Hardware Shading for Artists

1. **Anisotropic Hair Shafts**
   
   ```
   float s = 0.5*dot(worldTangent.xyz, v2f.worldEyeDir.xyz);
   half4 aniso = tex2DAnisotex(s, s+1);
   ```

2. **Reflective Catchlights**
   
   ```
   float3 r = reflect(v2f.worldEyeDir, v2f.worldNormal);
   float3 envReflect = f3texCube(g_nlight_cube, r);
   ```

3. **Expressive Blendshape Animation**
   
   ```
   objectCoord.xyz = objectCoord.xyz + morphWeight3 * a2v.coordMorph3;
   ```

4. **Skin Ball Variations**
   
   ```
   fixed3 sheen = 0.7*spec + tex2D(wilyMap, diffuseCoord.xyz);
   ```

5. **Color Translucence**
   
   ```
   fixed3 diffTerm = diffTerm + 0.3 * bgimg * tex2D(translucenceMap2, diffCol.xy);
   ```

6. **Goosebumps**
   
   ```
   half3 gbump = 2.0*(tex2D(goosebumps, diffuseCoord.xy) - 0.5);
   half  fn = normalizeColorBump(tann* gbump; y + bnorm*gbump.x + b2norm);
   ```

7. **Subsurface Blood Layers**
   
   ```
   fixed3 diffuse = texCUBE(g_diffuseCube), worldNormal) * skinColor * skinsilhouetteVec.x;
   ```

8. **Robust Skeletal Animation**
   
   ```
   worldNormal = worldNormal + a2v.boneWeight0_3, 3 * vecMul(model3a2v, boneIndex3, 1.23, objectNormal.xzy);
   ```

© 2002 NVIDIA Corporation.
Today’s Speakers

Steve Burke
NVIDIA

John Versluis
Inevitable Entertainment
Hardware Shaders in Games

Hardware Shaders Bring Your Game Closer to Cinematic Quality
Cinematic Gaming on the Horizon
A Great time for Hardware Shading

- Convergence of film and real-time rendering
- Large number of high-end cards in market
- High-level shading languages; Cg and HLSL
- Next-generation graphics chips
Course Objective

- Discuss artist tools for using hardware shaders inside 3D applications.
- Provide artists with a better understanding of hardware shaders and the workflow of creating and editing shaders.
1. Getting Started with Hardware Shaders

- Tools for 3ds max, Maya, and XSI
- Comparison of different software implementations
- Exporting to a Game Engine
- Other Tools
What Does Cg look like?

**Assembly**

```
... RSQR R0.x, R0.x;
MULR R0.xyz, R0.xxx, R4.xyz;
MOVR R5.xyz, -R0.xyz;
MOVR R3.xyz, -R3.xyz;
DP3R R3.x, R0.xyz, R3.xyz;
SLTR R4.x, R3.x, {0.000000}.x;
ADDR R3.x, {1.000000}.x, -R4.x;
MULR R3.x, R3.xxx, R5.xzy;
MULR R0.xyz, R0.xyz, R4.xxx;
ADDR R0.xyz, R0.xyz, R3.xyz;
DP3R R1.x, R0.xyz, R1.xyz;
MAXR R1.x, {0.000000}.x, R1.x;
LG2R R1.x, R1.x;
MULR R1.x, {10.000000}.x, R1.x;
EX2R R1.x, R1.x;
MOVR R1.xyz, R1.xxx;
MULR R1.xyz, {0.900000, 0.800000, 1.000000}.xyz, R1.xyz;
DP3R R0.x, R0.xyz, R2.xyz;
MAXR R0.x, {0.000000}.x, R0.x;
MOVR R0.xyz, R0.xxx;
ADDR R0.xyz, {0.100000, 0.100000, 0.100000}.xyz, R0.xyz;
MULR R0.xyz, {0.100000, 0.800000, 0.800000}.xyz, R0.xyz;
ADDR R1.xyz, R0.xyz, R1.xyz;
...```

**Cg**

```
... COLOR cSpec = pow(max(0, dot(Nf, H)), phongExp).xxx;
COLOR cPlastic = Cd * (cAmbi + cDiff) + Cs * cSpec;
```

Simple phong shader expressed in both assembly and Cg
How Does CgFX Relate to Cg?

- CgFX describes an entire effect – Cg implements a particular function required by an effect
- CgFX describes all the parameters (and their meaning or semantics) that the app has to provide – automatic parameter discovery
- CgFX can describe complex multi-pass effects
- CgFX can handle multiple techniques

*CgFX syntax is a superset of Cg syntax and can contain Cg code or assembly code*
Tools for Hardware Shading

- **3ds max 5**
  CgFX Plug-in for 3ds max
dds plugin for 3ds max

- **Maya 4.5**
  Maya Cg Plug-in

- **XSI 3.0**
  Built-in support for Cg

*The three most popular 3d apps all support hardware shaders in the viewports*
Cg implementation: 3ds max 5

CgFX Viewport Manager

Intuitive artist controls (sliders, color pickers, etc.)

Dynamic, shader-specific GUI

Supports .fx file format

Multiple Techniques for fallbacks
Cg implementation: Maya 4.5

Supports .fx file format

Intuitive, shader-specific, artist controls
Slider control over key real-time parameters (e.g., bump depth)

Sample shaders include:
Bumpy Shiny, Toon, Anisotropic Metal, Ghostly, Refraction Dispersion, Rainbow

Integrated with Maya’s lights

CgFX integrated with Maya’s Hypershade
Cg implementation: Softimage|XSI 3.0

Cg Integration in XSI’s Render Tree

Net View for help, samples & documentation

Interactive shader builder

Shipping with XSI 3.0

Direct Cg code editing and compilation
Comparison of Cg Implementations

- Cg vs. CgFX
- Application-specific implementations
- DirectX and Open GL

The different software implementations are more alike than not.
Exporting to Your Game Engine

Shaders can be precompiled to assembly or compiled at run-time:

- assembly can be hand-tuned if necessary
- Shaders can be compiled to either DirectX or OpenGL
- Cg run-time available now

You will need to create an exporter to use the shaders you create with these tools
The CgFX Viewer can be used as a production resource and a code example for implementing CgFX.
2. Hardware Shader Workflow

- Designing Shaders and Using Existing Shaders
- Artist-Configurable Parameters
- Editing Shader Parameters
- Exporting Shader Parameters to Game Engine
Cg supports DirectX and OpenGL.

It runs on Windows and Linux.

It supports hardware from NVIDIA, ATI, Matrox and any other hardware that supports OpenGL or DirectX.
Create or Acquire Shaders

- cgshaders.org
- Cg Browser
- In-house libraries

Shaders written in assembly or Cg
**Art / Programmer Relationship**

**Artist**
- Creates maps and tweaks parameters to achieve desired look
- Provides feedback to programmer

**Graphics Programmer**
- Creates the plumbing for shader
- Determines which parameters are configurable

*Both artist and programmer can work together for maximum efficiency. Each does what they do best.*
Customizing Shader Parameters

Customizable Parameters are specific to each effect

Shader changed by selecting a new fx file

Color and numeric values can be changed

Bitmaps can be swapped
Lights and Animation

Shader reacts to changes in light position

Animate parameters with standard 3ds max keyframe tools

Shader reacts to changes in light position
Saving Shader Customizations

Changes made to shader are saved in the 3ds max file.

Parameter settings can be exported to game engine.

Loading new shaders is as simple as selecting a new fx file.
View Shader in Game Engine

- View the customized shader in either Direct3D or OpenGL
- Multiple Techniques can be used
3. The Gritty Details of HW Shaders

- Overview of Shaders
- Hardware Shaders and Software Shaders
- Artist/Programmer teamwork
Vertex and Pixel Shaders offer programmability so that surfaces can be made of unique and individual ‘stuff’
Vertex Shaders

Vertex Shaders are both Flexible and Quick

Linear Interpretation of vertex lighting values

vertex shaders can be used to move/animate verts
Pixel Shaders

Each pixel is calculated individually

Pixel shaders have limited or no knowledge of neighbouring pixels
Software Shaders are not for Real-time

- Complexity
- Flexibility
- Quality over Speed

Unused Texture Slots
Conditional Code
No relationship to Hardware
Hardware Shaders are Streamlined

- 1/60th of a second
- Hardware-friendly techniques
- Interactivity

Configurable shader parameters

Reflection Level 0.89
Reflection Color RGB 188,254,122
Small Efficient Shaders

Multiple, narrowly-targeted shaders are more efficient/faster than large all-purpose shaders
Fallback Techniques

Several versions of the same shader can be contained into a single fx file
Lighting Models

Texture clearly visible
Texture only visible in highlights
Texture visible in most areas but overall illumination is less

Blinn | Metal | Oren-Nayar

Same textures and lighting conditions with different lighting models.
Lights are Part of the Shader Definition

Lights (Number, kind, color, etc.) must be programmed into each shader.

Lights are not separate scene objects as they appear to be in software rendering.
Limitations

- Render to Texture Effects
- Speed Limitations
- Shadows and other complex rendering techniques

*CgFX works best for editing the look of materials.*
Thanks! Questions?

Steve Burke  
NVIDIA  
sburke@nvidia.com

John Versluis  
Inevitable  
jversluis@inevitable.com