEIGHT, ATTR1	Input vertex weight, Generic Attribute 1
NORMAL, ATTR2	Input normal, Generic Attribute 2
DIFFUSE, COLOR0, ATTR3	Input primary color, Generic Attribute 3
SPECULAR, COLOR1, ATTR4	Input secondary color, Generic Attribute 4
TESSFACTOR, FOGCOORD, ATTR5	Input fog coordinate, Generic Attribute 5
PSIZE, ATTR6	Input point size, Generic Attribute 6
BLENDINDICES, ATTR7	Generic Attribute 7
TEXCOORD0-TEXCOORD7, ATTR8-ATTR15	Input texture coordinates (texcoord0-texcoord7) Generic Attributes 8-15
TANGENT, ATTR14	Generic Attribute 14
BINORMAL, ATTR15	Generic Attribute 15

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

Binding Semantics Name	Corresponding Data
POSITION, HPOS	Output position
PSIZE, PSIZ	Output point size
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

Standard library routines for texture cube map and rectangle samplers are not supported by vp40.

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

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.

```
// Handbook of Mathematical Functions
// M. Abramowitz and I.A. Stegun, Ed.

// Absolute error <= 6.7e-5
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

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

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

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

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

atan for a **float** scalar could be implemented like this.

```
float atan(float x) {
    return atan2(x, float(1));
}
```

atan2 is typically implemented as an approximation.

PROFILE SUPPORT

atan is supported in all profiles but fp20.

SEE ALSO

abs, acos, asin, atan2. sqrt, tan

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

NAME

ceil − returns smallest integer not less than a scalar or each vector component.

SYNOPSIS

```
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.

```
float ceil(float v)
{
  return -floor(-v);
}
```

PROFILE SUPPORT

ceil is supported in all profiles except fp20.

SEE ALSO

floor

NAME

clamp − returns smallest integer not less than a scalar or each vector component.

SYNOPSIS

```
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);
```

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.

```
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

NAME

cos − returns cosine of scalars and vectors.

SYNOPSIS

```
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 **float** scalar could be implemented by an approximation like this.

```
cos(float a)
{
  /* C simulation gives a max absolute error of less than 1.8e-7 */
  const float4 c0 = float4( 0.0,           0.5,           1.0,          0.0
  const float4 c1 = float4( 0.25,         -9.0,           0.75,         0.15915
  const float4 c2 = float4( 24.9808039603, -24.9808039603, -60.1458091736, 60.1458
  const float4 c3 = float4( 85.4537887573, -85.4537887573, -64.9393539429, 64.9393
  const float4 c4 = float4( 19.7392082214, -19.7392082214, -1.0,          1.0

  /* r0.x = cos(a) */
  float3 r0, r1, r2;
```

```
r1.x  = c1.w * a;                       // normalize input
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.xyx * r0 + c2.zwz;           // start power series
r1    =     r1 * r0 + c3.xyx;
r1    =     r1 * r0 + c3.zwz;
r1    =     r1 * r0 + c4.xyx;
r1    =     r1 * r0 + c4.zwz;
r0.x  = dot( r1, -r2 );                 // range extract

return r0.x;
```

PROFILE SUPPORT

cos is fully supported in all profiles unless otherwise specified.

cos is supported via an approximation (shown above) in the vs_1, vp20, and arbvp1 profiles.

cos is unsupported in the fp20, ps_1_1, ps_1_2, and ps_1_3 profiles.

SEE ALSO

acos, dot, frac, sin, tan

NAME

cosh − returns hyperbolic cosine of scalars and vectors.

SYNOPSIS

```
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.

```
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

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

NAME

cos − 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

NAME

cos − 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

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

NAME

determinant − returns 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 **determinant** functions can be implemented like this:

```
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_m20_m21
           - 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_m33_m30_m31)
             + 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));
}
```

PROFILE SUPPORT

determinant is supported in all profiles. However profiles such as fp20 and ps_2 without native floating-point will have problems computing the larger determinants and may have ranges issues computing even small determinants.

SEE ALSO

mul, transpose

NAME

dot − returns the scalar dot product of two vectors

SYNOPSIS

```
float  dot(float  a, float  b);
float1 dot(float1 a, float1 b);
float2 dot(float2 a, float2 b);
float3 dot(float3 a, float3 b);
float4 dot(float4 a, float4 b);

half   dot(half  a, half  b);
half1  dot(half1 a, half1 b);
half2  dot(half2 a, half2 b);
half3  dot(half3 a, half3 b);
half4  dot(half4 a, half4 b);

fixed  dot(fixed  a, fixed  b);
fixed1 dot(fixed1 a, fixed1 b);
fixed2 dot(fixed2 a, fixed2 b);
fixed3 dot(fixed3 a, fixed3 b);
fixed4 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:

```
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, mul

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 = exp(a[i]);  // this is the ANSI C standard library exp()
  }
  return i;
}
```

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

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 = exp2(a[i]);  // this is the ANSI C standard library exp2()
  }
  return i;
}
```

exp2 is typically implemented with a native base-2 exponentional 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

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 Peturbed normal vector.

I Incidence vector (typically a direction vector from the eye to a vertex).

Ng Geometric normal vector (for some facet the peturbed 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 funciton of the same name though the RenderMan version has only two parameters.

REFERENCE IMPLEMENTATION

faceforward for **float3** vectors could be implemented this way:

```
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

NAME

floatToIntBits − returns the 32−bit integer representation of an IEEE 754 floating-point scalar or vector

SYNOPSIS

```
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 cannonicalized 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 jave.lang.Float method of the same name. See:

```
http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Float.html
```

floatToIntBits requires instructions to be generated to cannonicalize NaN values so **floatToIntBits** is typically more expensive than **floatToRawIntBits**.

REFERENCE IMPLEMENTATION

floatToIntBits operates consistent with the following ANSI C code:

```
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

NAME

floatToRawIntBits − returns the raw 32−bit integer representation of an IEEE 754 floating-point scalar or vector

SYNOPSIS

```
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 jave.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 **floatToRawIntBits** so it has no instruction cost.

REFERENCE IMPLEMENTATION

floatToRawIntBits operates consistent with the following ANSI C code:

```
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

NAME

floor − returns largest integer not greater than a scalar or each vector component.

SYNOPSIS

```
float  floor(float a);
float1 floor(float1 a);
float2 floor(float2 a);
float3 floor(float3 a);
float4 floor(float4 a);

half   floor(half a);
half1  floor(half1 a);
half2  floor(half2 a);
half3  floor(half3 a);
half4  floor(half4 a);

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.

```
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

NAME

fmod − returns the remainder of *x*/*y* with the same sign as *x*

SYNOPSIS

```
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:

```
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

NAME

frac − returns the fractional portion of a scalar or each vector component.

SYNOPSIS

```
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.

```
float frac(float v)
{
  return v - floor(v);
}
```

PROFILE SUPPORT

frac is supported in all profiles except fp20.

SEE ALSO

ceil, floor, round, trunc

NAME

fwidth − returns sum of approximate window-space partial derivatives magnitudes

SYNOPSIS

```
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.*

.PP 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

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 x)
{
  // 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

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 x)
{
  // 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

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);

bool  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.

```
bool3 isnan(float3 x)
{
  // 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, isinf

NAME

ldexp − returns *x* times 2 rained to *n*

SYNOPSIS

```
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:

```
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

NAME

length − return scalar Euclidean length of a vector

SYNOPSIS

```
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.

```
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

NAME

lerp − returns linear interpolation of two scalars or vectors based on a weight

SYNOPSIS

```
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 with *w* is one.

b Vector or scalar to weight; returned with *w* is zero.

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 one and returns *b* when *w* is zero.

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

NAME

log − returns the natural logarithm of scalars and vectors

SYNOPSIS

```
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

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

```
float3 log(float3 a)
{
  float3 rv;
  int i;

  for (i=0; i<3; i++) {
    rv = log(a[i]);  // this is the ANSI C standard library log()
  }
  return i;
}
```

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

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 = log10(a[i]);  // this is the ANSI C standard library log10()
  }
  return i;
}
```

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

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;
  int i;

  for (i=0; i<3; i++) {
    rv = log2(a[i]);  // this is the ANSI C standard library log2()
  }
  return i;
}
```

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

NAME

max − returns the maximum of two scalars or each respective component of two vectors

SYNOPSIS

```
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

max for **float3** vectors could be implemented this way:

```
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

max is supported in all profiles. **max** is implemented as a compiler built-in.

Support in the fp20 is limited.

SEE ALSO

clamp, min

NAME

min − returns the minimum of two scalars or each respective component of two vectors

SYNOPSIS

```
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:

```
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

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 algebrically equal (if not necessarily numerically equal):

```
mul(M,v) == mul(v, tranpose(M))
mul(v,M) == mul(tranpose(M), v)
```

REFERENCE IMPLEMENTATION

mul for a **float4x3** matrix by a **float3** column vector could be implemented this way:

```
float4 mul(float4x3 M, float3 v)
{
  float4 r;

  r.x = dot( M._m00_m01_m02, v );
  r.y = dot( M._m10_m11_m12, v );
  r.z = dot( M._m20_m21_m22, v );
  r.w = dot( M._m30_m31_m32, v );

  return r;
}
```

PROFILE SUPPORT

mul is supported in all profiles.

The **fixed3** matrix-by-vector and vector-by-matrix multiplies are very efficient in the fp20 and fp30 profiles.

SEE ALSO

cross, dot, transpose

NAME

radians − converts values of scalars and vectors from degrees to radians

SYNOPSIS

```
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);
```

PARAMETERS

a Vector or scalar of which to convert from degrees to radians.

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.

REFERENCE IMPLEMENTATION

radians for a **float** scalar could be implemented like this.

```
float radians(float a)
{
  return 0.017453292 * a;
}
```

PROFILE SUPPORT

radians is supported in all profiles except fp20.

SEE ALSO

cos, degrees, sin, tan

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:

```
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

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:

```
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

NAME

round − returns the rounded value of scalars or vectors

SYNOPSIS

```
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

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.

```
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

NAME

sin − returns sine of scalars and vectors.

SYNOPSIS

```
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

sin is best implemented as a native sine instruction, however **sin** for a **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,          0.15915494309
  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.xyx * r0 + c2.zwz;          // start power series
  r1    =     r1 * r0 + c3.xyx;
  r1    =     r1 * r0 + c3.zwz;
  r1    =     r1 * r0 + c4.xyx;
  r1    =     r1 * r0 + c4.zwz;
  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, 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, tan

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 **float** could be implemented like this.

```
float sinh(float x)
{
  return 0.5 * (exp(x)-exp(-x));
}
```

PROFILE SUPPORT

sinh is supported in all profiles except fp20.

SEE ALSO

acos, cos, cosh, exp, tanh

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, 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

NAME

tanh − returns hyperbolic tangent of scalars and vectors.

SYNOPSIS

```
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);
```

PARAMETERS

a Vector or scalar of which to determine the hyperbolic tangent.

DESCRIPTION

Returns the hyperbolic tangent of *a*.

For vectors, the returned vector contains the hyperbolic tangent of each element of the input vector.

REFERENCE IMPLEMENTATION

tanh for a scalar **float** could be implemented like this.

```
float tanh(float x)
{
  float exp2x = exp(2*x);
  return (exp2x - 1) / (exp2x + 1);
}
```

PROFILE SUPPORT

tanh is supported in all profiles except fp20.

SEE ALSO

atan, atan2, cosh, exp, sinh, tan

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

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 t
```

PARAMETERS

samp	Sampler array to look up.
s	Coordinates to perform the lookup. The value used to select the layer is passed immediatelly 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

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

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

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.de Sh

\$1

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.de Sh

\$1

NAME

tex1DARRAYlod − performs a texture lookup with a specified level of detail in a given sampler array.

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	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 level of detail 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 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.de Sh

\$1

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)
unsgined 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 them 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

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.de Sh

\$1

NAME

tex1Dbias − performs a texture lookup with bias in a given sampler.

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	Coordinates to perform the lookup. 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*.

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 suppported in gp4 and newer profiles.

SEE ALSO

tex1Dlod, tex1Dcmpbias.de Sh

\$1

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.de Sh

\$1

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 suppported in gp4 and newer profiles.

SEE ALSO

tex1Dlod, tex1Dcmpbias

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.de Sh

\$1

NAME

tex1Dlod − performs a texture lookup with a specified level of detail in a given sampler.

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	Coordinates to perform the lookup. The level of detail 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 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 suppported in gp4 and newer profiles.

SEE ALSO

tex1Dbias, tex1Dcmplod.de Sh

\$1

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)
unsgined 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 them 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

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.de Sh

\$1

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

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 t
```

PARAMETERS

samp	Sampler array to look up.
s	Coordinates to perform the lookup. The value used to select the layer is passed immediatelly 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

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

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.de Sh

\$1

NAME

tex2DARRAYlod − performs a texture lookup with a specified level of detail in a given sampler array.

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	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 level of detail 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 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.de Sh

\$1

NAME

tex2DARRAYproj − performs a texture lookup with projection in a given sampler array.

SYNOPSIS

```
float4 tex2DARRAYproj(sampler2DARRAY samp, float3 s)
float4 tex2DARRAYproj(sampler2DARRAY samp, float3 s, int texelOff)

int4 tex2DARRAYproj(isampler2DARRAY samp, float3 s)
int4 tex2DARRAYproj(isampler2DARRAY samp, float3 s, int texelOff)

unsigned int4 tex2DARRAYproj(usampler2DARRAY samp, float3 s)
unsigned int4 tex2DARRAYproj(usampler2DARRAY 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.

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 them used in the lookup.

PROFILE SUPPORT

tex2DARRAYproj is only supported in gp4 and newer profiles.

SEE ALSO

tex2D, tex2Dproj

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.de Sh

\$1

NAME

tex2Dbias − performs a texture lookup with bias in a given sampler.

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	Coordinates to perform the lookup. 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*.

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 suppported in gp4 and newer profiles.

SEE ALSO

tex2Dlod, tex2Dcmpbias.de Sh

\$1

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, 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

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.de Sh

\$1

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

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

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

NAME

tex2Dfetch − performs an unfiltered texture lookup in a given sampler.

SYNOPSIS

```
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.de Sh

\$1

NAME

tex2Dlod − performs a texture lookup with a specified level of detail in a given sampler.

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	Coordinates to perform the lookup. The level of detail 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 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 suppported in gp4 and newer profiles.

SEE ALSO

tex2Dbias, tex2Dcmplod.de Sh

\$1

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)

unsigned int4 tex2Dproj(usampler2D samp, float3 s)
unsgined 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 them 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

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.de Sh

\$1

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 suppported in gp4 and newer profiles.

SEE ALSO

tex3Dbias, tex3Dlod, tex3Dproj

NAME

tex3Dbias − performs a texture lookup with bias in a given sampler.

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	Coordinates to perform the lookup. 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*.

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 suppported in gp4 and newer profiles.

SEE ALSO

tex3Dlod.de Sh

\$1

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.de Sh

\$1

NAME

tex3Dlod − performs a texture lookup with a specified level of detail in a given sampler.

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)
unsgined int4 tex3Dlod(usampler3D samp, float4 s, int texelOff)
```

PARAMETERS

samp	Sampler to lookup.
s	Coordinates to perform the lookup. The level of detail 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 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 suppported in gp4 and newer profiles.

SEE ALSO

tex3Dbias.de Sh

\$1

NAME

tex3Dproj − performs a texture lookup with projection in a given sampler. May perform a shadow comparison if argument for shadow comparison is provided.

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. 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 them 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

tex3Dproj 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

tex3D

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.de Sh

\$1

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:

```
http://developer.download.nvidia.com/opengl/specs/GL_EXT_texture_buffer_object.txt
```

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

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:

```
http://developer.download.nvidia.com/opengl/specs/GL_EXT_texture_buffer_object.txt
```

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, texBUFsize

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 suppported in gp4 and newer profiles.

SEE ALSO

texCUBEbias, texCUBElod, texCUBEproj

NAME

texCUBEbias − performs a texture lookup with bias in a given sampler.

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	Coordinates to perform the lookup. 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*.

PROFILE SUPPORT

texCUBEbias 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

texCUBElod.de Sh

\$1

NAME

texCUBElod − performs a texture lookup with a specified level of detail in a given sampler.

SYNOPSIS

```
float4 texCUBElod(samplerCUBE samp, float4 s)

int4 texCUBElod(isamplerCUBE samp, float4 s)

unsigned int4 texCUBElod(usamplerCUBE samp, float4 s)
```

PARAMETERS

samp	Sampler to lookup.
s	Coordinates to perform the lookup. The level of detail should be passed as the last component of the coordinate vector.

DESCRIPTION

Performs a texture lookup with a specified level of detail in sampler *samp* using coordinates *s*.

PROFILE SUPPORT

texCUBElod 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

texCUBEbias.de Sh

\$1

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 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 them 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

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

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 texel
```

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

NAME

texRECTbias − performs a texture lookup with bias in a given sampler.

SYNOPSIS

```
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	Coordinates to perform the lookup. 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*.

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 suppported in gp4 and newer profiles.

SEE ALSO

texRECTlod.de Sh

\$1

NAME

texRECTfetch − performs an unfiltered texture lookup in a given sampler.

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

texRECT, texRECTARRAYfetch.de Sh

\$1

NAME

texRECTlod − performs a texture lookup with a specified level of detail in a given sampler.

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	Coordinates to perform the lookup. The level of detail 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 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 suppported in gp4 and newer profiles.

SEE ALSO

texRECTbias.de Sh

\$1

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 them 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

NAME

texRECTsize − returns the size of a given texture image for a given level of detail.

SYNOPSIS

```
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.de Sh

\$1

NAME

transpose − returns transpose matrix of a matrix

SYNOPSIS

```
float4x4 transpose(float4x4 A)
float3x4 transpose(float4x3 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 tranpose.

DESCRIPTION

Returns the transpose of the matrix *A*.

REFERENCE IMPLEMENTATION

transpose for a **float4x3** matrix can be implemented like this:

```
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

NAME

trunc − returns largest integer not greater than a scalar or each vector component.

SYNOPSIS

```
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);
```

PARAMETERS

x Vector or scalar which to truncate.

DESCRIPTION

Returns the integral value nearest to but no larger in magnitude than x.

REFERENCE IMPLEMENTATION

trunc for a **float3** vector could be implemented like this.

```
float3 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;
}
```

PROFILE SUPPORT

trunc is supported in all profiles except fp20.

SEE ALSO

ceil, floor, round

NAME

AlphaFunc − 3D API alpha function

USAGE

AlphaFunc = float2(function, reference_value)

VALID ENUMERANTS

function: Never, Less, LEqual, Equal, Greater, NotEqual, GEqual, Always

DESCRIPTION

Set the OpenGL or Direct3D alpha function and reference value state. See the OpenGL glAlphaFunc manual page for details. See the D3DRS_ALPHAREF and D3DRS_ALPHAFUNC render state descriptions for DirectX 9.

The standard reset callback sets the AlphaFunc state to float2(Always, 0.0).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaTestEnable, DepthFunc, StencilFunc

NAME

AlphaTestEnable − 3D API alpha test enable

USAGE

AlphaTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D alpha test enable. 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

BlendEnable, DepthTestEnable, StencilTestEnable

NAME

AutoNormalEnable − 3D API auto normal test enable

USAGE

AutoNormalEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL auto normal enable for evaluators. See the GL_AUTO_NORMAL description on the OpenGL glEnable manual page for details.

The standard reset callback sets the AutoNormalEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Evaluators are not supported by Direct3D so this state is parsed but has no effect.

SEE ALSO

BlendEnable, DepthTestEnable, StencilTestEnable

NAME

BlendEnable − 3D API blend enable

USAGE

BlendEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D blend enable. 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).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

AlphaTestEnable, DepthTestEnable, StencilTestEnable

NAME

ClipPlaneEnable − 3D API clip plane enable

USAGE

ClipPlaneEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D clip plane enable. See the GL_CLIP_PLANEn description on the OpenGL glEnable manual page for details. See the D3DRS_CLIPPLANEENABLE render state description.

The standard reset callback sets the ClipPlaneEnable state to bool(false).

The index ndx identifies the clip plane must be greater or equal to zero and less than the value of GL_MAX_CLIP_PLANES (typically 6). DirectX 9 supports 6 clip planes.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightingEnable, LightEnable

NAME

ColorLogicOpEnable − 3D API color logical operation enable

USAGE

ColorLogicOpEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL color logical operation (logic-op) enable. 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 1.1

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported.

SEE ALSO

BlendEnable, DepthTestEnable, LogicOp, StencilTestEnable

NAME

CullFaceEnable − 3D API cull face enable

USAGE

CullFaceEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL cull face enable. 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

DirectX 9 Not supported. DirectX 9 has D3DRS_CULLMODE rather than a cull face enable.

SEE ALSO

ClipPlaneEnable, DepthTestEnable

NAME

DepthBoundsEnable − 3D API depth bounds enable

USAGE

DepthBoundsEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL depth bounds enable. See the GL_DEPTH_BOUNDS_TEST_EXT description on the OpenGL glEnable manual page for details.

The standard reset callback sets the DepthBoundsEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported.

SEE ALSO

DepthTestEnable, StencilTestEnable

NAME

DepthClampEnable − 3D API depth clamp enable

USAGE

DepthClampEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D depth clamp enable. See the GL_DEPTH_CLAMP_EXT description on the OpenGL glEnable manual page for details.

The standard reset callback sets the DepthClampEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported; ignored.

SEE ALSO

DepthBoundsEnable, DepthTestEnable, StencilTestEnable

NAME

DepthTestEnable − 3D API depth test enable

USAGE

DepthTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D depth test enable. 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

DepthBoundsEnable, DepthClampEnable, DepthTestEnable

NAME

DitherEnable − 3D API color dithering enable

USAGE

DitherEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D dither enable. 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

ColorMask

NAME

FogEnable − 3D API fog enable

USAGE

FogEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D fog enable. 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

BlendEnable, FogMode, FogColor, FogDensity, FogStart, FogEnd

NAME

LightEnable − 3D API per-vertex light source enable

USAGE

LightEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D per-vertex light source enable. See the GL_LIGHTn description on the OpenGL glEnable manual page for details.

The standard reset callback sets the LightEnable state to bool(false).

The index ndx identifies the light source and must be greater or equal to zero and less than the value of GL_MAX_LIGHTS (typically 8). DirectX 9 supports 8 light sources.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightingEnable, DepthTestEnable, StencilTestEnable

NAME

LightModelLocalViewerEnable − 3D API local viewer light model enable

USAGE

LightModelLocalViewerEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D local viewer light model enable. See the GL_LIGHT_MODEL_LOCAL_VIEWER description on the OpenGL glLightModeli manual page for details. See the D3DRS_LOCALVIEWER render state description for DirectX 9.

The standard reset callback sets the LightModelLocalViewerEnable state to bool(false).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightEnable, LightingEnable

NAME

LightModelTwoSideEnable − 3D API two-sided lighting enable

USAGE

LightModelTwoSideEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D two-sided lighting enable. See the GL_LIGHT_MODEL_TWO_SIDE description on the OpenGL glLightModeli 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 supported

SEE ALSO

LightEnable, LightingEnable, LightModelLocalViewer

NAME

LightingEnable − 3D API per-vertex lighting enable

USAGE

LightingEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D per-vertex lighting enable. 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

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LightEnable

NAME

LineSmoothEnable − 3D API line smooth enable

USAGE

LineSmoothEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D line smooth enable. 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

BlendEnable, PolygonSmoothEnable, PointSmoothEnable

NAME

LineStippleEnable − 3D API line stipple enable

USAGE

LineStippleEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL line stipple enable. 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 supported

SEE ALSO

LineStipple, LineSmoothEnable, LineWidth

NAME

LogicOpEnable − 3D API index logical operation enable

USAGE

LogicOpEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL color index logical operation (logic-op) enable. See the GL_INDEX_LOGIC_OP (or GL_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 supported.

SEE ALSO

BlendEnable, ColorLogicOp

NAME

MultisampleEnable − 3D API multisample enable

USAGE

MultisampleEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D multisample enable. 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 MultisampleEnable state to bool(true).

This state is ignored if the color buffer/surface is not multisample capable.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.3 or ARB_multisample

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

LineSmoothEnable, PointSmoothEnable, PolygonSmoothEnable

NAME

NormalizeEnable − 3D API surface normal vector normalization enable

USAGE

NormalizeEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D surface normal vector normalization enable. 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

LightingEnable

NAME

PointSmoothEnable − 3D API smooth point rasterization enable

USAGE

PointSmoothEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL smooth point rasterization enable. 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 supported; consider using PointSpriteEnable as an alternative.

SEE ALSO

PointSpriteEnable, PolygonSmoothEnable, PointSmoothEnable

NAME

PointSpriteEnable − 3D API point sprite enable

USAGE

PointSpriteEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D point sprite enable. See the GL_POINT_SPRITE 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

OpenGL 2.0 or ARB_point_sprite

DIRECT3D FUNCTIONALITY REQUIREMENTS

DirectX 9

SEE ALSO

SmoothPointEnable, PointSize

NAME

PolygonOffsetFillEnable − 3D API polygon offset fill enable

USAGE

PolygonOffsetFillEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL polygon offset fill enable. 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 supported; effectively always enabled.

SEE ALSO

PolygonOffsetPointEnable, PolygonOffsetLineEnable, PolygonMode, PolygonOffset, DepthTestEnable

NAME

PolygonOffsetLineEnable − 3D API polygon offset line enable

USAGE

PolygonOffsetLineEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL polygon offset line enable. See the GL_POLYGON_OFFSET_FILL 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 supported; effectively always enabled.

SEE ALSO

PolygonOffsetPointEnable, PolygonOffsetFillEnable, PolygonMode, PolygonOffset, DepthTestEnable

NAME

PolygonOffsetPointEnable − 3D API polygon offset point enable

USAGE

PolygonOffsetPointEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL polygon offset point enable. See the GL_POLYGON_OFFSET_FILL 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 supported; effectively always enabled.

SEE ALSO

PolygonOffsetLineEnable, PolygonOffsetFillEnable, PolygonMode, PolygonOffset, DepthTestEnable

NAME

PolygonSmoothEnable − 3D API polygon smooth enable

USAGE

PolygonSmoothEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL polygon smooth enable. 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 supported

SEE ALSO

BlendEnable, DepthTestEnable, StencilTestEnable

NAME

PolygonStippleEnable − 3D API polygon stipple enable

USAGE

PolygonStippleEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL polygon stipple enable. 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 supported

SEE ALSO

LineStippleEnable, PolygonMode, ShadeModel

NAME

RescaleNormalEnable − 3D API rescale normal enable

USAGE

RescaleNormalEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL rescale normal enable. 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

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported.

SEE ALSO

NormalizeEnable, LightingEnable

NAME

SampleAlphaToCoverageEnable − 3D API sample alpha to coverage enable

USAGE

SampleAlphaToCoverageEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL sample alpha to coverage enable. See the GL_SAMPLE_ALPHA_TO_COVERAGE 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 ARB_multisample

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

MultisampleEnable, SampleAlphaToCoverageEnable, SampleCoverageEnable

NAME

SampleAlphaToOneEnable − 3D API sample alpha to coverage enable

USAGE

SampleAlphaToOneEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL sample alpha to coverage enable. See the GL_SAMPLE_ALPHA_TO_ONE 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 ARB_multisample

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

MultisampleEnable, SampleAlphaToCoverageEnable, SampleCoverageEnable

NAME

SampleCoverageEnable − 3D API sample alpha to coverage enable

USAGE

SampleCoverageEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL sample alpha to coverage enable. See the GL_SAMPLE_ALPHA_TO_COVERAGE 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 ARB_multisample

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

MultisampleEnable, SampleAlphaToCoverageEnable, SampleAlphaToOneCoverageEnable

NAME

ScissorTestEnable − 3D API scissor test enable

USAGE

ScissorTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D scissor test enable. 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

AlphaTestEnable, DepthTestEnable

NAME

StencilTestEnable − 3D API stencil test enable

USAGE

StencilTestEnable = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL or Direct3D stencil test enable. 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

AlphaTestEnable, BlendEnable, DepthTestEnable

NAME

TexGenQEnable − 3D API texture coordinate generation for Q enable

USAGE

TexGenQEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture coordinate generation for Q enable for the given texture coordinate set. See the GL_TEXTURE_GEN_Q description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenQEnable state to bool(false).

The index ndx identifies the texture coordinate set and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable; D3DTSS_TEXCOORDINDEX texture stage state determines texture coordinate generation without regard to enable state.

SEE ALSO

TexGenSEnable, TexGenTEnable, TexGenREnable

NAME

TexGenREnable − 3D API texture coordinate generation for R enable

USAGE

TexGenREnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture coordinate generation for R enable for the given texture coordinate set index. See the GL_TEXTURE_GEN_R description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenREnable state to bool(false).

The index ndx identifies the texture coordinate set and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable; D3DTSS_TEXCOORDINDEX texture stage state determines texture coordinate generation without regard to enable state.

SEE ALSO

TexGenSEnable, TexGenTEnable, TexGenQEnable

NAME

TexGenSEnable − 3D API texture coordinate generation for S enable

USAGE

TexGenSEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture coordinate generation for S enable for the given texture coordinate set index. See the GL_TEXTURE_GEN_S description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenSEnable state to bool(false).

The index ndx identifies the texture coordinate set and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable; D3DTSS_TEXCOORDINDEX texture stage state determines texture coordinate generation without regard to enable state.

SEE ALSO

TexGenTEnable, TexGenREnable, TexGenQEnable

NAME

TexGenTEnable − 3D API texture coordinate generation for T enable

USAGE

TexGenTEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture coordinate generation for T enable for the given texture coordinate set index. See the GL_TEXTURE_GEN_T description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TexGenTEnable state to bool(false).

The index ndx identifies the texture coordinate set and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not applicable; D3DTSS_TEXCOORDINDEX texture stage state determines texture coordinate generation without regard to enable state.

SEE ALSO

TexGenSEnable, TexGenREnable, TexGenQEnable

NAME

Texture1DEnable − 3D API one-dimensional texture enable

USAGE

Texture1DEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL one-dimensional texture enable for the given texture unit. See the GL_TEXTURE_1D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture1DEnable state to bool(false).

This state should not be used in combination with a FragmentProgram state assignment.

The index ndx identifies the texture unit and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

Texture2DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable

NAME

Texture2DEnable − 3D API two-dimensional texture enable

USAGE

Texture2DEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL two-dimensional texture enable for the given texture unit. See the GL_TEXTURE_2D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture2DEnable state to bool(false).

This state should not be used in combination with a FragmentProgram state assignment.

The index ndx identifies the texture unit and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.0

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

Texture1DEnable, Texture3DEnable, TextureCubeMapEnable, TextureRectangleEnable

NAME

Texture3DEnable − 3D API three-dimensional texture enable

USAGE

Texture3DEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL three-dimensional texture enable for the given texture unit. See the GL_TEXTURE_3D description on the OpenGL glEnable manual page for details.

The standard reset callback sets the Texture3DEnable state to bool(false).

This state should not be used in combination with a FragmentProgram state assignment.

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.2

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

Texture1DEnable, Texture2DEnable, TextureCubeMapEnable, TextureRectangleEnable

NAME

TextureRectangleEnable − 3D API texture cube map enable

USAGE

TextureRectangleEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture cube map enable for the given texture unit. See the GL_TEXTURE_RECTANGLE_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TextureRectangleEnable state to bool(false).

This state should not be used in combination with a FragmentProgram state assignment.

The index ndx identifies the texture unit and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

OpenGL 1.3, ARB_texture_cube map, or EXT_texture_cube map

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureRectangleEnable

NAME

TextureRectangleEnable − 3D API texture rectangle enable

USAGE

TextureRectangleEnable[ndx] = bool(enable)

VALID ENUMERANTS

enable: true, false

DESCRIPTION

Set the OpenGL texture rectangle enable for the given texture unit. See the GL_TEXTURE_RECTANGLE_ARB description on the OpenGL glEnable manual page for details.

The standard reset callback sets the TextureRectangleEnable state to bool(false).

This state should not be used in combination with a FragmentProgram state assignment.

The index ndx identifies the texture unit and must be greater or equal to zero and less than the value of GL_MAX_TEXTURE_UNITS (typically 2 or 4).

OPENGL FUNCTIONALITY REQUIREMENTS

ARB_texture_rectangle, EXT_texture_rectangle, or NV_texture_rectangle

DIRECT3D FUNCTIONALITY REQUIREMENTS

Not supported

SEE ALSO

Texture1DEnable, Texture2DEnable, Texture3DEnable, TextureCubeMapEnable