nvision 08

THE WORLD OF VISUAL COMPUTING

GeForce 8 Features for OpenGL

Mark Kilgard

NVIDIA

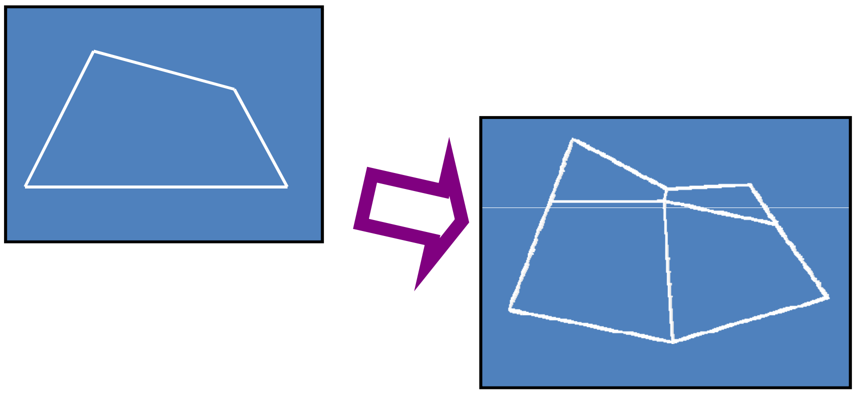
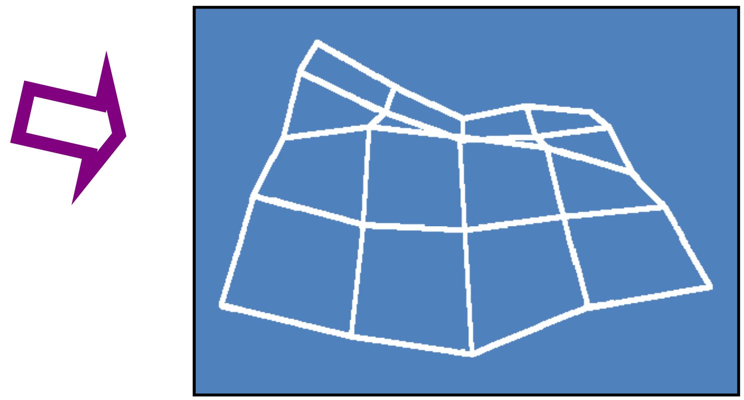
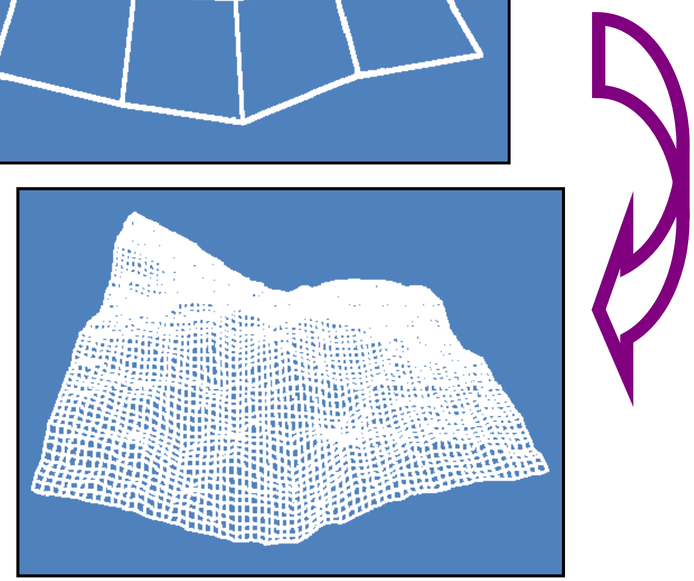
GeForce 8 OpenGL Functionality

- Broad functional categories for GeForce 8 extensions
  - **Vertex**
  - **Programmability**
  - **Texture**
  - **Framebuffer**
- Much of GeForce 8 OpenGL is already standard in OpenGL 3.0 (August 2008)
  - *Don't wait for the functionality to become standardized—because it already is standard!*
- Functional parity with Direct3D 10
  - On any platform: XP, 2000, Vista, Mac OS X, Linux, Solaris, FreeBSD

GeForce 8 OpenGL Vertex Functionality

- Vertex stream output
  - **EXT_transform_feedback** – write stream of transformed vertex attributes into separate or interleaved buffer objects
  - **NV_transform_feedback** – like EXT_transform_feedback but varying outputs for streaming can be designated without re-linking your GLSL shader
- Vertex attribute formats
  - **EXT_gpu_shader4** & **NV_gpu_program4** – signed and unsigned integer vertex attributes
- Vertex instances
  - **EXT_draw_instanced** – send a vertex instance ID for batches of vertices to be accessed by a vertex or geometry program or shader

Transform Feedback for Terrain Generation by Recursive Subdivision

1. Render quads (use 4-vertex line adjacency primitive) from vertex buffer object
2. Fetch height field
3. Stream subdivided positions and normals to transform feedback “other” buffer object
4. Use buffer object as vertex buffer
5. Repeat, ping-pong buffer objects

Computation and data all stays on the GPU!

Skin Deformation by Transform Feedback

Transform feedback allows the GPU to calculate the interactive, deforming elastic skin of the frog

GeForce 8 OpenGL Programmable Functionality

- Low-level assembly
  - **NV_gpu_program4** – extends ARB vertex and fragment program assembly syntax for unified G80 programmability and geometry shaders
    - Incorporates NV_fragment_program4, NV_vertex_program4, and NV_geometry_program4 specifications
    - One extension broken into 4 specification text files
  - **NV_parameter_buffer_object** – read parameters from bind-able buffer objects from low-level assembly
  - Works best with Cg
- High-level OpenGL Shading Language (GLSL)
  - **EXT_gpu_shader4** – additions to GLSL comparable to the NV_gpu_program4 unified G80 programmability functionality
  - **EXT_geometry_shader4** – additions to GLSL comparable to the NV_gpu_program4 geometry shader functionality
  - **NV_geometry_shader4** – dynamic control of maximum output vertices without re-linking & quadrilateral support
  - **EXT_bindable_uniform** – additions to GLSL to read uniform variables from bind-able buffer objects

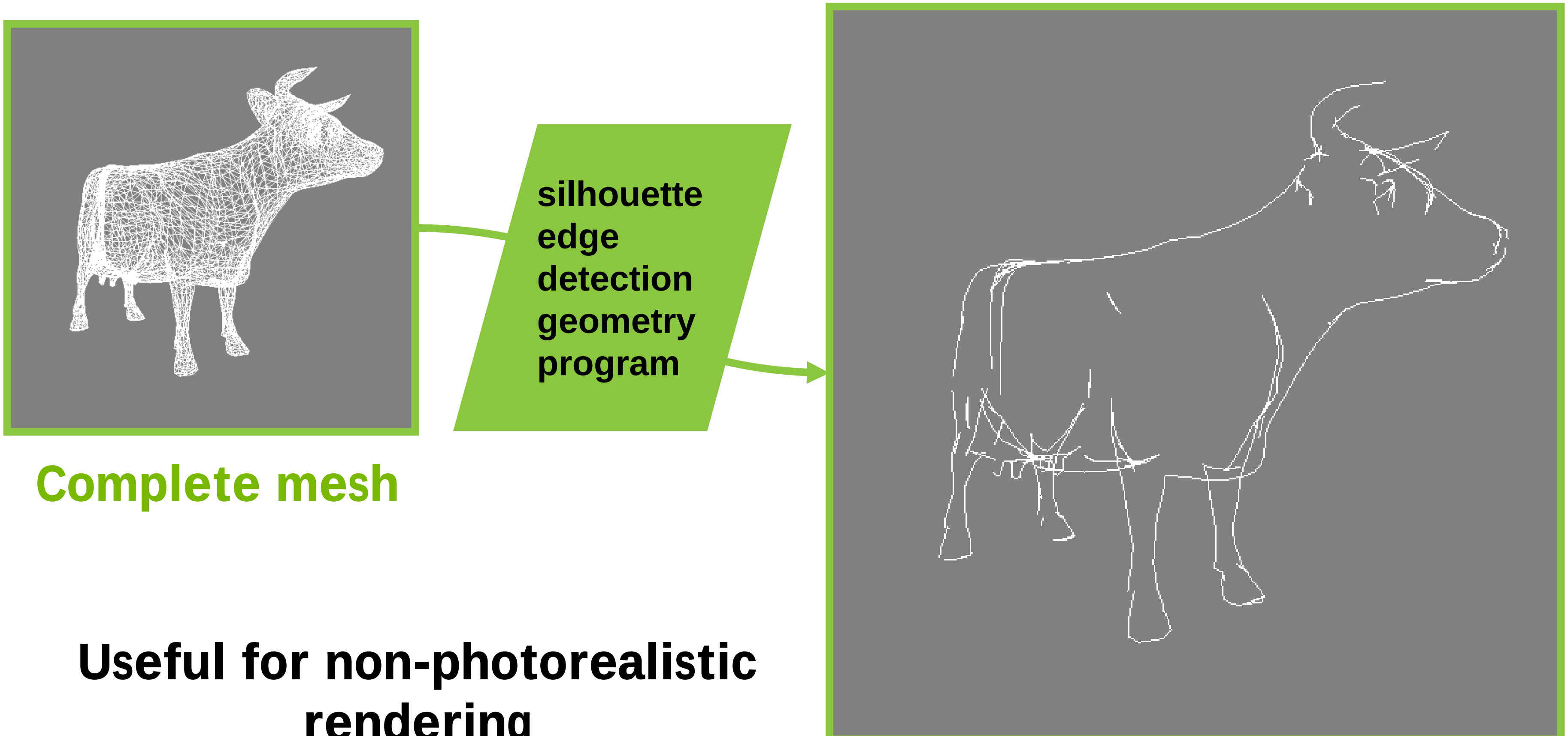
Froggy Demo Surface Shading With New Programmability

Eyes have ray-traced irises and simulated refraction

Skin shader simulates sub-surface scattering

High-detail bump and detail maps

Geometry Shader
Silhouette Edge Rendering



Complete mesh

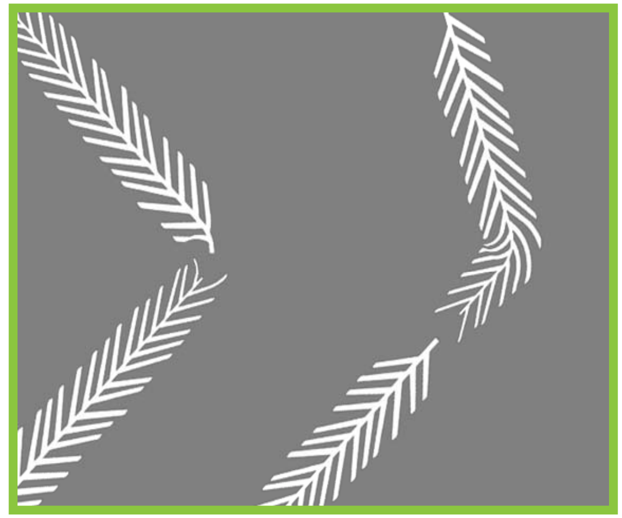
Silhouette edges

Useful for non-photorealistic rendering

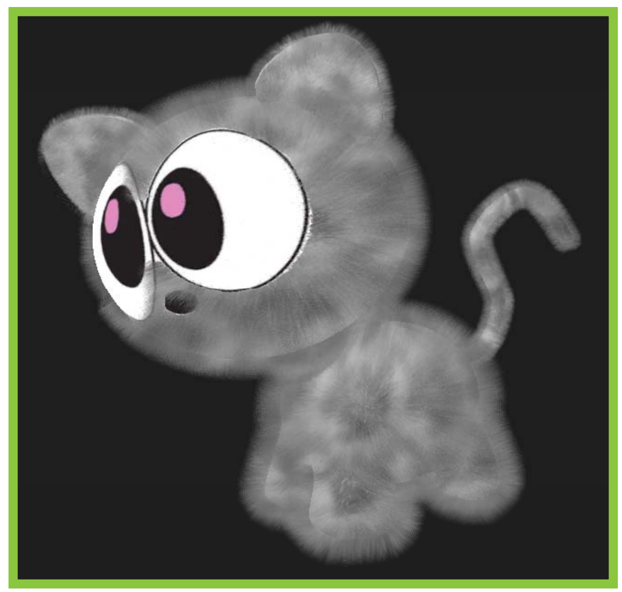
Looks like human sketching

More Geometry Shader Examples

Shimmering point sprites



Generate fins for lines



Generate shells for fur rendering

GeForce 8 OpenGL Texture Functionality (1)

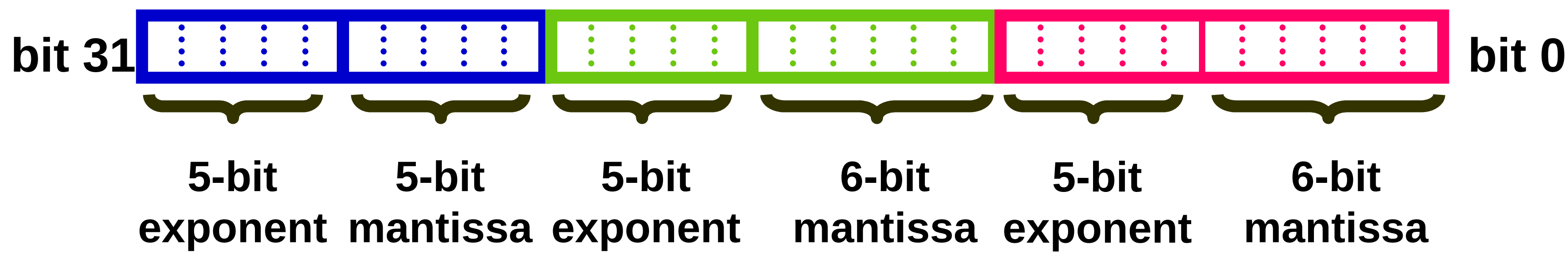
- **New formats**
  - **EXT_texture_integer** - signed & unsigned integer texture formats
  - **EXT_packed_float** – packs 3 unsigned floating-point values with independent 5-bit exponents into a 32-bit texture format
  - **EXT_texture_shared_exponent** – packs 3 unsigned floating-point values with a shared 5-bit exponent into a 32-bit texture format
  - **EXT_texture_compression_latc** & **EXT_texture_compression_rgtc** – one- and two-component texture compression formats based on DXT5's 2:1 alpha component compression scheme for luminance-alpha and red-green data respectively
- **New texture targets**
  - **EXT_texture_array** – indexing into a set of 1D or 2D texture slices
  - **EXT_texture_buffer_object** - unfiltered access to (potentially huge) buffer objects as a 1D texture-formatted array
  - **EXT_gpu_shader4** & **NV_gpu_program4** – shadow cube maps for omni-directional shadow mapping

GeForce 8 OpenGL Texture Functionality (2)

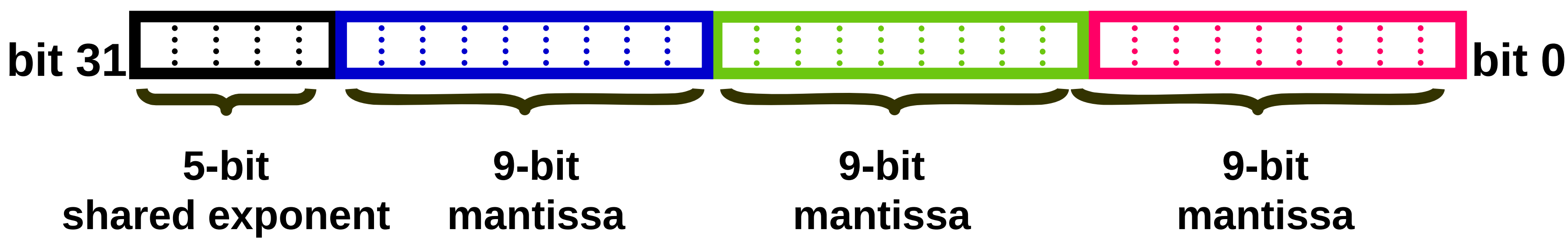
- **New texture access instructions**
  - **EXT_gpu_shader4** & **NV_gpu_program4** – shader-time query for texture size
  - **EXT_gpu_shader4** & **NV_gpu_program4** – integer addressing of texel locations including level of-detail
  - **EXT_gpu_shader4** & **NV_gpu_program4** – texture lookups with an small integer texel offset
- **Texture generality**
  - **EXT_geometry_shader4** & **NV_gpu_program4** – texture fetches from geometry domain
  - No limitations on texture format, sampling modes, etc. in vertex and geometry domains

Compact Floating-point Texture Formats

- **EXT_packed_float**
  - No sign bit, independent exponents



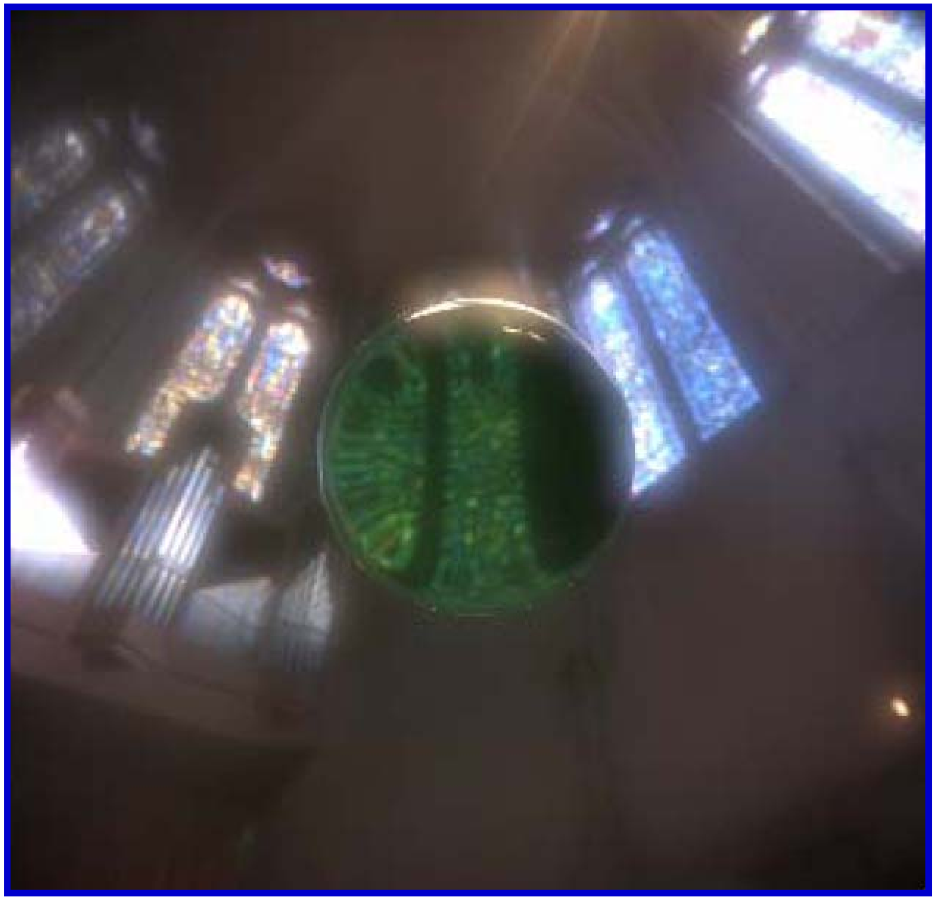
- **EXT_texture_shared_exponent**
  - No sign bit, shared exponent, no implied leading 1



Compact Floating-point Texture Details

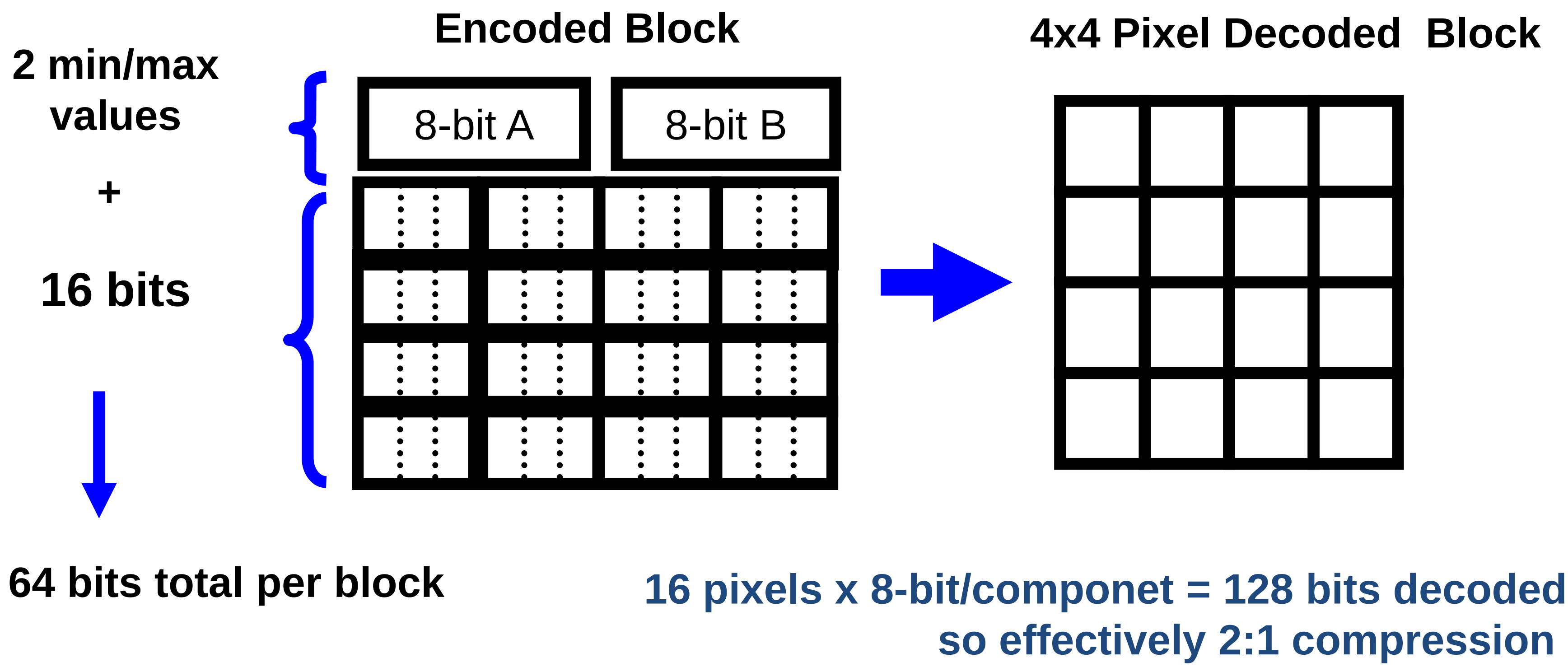
- Intended for High Dynamic Range (HDR) applications
  - Where range matters
  - Magnitudes so signed data unnecessary
  - Texture filtered as half-precision (s10e5) floating-point
- Render-able
  - **EXT_packed_float**: YES, *including* blending
  - **EXT_texture_shared_exponent**: NO
- Easy to use
  - By requesting a compact float texture internal format, OpenGL driver will automatically pack conventional floating-point texture image data

High Dynamic Range Example Using Compact Floating-point Textures



1- and 2-component Block Compression Scheme

- Basic 1-component block compression format



1- and 2-component Block Compression Scheme

- **Matches existing DXT5 compression scheme for alpha component**
  - Translation of 3-bit field to weight depends on A > B ordering
    - A > B: A, B, and 6 values spaced between [A,B]
    - A ≤ B: A, B, 4 values spaced between [A,B], and 0/1 or -1/+1 range
- **Four RGBA component arrangements**
  - Red: [R, 0, 0, 1]
  - Red-Green: [R, G, 0, 1]
  - Luminance: [L, L, L, 1]
  - Luminance-Alpha: [L, L, L, A]

OpenGL 3.0 feature

Multi-vendor EXT extension

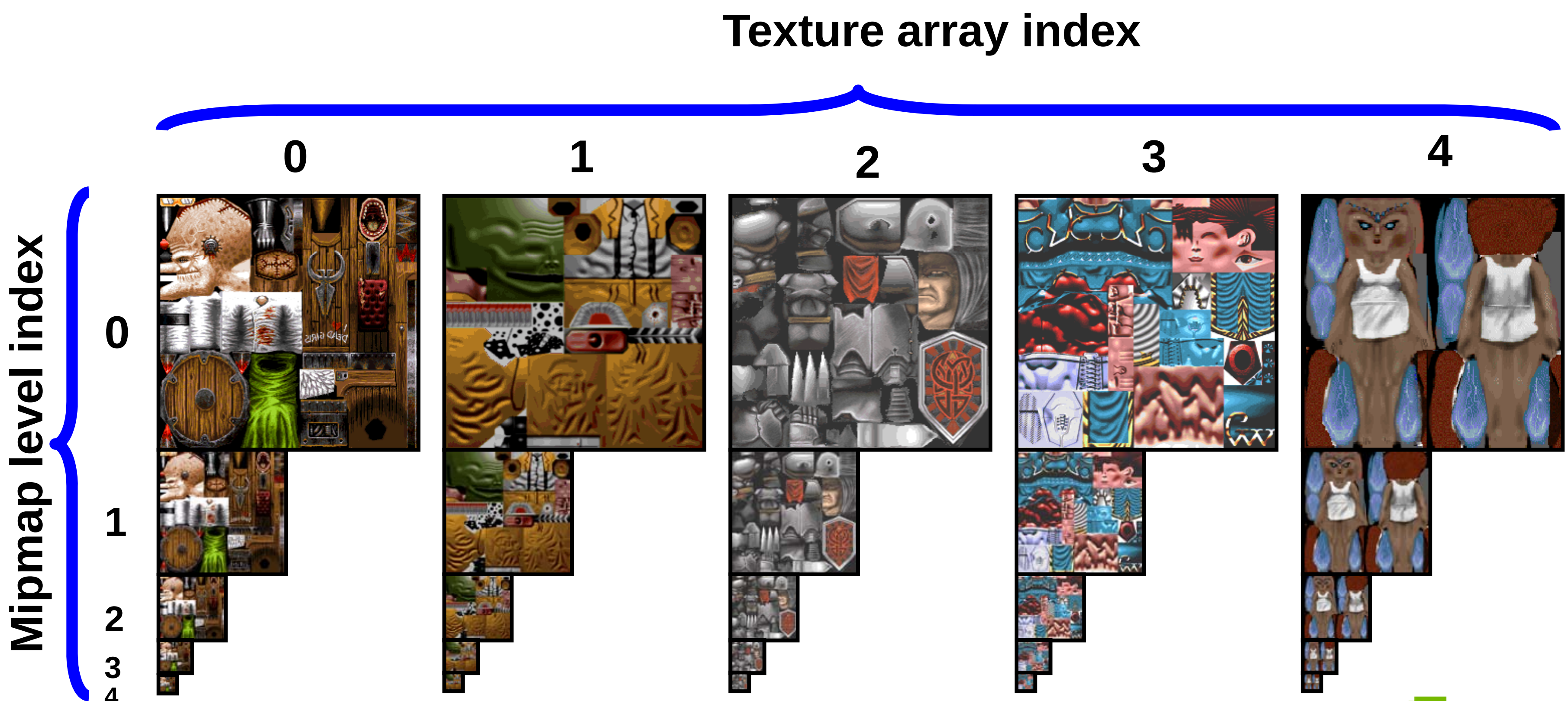
- **2-component formats combine two 64-bit encoded blocks**
  - So 2 components are compressed independently, good for normal maps
- **Signed and unsigned versions of the formats**
  - [0,1] fixed-point unsigned range
  - [-1,+1] fixed-point signed range, appropriate for normal maps
- **Based on requested texture internal format, OpenGL driver can automatically compress uncompressed 1- and 2-component formats into these formats**

Texture Arrays

- **Conventional texture**
  - One logical pre-filtered image
- **Texture array**
  - An array of mipmap sets, a <u>plurality</u> of pre-filtered images
  - <u>No</u> filtering between mipmap sets in a texture array
  - All mipmap sets in array share same format/border & base dimensions
  - Both 1D and 2D texture arrays
  - Require shaders, no fixed-function support
- **Texture image specification**
  - Use glTexImage3D, glTexSubImage3D, etc. to load 2D texture arrays
    - <u>No new</u> OpenGL commands for texture arrays
  - 3rd dimension specifies integer array index
    - No halving in 3rd dimension for mipmaps
    - So $64 \times 128 \times 17$ reduces to $32 \times 64 \times 17$ all the way to $1 \times 1 \times 17$

Texture Arrays Example

- Multiple skins packed in texture array
  - Motivation: binding to one multi-skin texture array avoids texture bind per object



GeForce 8 OpenGL Framebuffer Functionality

- **Framebuffer formats**
  - **EXT_framebuffer_sRGB** – color values assumed to be in a linear color space are converted to sRGB color space when writing and blending to sRGB-capable framebuffer
  - **EXT_texture_integer** – rendering to integer texture formats through Framebuffer Buffer Object (FBO) render-to-texture functionality
  - **NV_depth_buffer_float** – depth values are stored in the depth buffer 32-bit floating-point values, with or without [0,1] clamping
- **Multiple render targets**
  - **EXT_draw_buffers2** – per-color buffer blending and color masking
  - **EXT_geometry_shader4** & **NV_gpu_program4** – render to texture array and select output slice in shader
- **Multisample support**
  - **EXT_framebuffer_multisample_coverage** – render-to-texture control for Coverage Sample Anti-Aliasing (CSAA)

Delicate Color Fidelity Using sRGB

NVIDIA's Adriana GeForce 8 Launch Demo

Unnaturally deep facial shadows

Conventional rendering with uncorrected color

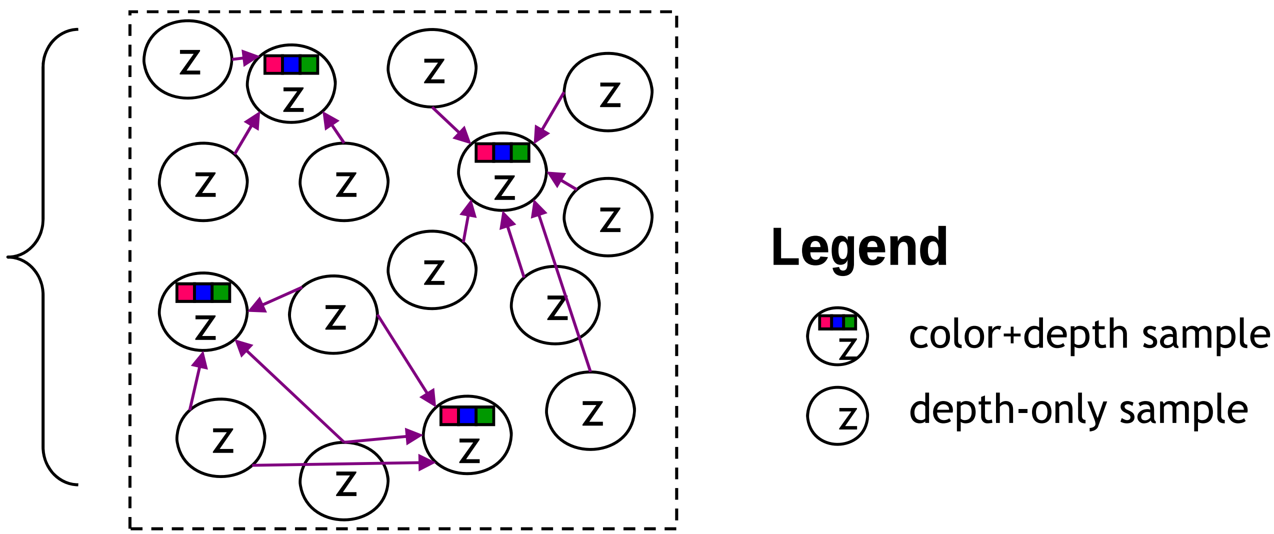
Gamma correct (sRGB rendered)

Softer and more natural

Coverage Sample Anti-Aliasing

- **Established multisampling approach**
  - Shade per-pixel, rasterize & update color and depth info per-sample
  - Multi (4x, 8x, etc.) samples per pixel
- **Coverage Sample Anti-Aliasing (CSAA) approach**
  - Maintain N color+depth samples and M additional depth-only samples that “share” the identical color of one or more color+depth samples
  - Typical configuration: 4 depth+color samples & 12 depth-only samples
  - Improves coverage anti-aliasing while minimizing bandwidth

4 color+depth & 12 depth-only samples within a single pixel



Modernizing the OpenGL API

- **OpenGL 1.0 (1992) is over 16 years old**
- **Modernization motivations**
  - Some aspects of the API have not scaled well
    - Must “bind for rendering” to a texture or program object just to modify it
  - Increase similarity to Direct3D
    - NVIDIA recognizes that developers must support 3D applications cross multiple APIs and platforms
- **Compatibility matters**
  - OpenGL is a big ecosystem of libraries, documentation, textbooks, tools, applications, and developer expertise

Beyond OpenGL 3.0

- **What's in OpenGL 3.0 and what's still not…**

OpenGL 3.0

- EXT_gpu_shader4
- NV_conditional_render
- ARB_color_buffer_float
- NV_depth_buffer_float
- ARB_texture_float
- EXT_packed_float
- EXT_texture_shared_exponent
- NV_half_float
- ARB_half_float_pixel
- EXT_framebuffer_object
- EXT_framebuffer_multisample
- EXT_framebuffer_blit
- EXT_texture_integer
- EXT_texture_array
- EXT_packed_depth_stencil
- EXT_draw_buffers2
- EXT_texture_compression_rgtc
- EXT_transform_feedback
- APPLE_vertex_array_object
- EXT_framebuffer_sRGB
- APPLE_flush_buffer_range (modified)

In GeForce 8

but not yet core

- EXT_geometry_shader4 (now ARB)
- EXT_bindable_uniform
- NV_gpu_program4
- NV_parameter_buffer_object
- EXT_texture_compression_latc
- EXT_texture_buffer_object (now ARB)
- NV_framebuffer_multisample_coverage
- NV_transform_feedback2
- NV_explicit_multisample
- NV_multisample_coverage
- EXT_draw_instanced (now ARB)
- EXT_direct_state_access

Make your desires for standardization of functionality clear

OpenGL Direct State Access

- Traditional OpenGL state access model
  - "Bind object to edit" model
    - Bind to a texture, program, etc. and then update its state
  - Prior state controls "which state" a second command will update
    - Generally a bad thing
    - **Example selectors:** active texture, current texture binding, matrix mode, current program binding, etc.
- New comprehensive OpenGL extension
  - Called EXT_direct_state_access
  - "Edit object by name" model
  - More like Direct3D API
  - Easier for layered libraries to update OpenGL state without disturbing selectors and bindings

EXT_direct_state_access Extension

- Collaboration with multiple OpenGL software and hardware vendors
  - NVIDIA, S3, TransGaming, Aspyr, Blizzard, Id Software
- Adds “direct state access” to OpenGL API
  - Purely an API feature—not a hardware feature
- Example: uniformly scaling model-view matrix by 2
  - Old

```
GLenum savedMatrixMode;
glGetIntegerv(GL_MATRIX_MODE, &savedMatrixMode);
glMatrixMode(GL MODELVIEW);
glScaleMatrixf(2,2,2);
glMatrixMode(savedMatrixMode);
```

  - New

```
glMatrixScalefEXT(GL_MODELVIEW, 2,2,2);
```

More EXT_direct_state_access Examples

- Binding textures to texture units
  - Old
    ```
    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D, texobj);
    ```
  - New
    ```
    glBindMultiTexture(GL_TEXTURE5,GL_TEXTURE_2D,texobj);
    ```
- Updating a uniform or program parameter
  - Old
    ```
    glBindProgramARB(GL_VERTEX_PROGRAM, vp);
    glProgramLocalParameter4fARB(index, x,y,z,w);
    glUseProgram(glslprog);
    glUniform4f(location, x,y,z,w);
    ```
  - New
    ```
    glNamedProgramLocalParameter4fEXT(vp,index, x,y,z,w);
    glProgramUniform4fEXT(glslprog, location, x,y,z,w);
    ```

EXT_direct_state_access Extras

- Selector-free (direct) access to
  - Matrices
  - Texture units
  - Texture objects
  - Program objects (assembly & GLSL)
  - Buffer objects
  - Framebuffer objects
- Fast “safe” (side-effect free) client state updates
  - **`glClientAttribDefaultEXT`**
  - **`glPushClientAttribDefaultEXT`**

EXT_direct_state_access Availability

- Shows up in up-coming Release 180.xx drivers
- Cg 2.1 uses EXT_direct_state_access for improved performance

NVIDIA Mac OS X OpenGL Functionality

- Mac OS X 10.5 (Leopard)
  - Support for GeForce 8
    - GeForce 8600M in all MacBook Pros
    - GeForce 8800 GS in 24-inch iMac
    - GeForce 8800 GT option for Mac Pro
  - GeForce 8 supports key DirectX 10-class OpenGL extensions
    - **EXT_gpu_shader4**
    - **EXT_geometry_shader4**
    - **EXT_bindable_uniform**
    - **EXT_transform_feedback**
- More coming!

Summary

- NVIDIA delivers world’s most functional OpenGL implementation
  - Everything in DirectX 10 is exposed by NVIDIA’s OpenGL
- NVIDIA is working hard to standardize latest GPU functionality for OpenGL
  - Witness OpenGL 3.0
    - NVIDIA delivered OpenGL 3.0 beta driver same week as OpenGL 3.0 was announced
  - Working to improve OpenGL API
    - Witness **EXT_direct_state_access**

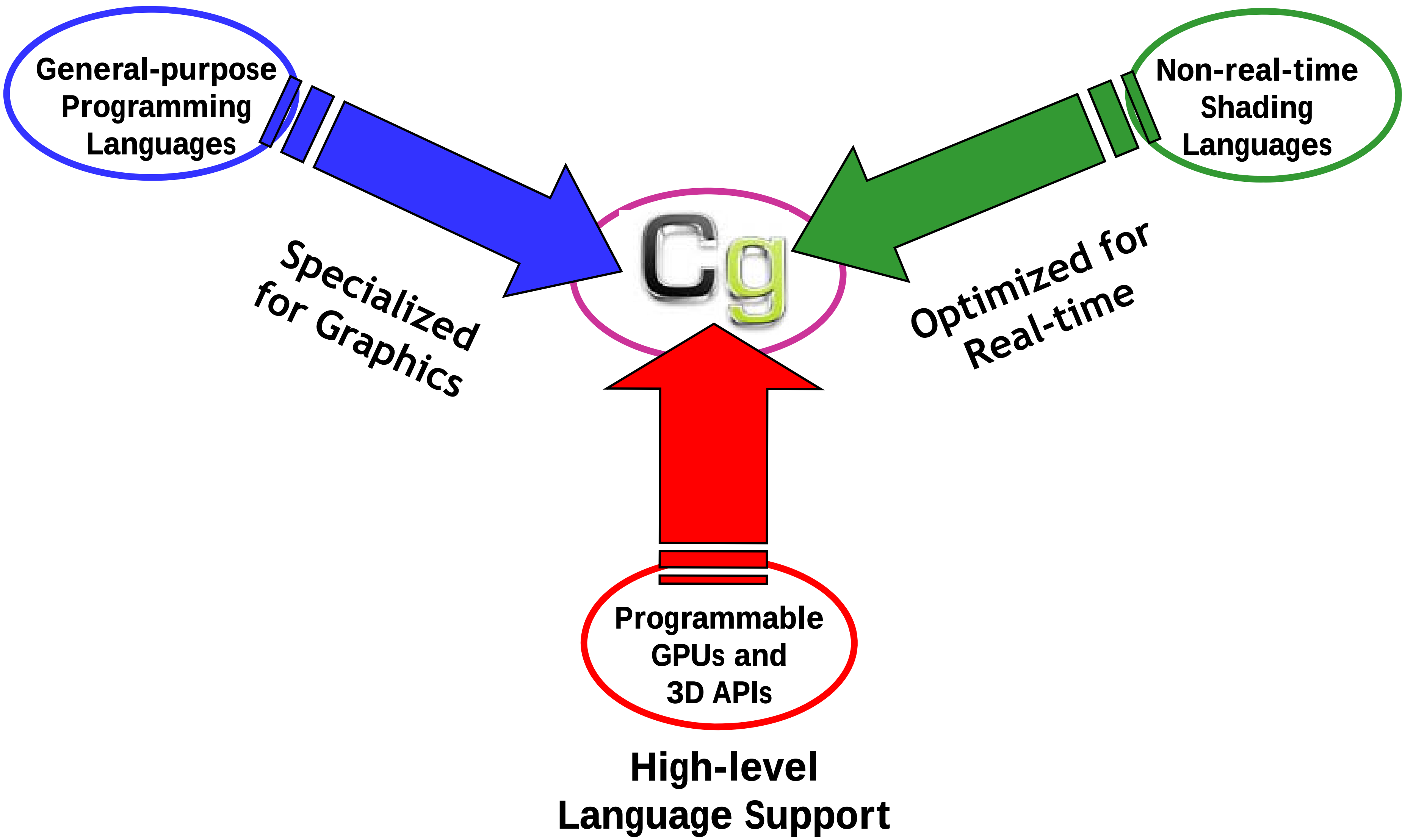
NVISION 08
THE WORLD OF VISUAL COMPUTING

Cg 2.1 – "C for Graphics" for all 3D APIs and Platforms

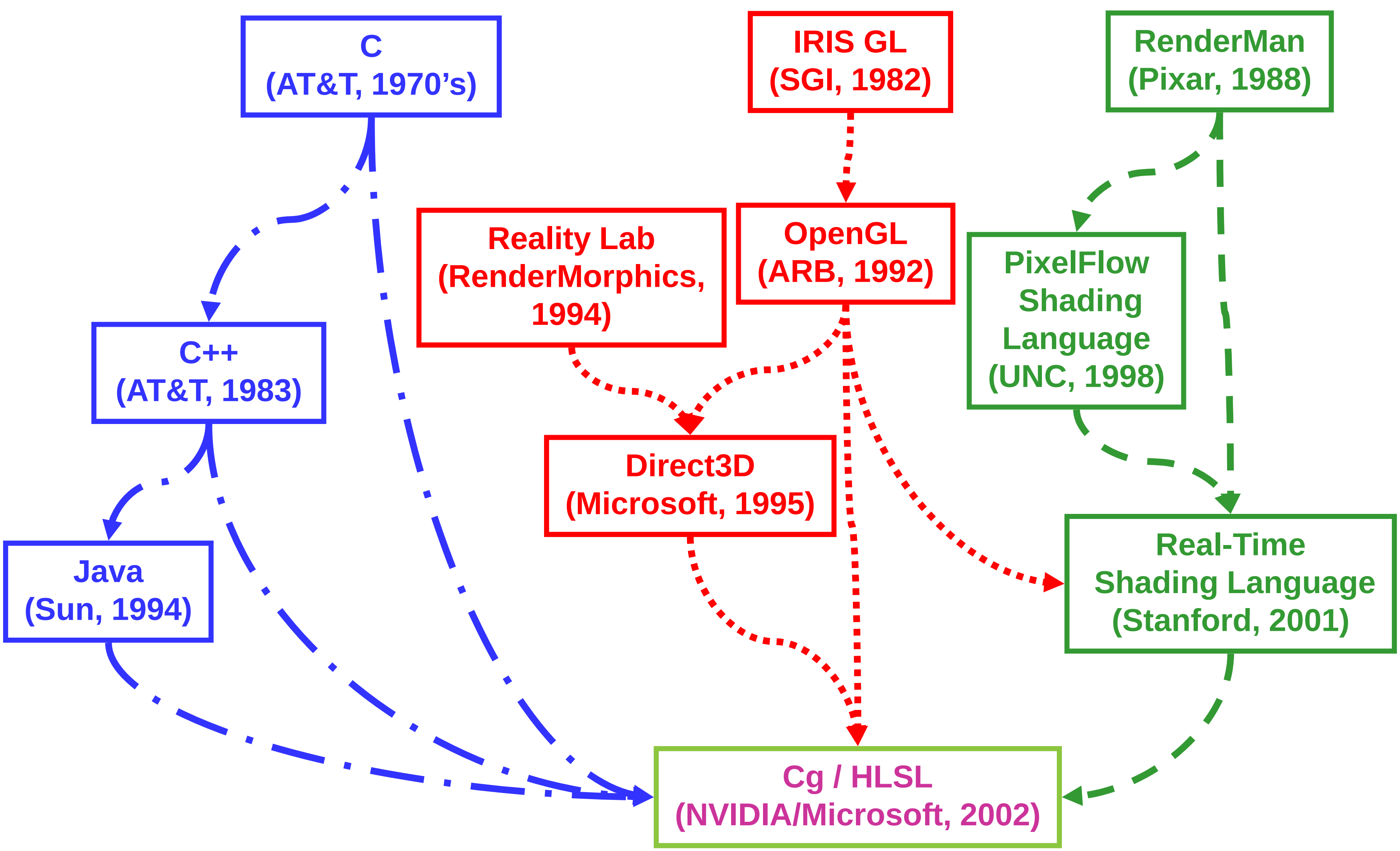
Mark Kilgard

NVIDIA

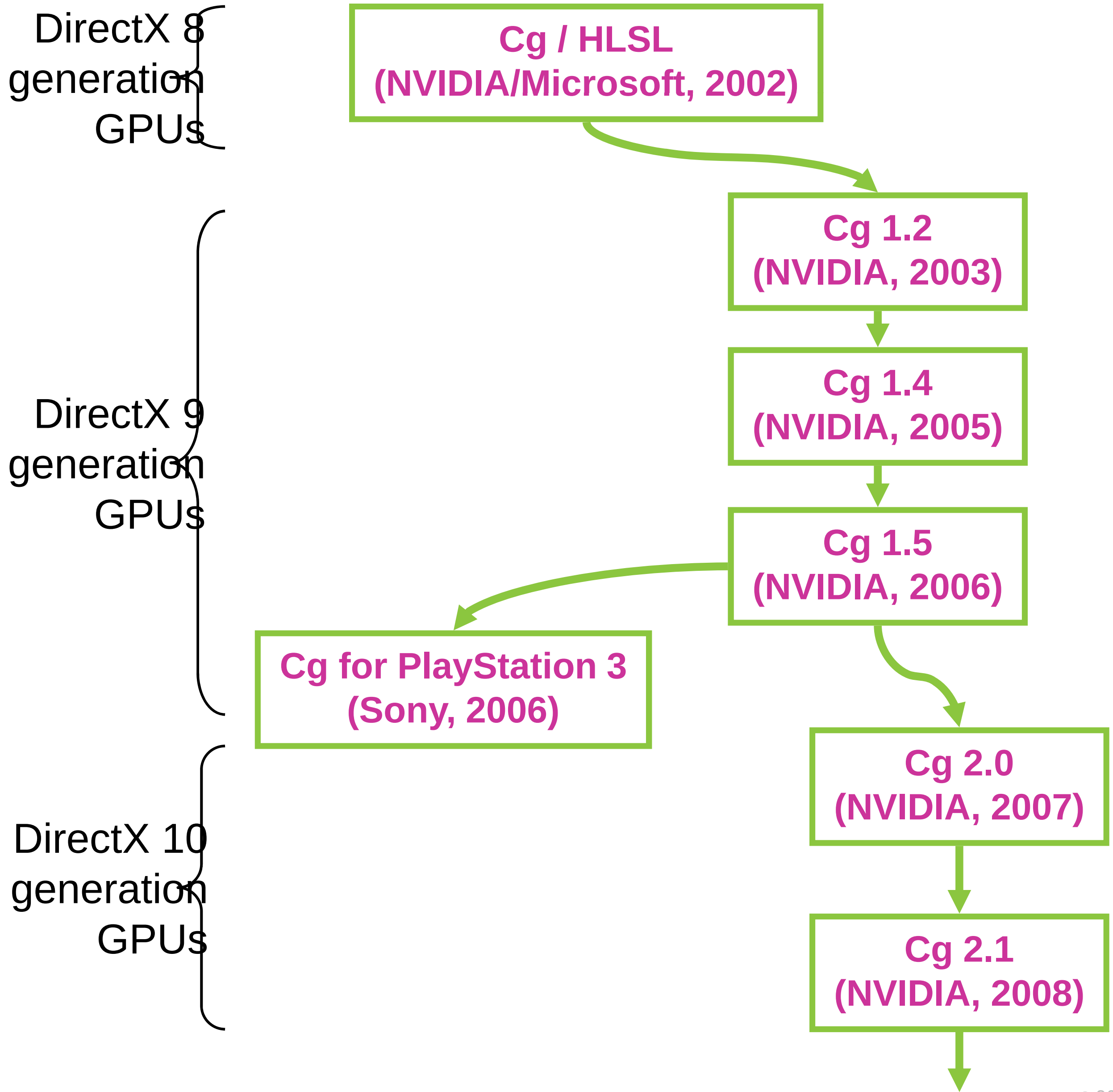
Technology Pressures Leading to Real-time Shading Languages



The Evolution to Cg



Cg's Continuing Evolution



Core language

Expressiveness:
interfaces &
un-sized arrays

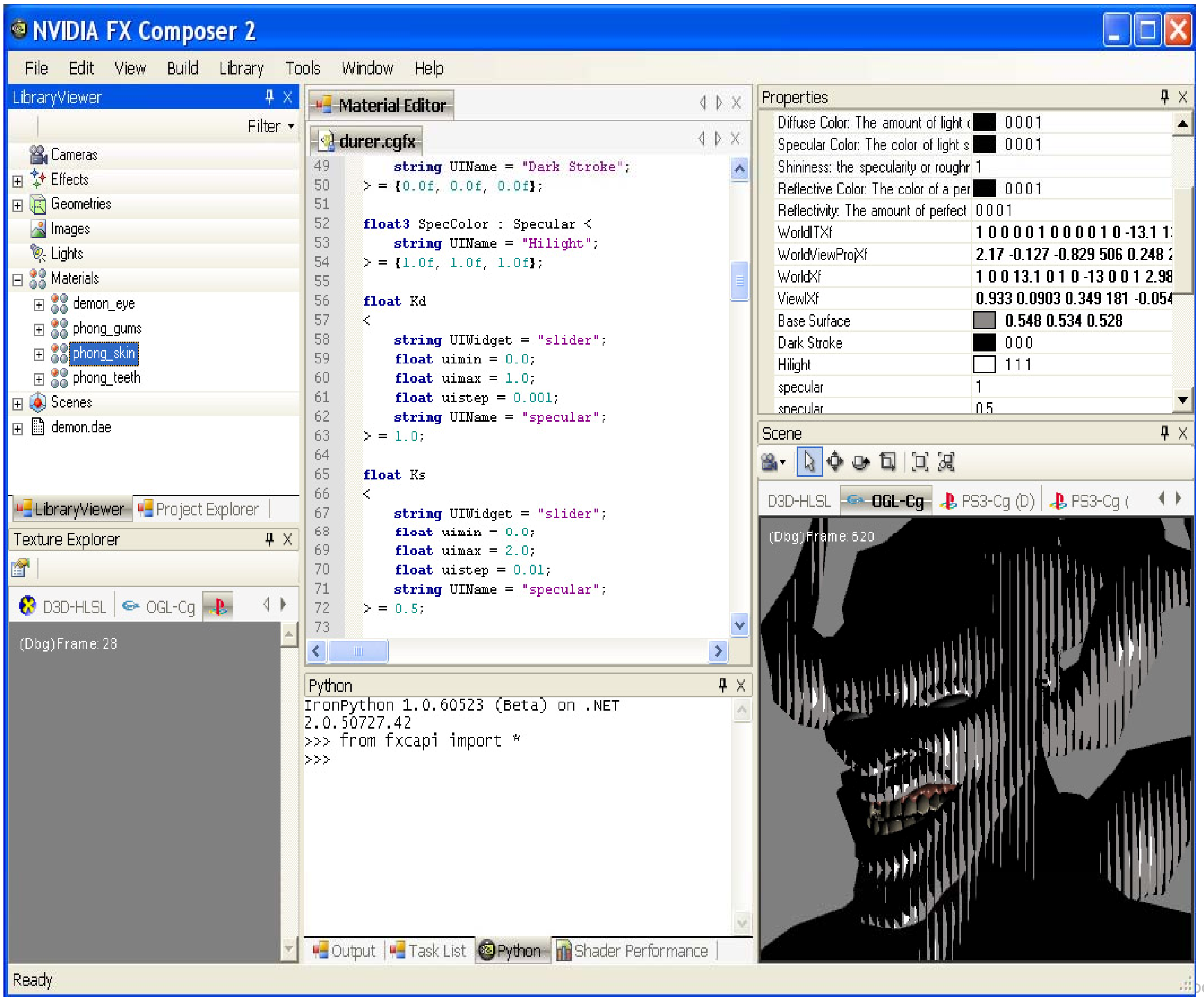
Authoring:
CgFX

Multi-platform:
GLSL & HLSL9
cross-compilation

GeForce 8:
geometry shaders,
buffers, & more

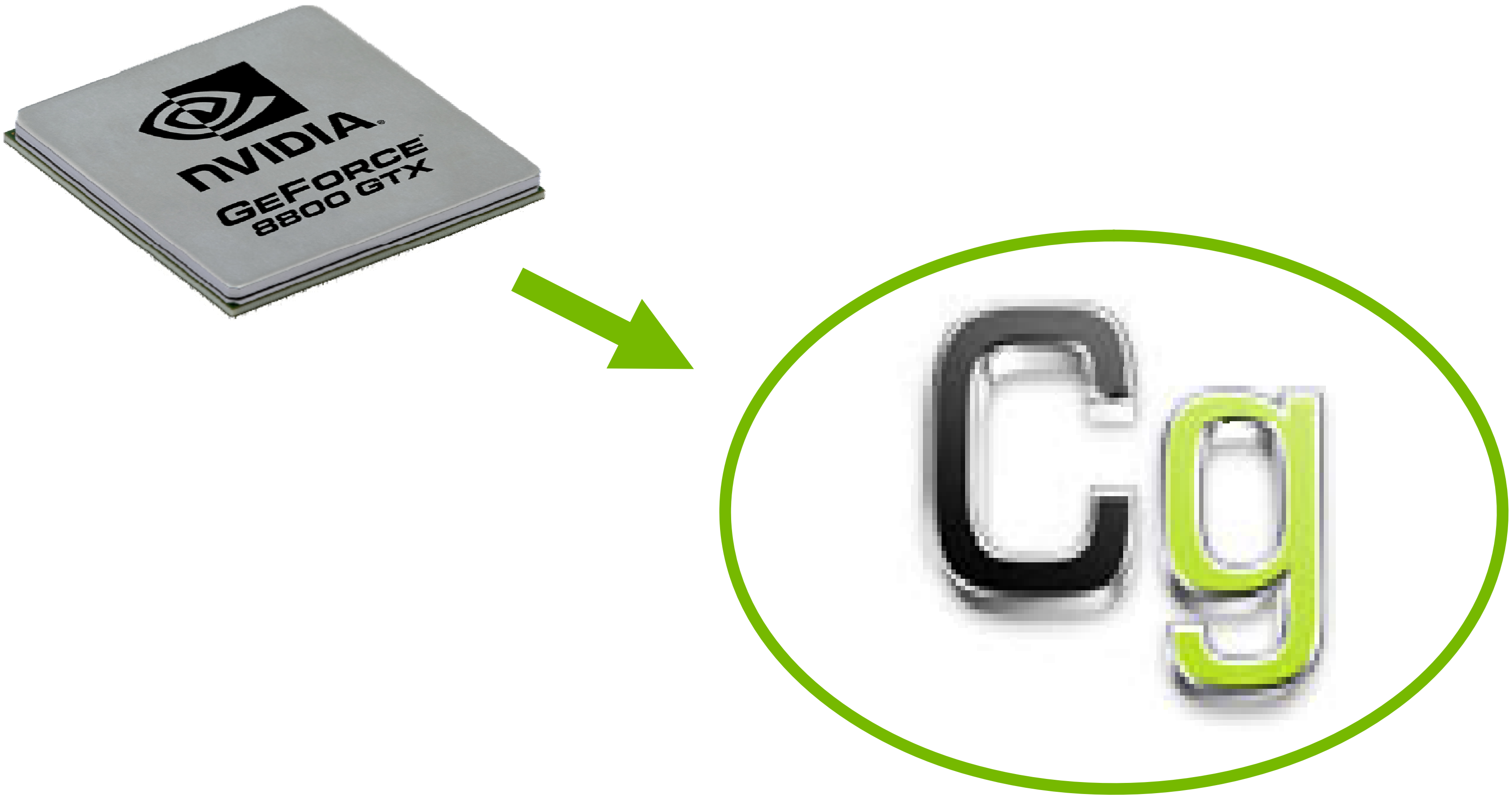
Multi-platform:
HLSL10
cross-compilation

Use FX Composer 2 for Interactive Cg Shader Authoring



Free download— just like Cg Toolkit

Cg 2.0—Supporting GeForce 8



Cg 2.0 Features

- Programmable per-primitive processing
  - Geometry shaders
- Uniform parameters read from bind-able buffers
  - “constant buffers” (HLSL) or “parameter buffers” (EXT_parameter_buffer_object OpenGL assembly) or “bind-able uniform” (GLSL)
- Compilation to other high-level languages
  - Cg to GLSL and Direct3D 9 HLSL
- Meta-shading view CgFX effects
  - State assignments for geometry shader
  - Better Microsoft FX compatibility
  - Interfaces and unsigned arrays
- Bug and performance fixes, compiler updates for GeForce 8

GeForce 8 Cg 2.0 Details

- New GeForce 8 profiles for Shader Model 4.0
  - gp4vp: **NV_gpu_program4** vertex program
  - gp4gp: **NV_gpu_program4** geometry program
  - gp4fp: **NV_gpu_program4** fragment program
- New Cg language support
  - int variables really are integers now
  - Temporaries dynamically index-able now
  - All G80 texturing operations exposed
    - New samplers, new standard library functions
  - New semantics
    - Instance ID, vertex ID, bind-able buffers, viewport ID, layer
  - Geometry shader support
    - Attrib arrays, emitVertex & restartStrip library routines
    - Profile modifiers for primitive input and output type

Geometry Shader Pass Through Example

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

nVISION 08 THE WORLD OF VISUAL COMPUTING

NVIDIA.

Geometry Shader Pass Through Example

Length of attribute arrays depends on the input primitive mode, 3 for TRIANGLE

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

Geometry Shader Pass Through Example

Semantic ties uniform parameter to a buffer, compiler assigns offset

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

Geometry Shader Pass Through Example

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

Bundles a vertex based on parameter values and semantics

Geometry Shader Pass Through Example

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

Makes sure flat attributes are associated with the proper provoking vertex convention

Geometry Shader Pass Through Example

Length of attribute arrays depends on the input primitive mode, 3 for TRIANGLE

Semantic ties uniform parameter to a buffer, compiler assigns offset

```
uniform float4 flatColor : BUFFER[0] ;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i], float3(1,0,0):TEXCOORD1);
  }
}
```

Makes sure flat attributes are associated with the proper provoking vertex convention

Bundles a vertex based on parameter values and semantics

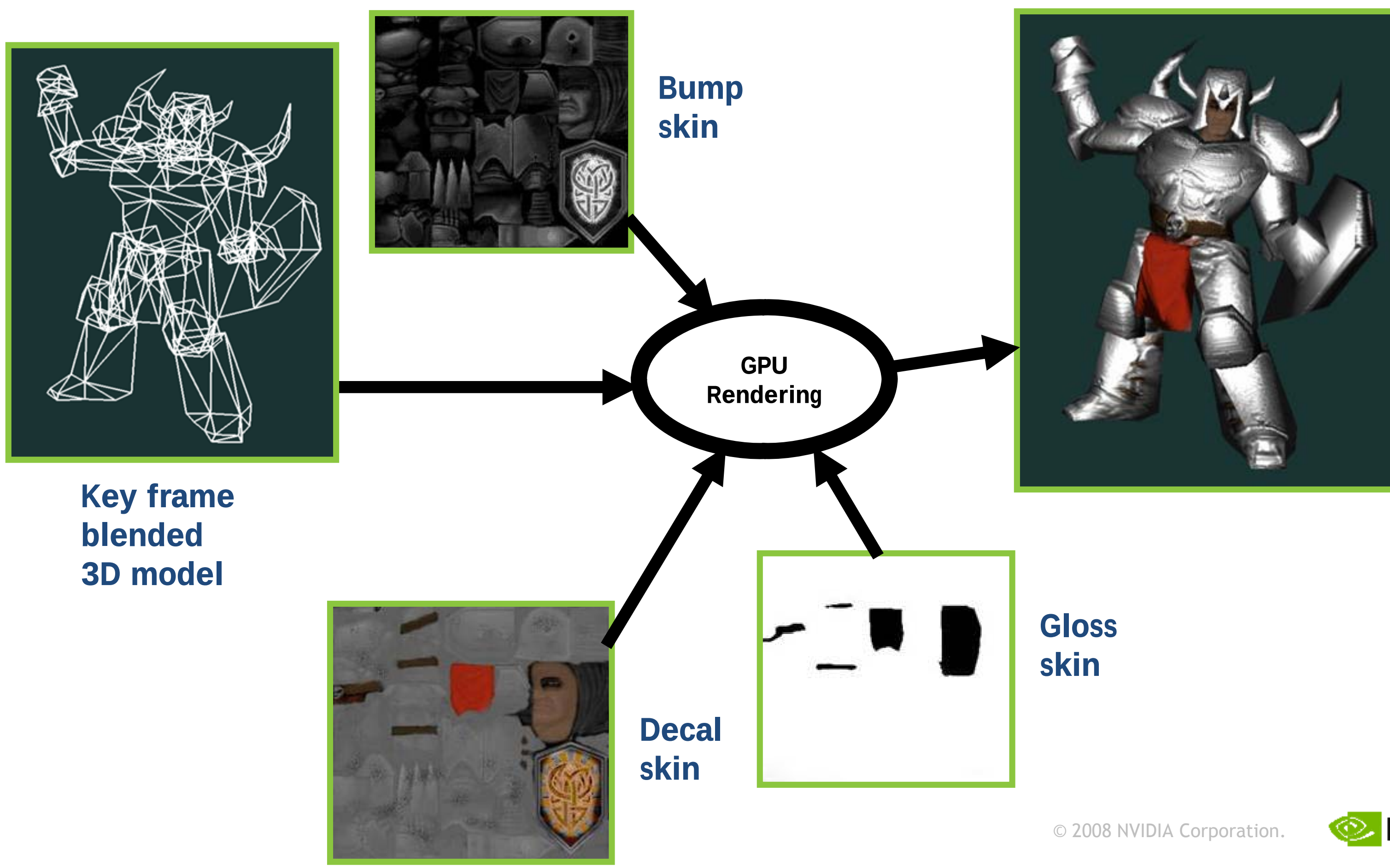
New GeForce 8 Cg 2.0 Features

```
void LINE hermiteCurve(AttribArray<float4> position : POSITION,
                       AttribArray<float4> tangent  : TEXCOORD0,

               uniform float steps)  // # line segments to approx. curve
{
  emitVertex(position[0]);
  for (int t=1; t<steps; t++) {
    float s        = t / steps;
    float ssquared = s*s;
    float scubed   = s*s*s;

    float h1 =  2*scubed - 3*ssquared  + 1;  // calculate basis function 1
    float h2 = -2*scubed + 3*ssquared;       // calculate basis function 2
    float h3 =    scubed - 2*ssquared + s;   // calculate basis function 3
    float h4 =    scubed -   ssquared;       // calculate basis function 4

    float4 p : POSITION = h1*position[0] +   // multiply and sum all functions
                          h2*position[1] +   // together to build interpolated
                          h3*tangent[0]  +   // point along the curve
                          h4*tangent[1];
    emitVertex(p);
  }
  emitVertex(position[1]);
}
```

(Geometry shaders not really ideal for tessellation.)

Bump Mapping Skinned Characters

- **Pre-geometry shader approach:** CPU computes texture-space basis per skinned triangle to transform lighting vectors properly
  - **Problem:** Meant skinning was done on the CPU, not GPU

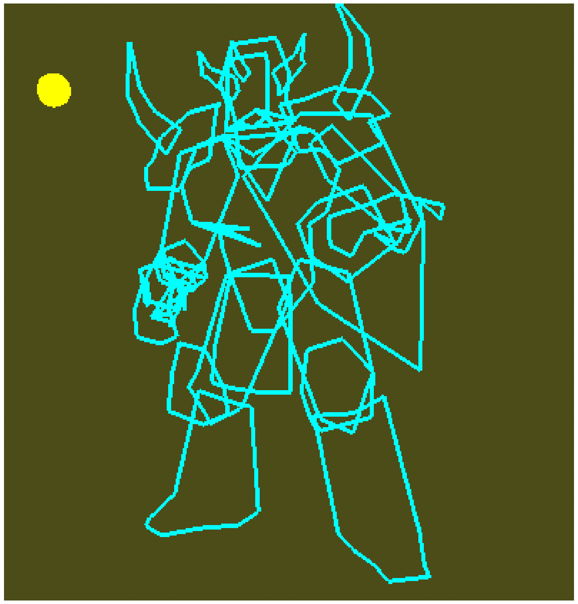


Bump Mapping Skinned Characters With Cg 2.0 Geometry Shader

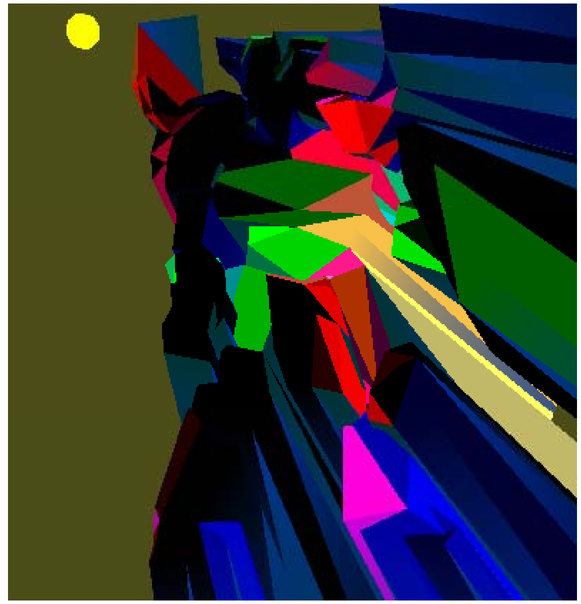
- Cg vertex shader does skinning
- Cg geometry shader computes transform from object- to texture-space based on each triangle
- Cg geometry shader then transforms skinned object-space vectors (light and view) to texture space
- Cg fragment shader computes bump mapping using texture-space normal map
- *Computations all stay on the GPU*

Cg Toolkit 2.x includes full source code example

Next, Add Geometry-Shader Generated Shadows with Stenciled Shadow Volumes



Cg geometry shader computes possible silhouette edges from triangle adjacency
(visualization)



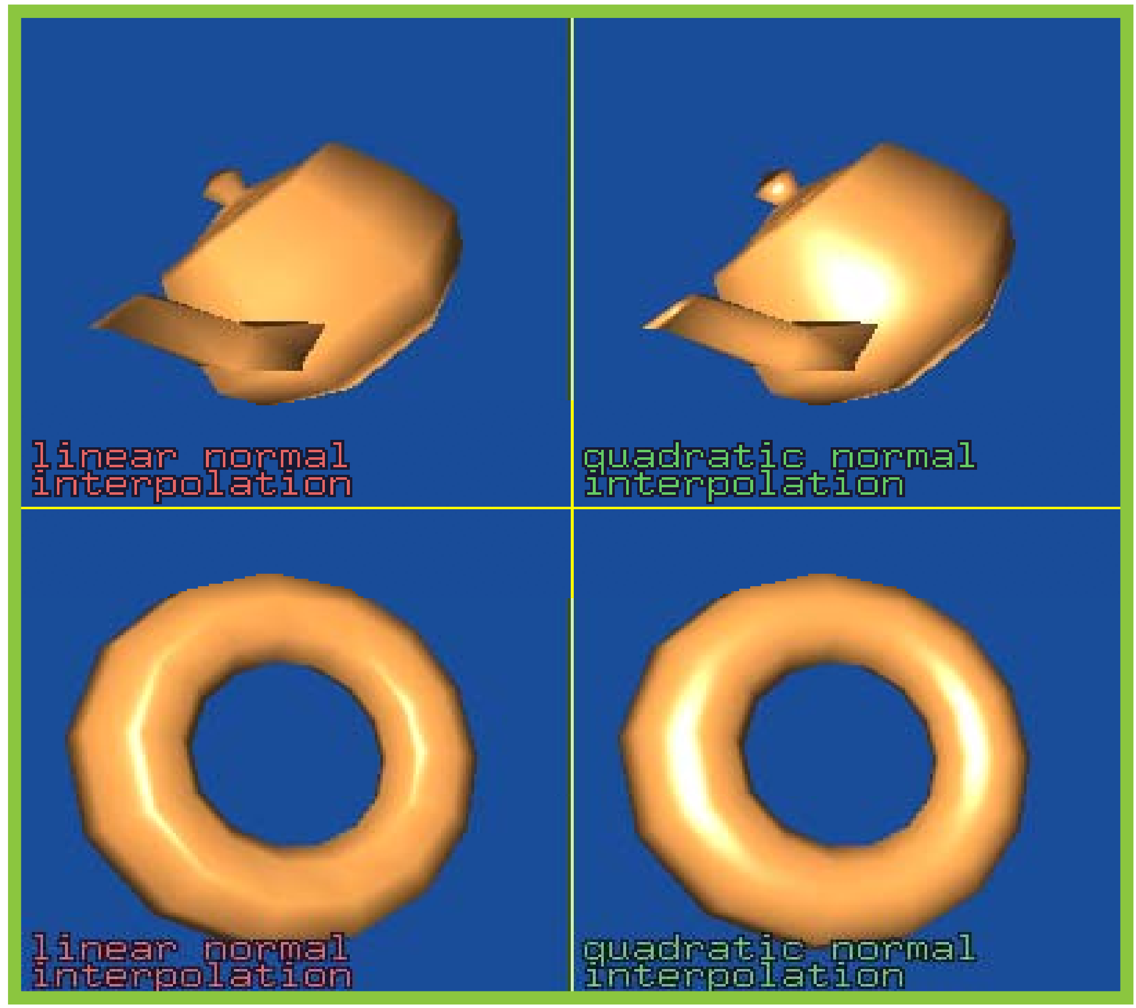
Extrude shadow volumes based on triangle facing-ness and silhouette edges
(visualization)

Add bump mapped lighting based on stenciled shadow volume rendering
(complete effect)

Again—Cg Toolkit 2.x includes full source code example

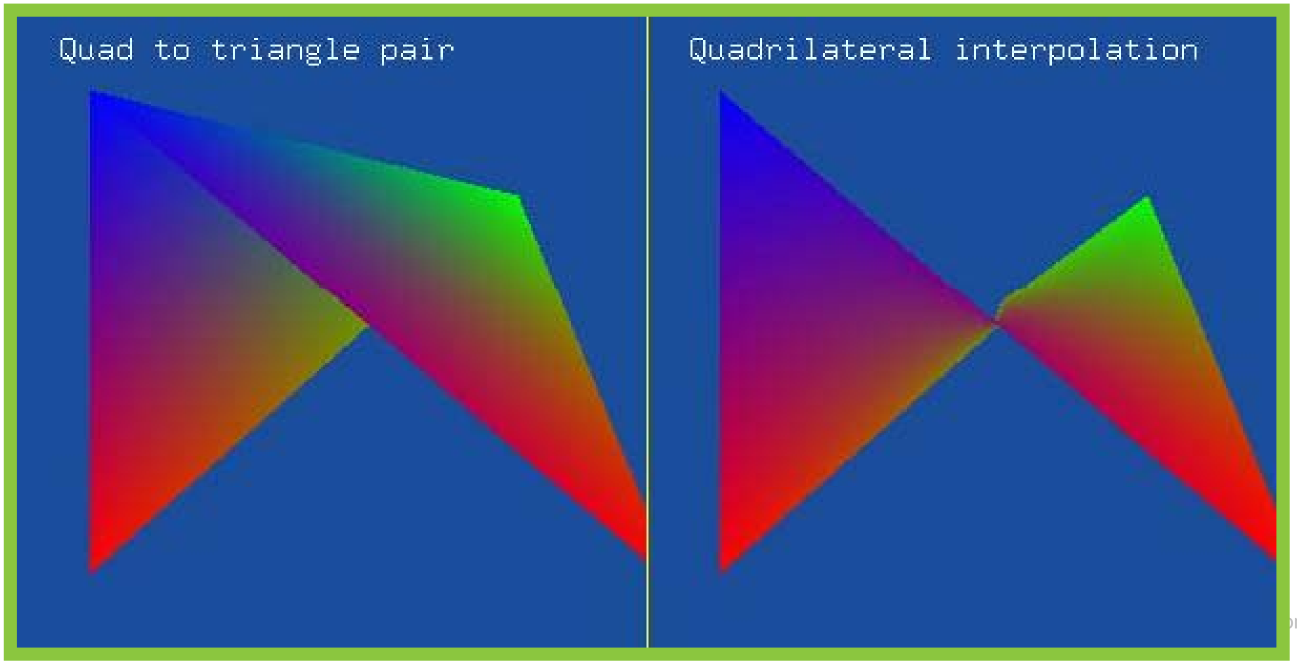
Geometry Shader Setup for Quadratic Normal Interpolation

- Linear interpolation of surface normals don't match real surfaces (except for flat surfaces)
- *Quadratic normal interpolation* [van Overveld & Wyvill]
  - Better Phong lighting, even at low tessellation
- Approach
  - Geometry shader sets up linear parameters
  - Fragment shader combines them for quadratic result
- Best exploits GPU's linear interpolation resources



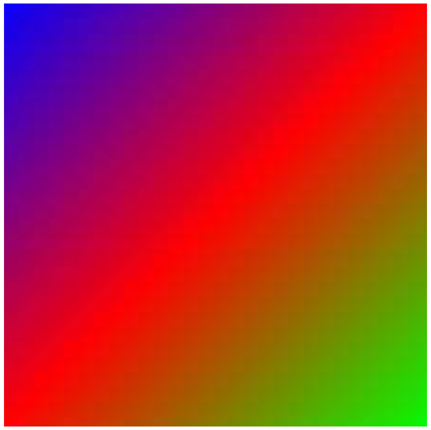
True Quadrilateral Rasterization and Interpolation (1)

- The world is not all triangles
  - Quads exist in real-world meshes
- Fully continuous interpolation over quads is not linear
  - Mean value coordinate interpolation [Floater, Hormann & Tarini]
- Quads can "bow tie"

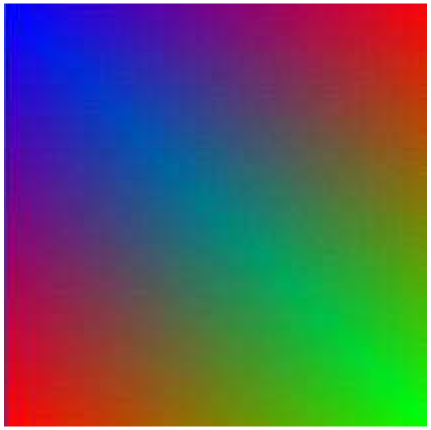


True Quadrilateral Rasterization and Interpolation (2)

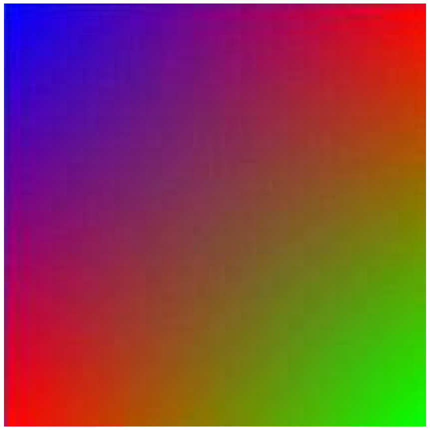
- Conventional hardware: How you split quad to triangles can greatly alter interpolation
  - Both ways to split introduce interpolation discontinuities



"Slash" split



"Backslash" split



Mean value coordinate interpolation via Cg geometry and fragment shaders

Cg 2.1 Updates

- DirectX 10 support
  - New translation profiles for DirectX 10 Shader Model 4.0
    - **vs_4_0** and **ps_4_0** profiles
  - New cgD3D10.dll for Cg runtime Direct3D 10 support
    - With examples
  - Now you can cross-compile Cg to <u>all</u> standard GPU shading languages & assembly interfaces
    - Languages: GLSL, HLSL9, HLSL10
    - Assembly: ARB & NV extensions, DirectX 8 & 9
- Shader source virtual file system for compilation
  - Allows #include to find files like a C compiler
  - Callback for providing #include'ed shader source
- Improved handling of GLSL generic profiles
- Uses EXT_direct_state_access for performance

Cg 2.1 Supported Platforms

- Windows
  - All flavors: 2000, XP, and Vista
  - All Direct X versions: DirectX 8, 9, and 10
  - All OpenGL versions: OpenGL 1.*x* + ARB extensions, 2.0, 2.1, 3.0, plus NV extensions
- Mac OS X
  - Mac OS 10.4 (Tiger) & Mac OS 10.5 (Leopard)
  - Both 32-bit x86 and x86_64 64-bit available
  - Compiled for both x86 and PowerPC
    - So-called “fat” binaries
- Linux x86
  - Both 32-bit x86 and x86_64 64-bit available
- Solaris 10 x86

Cg 2.1 Beta Available Now

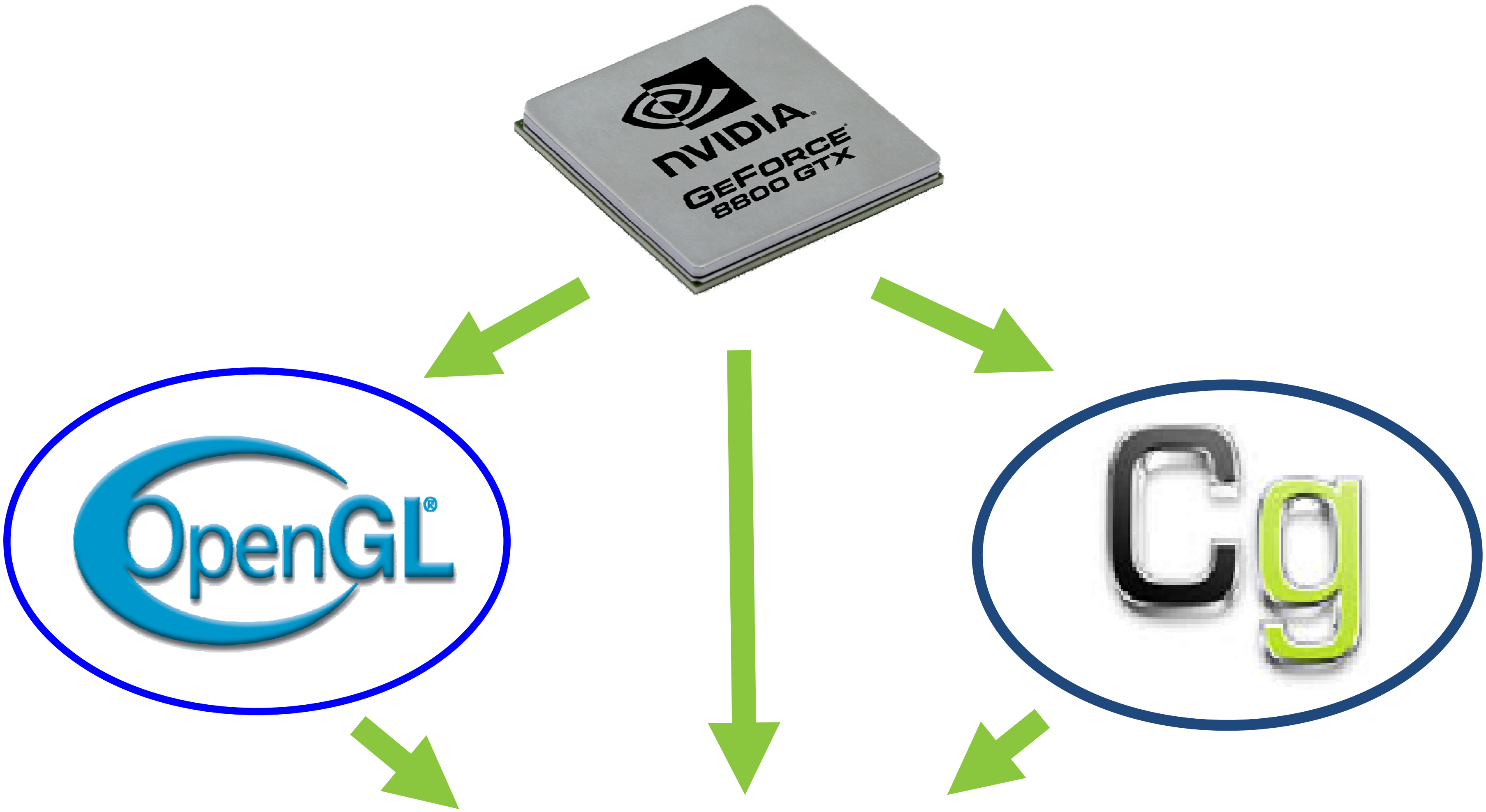
- On NVIDIA's web-site now (August 2008)
  - http://developer.nvidia.com/object/cg_2_1_beta.html

- Use NVIDIA Cg Developer Forum to learn more
  - http://developer.nvidia.com/forums/index.php?showforum=14

Summary

- Cg provides broadest platform support for any GPU shading language & platform
  - NVIDIA, ATI, Mac, Windows, Linux, Solaris, PlayStation 3
  - Supports all programmable GPU generations from DirectX 8 to GeForce 8 with latest DirectX 10 features
  - Cross-compile to GLSL, HLSL9, or HLSL10
  - Supports both run-time API and command line compiler
- Includes CgFX "effect system"
  - Compatible with Microsoft's FX effect system
  - Use NVIDIA's FX Composer 2 for authoring
  - Easily integrate CgFX with NVIDIA's NVSG scene graph
- Supports latest hardware features
  - Geometry shaders
  - Constant buffers
  - Texture arrays, etc.

Call for Innovation



<Your Innovation Here>