Hardware Shading for Artists

Just an image to remind us of the power of programmable shaders.
Today’s Speakers

Steve Burke
NVIDIA

John Versluis
Inevitable Entertainment

Experience in game industry, high-end 3D art, blah, blah, blah,
Interest in real-time work and role at NVIDIA, working with developers to
raise the overall level of quality in real-time art by providing both technical
information and artistic information to artists.
John’s experience in games and high-end 3d art. Expertise at technical
art and real-time game issues.
Hardware Shaders in Games

Discuss some of the current games using hardware shaders, effects on gameplay, quality of user experience.

Ability for artist to be more expressive. Effects also. Not just hardware shaders.
Discuss some of the current games using hardware shaders, effects on gameplay, quality of user experience.

Ability for artist to be more expressive. Effects also. Not just hardware shaders.
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

There have been a lot of changes in real-time 3D in the last year. Now, more than ever, it is practical and profitable to support high-end vertex and pixel shaders in your game.

CPUs getting more and more use with fun stuff like physics calculations and animation
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.

I want to show the artists the tools involved in hardware shader design but also to provide a solid understanding of what hardware shaders are, how they work and how artists can use them to their advantage.

There are substantial differences between how shaders normally work in a 3D program and how they work in real-time. I want to point out these differences and basically give artists they need to use the tools.

Important for artist to know constraints. Seeing what is possible allows you to plan better what type of effects you want to create.

First, give a quick overview of the workflow and the tools involved. Second, talk about the shaders and all the nuances of working with hardware shaders as opposed to software 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

Cg is one of the big factors in why you can now work with hardware shaders inside your favorite 3D programs.

Cg allows for shader writing in a painless way. Also have NVB Exporter and CgFX Viewer but they aren’t necessary.

Other tools can support Cg or CgFX
What Does Cg look like?

Cg, is infinitely more understandable to the programmer – they clearly can see elements like a specular highlight, a combination of ambient and diffuse colors… But they’ll still see some GPU functions – such as a dot product – explicitly addressed in the code. It’s a high-level language that works they way hardware rendering does.

Renderman offers the a higher level of abstraction than Cg but doesn’t correlate to the hardware.
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 is one of the big factors in why you can now work with hardware shaders inside your favorite 3D programs.
Cg allows for shader writing in a painless way. Also have NVBExporter and CgFX Viewer but they aren’t necessary
Other tools can support Cg or CgFX. Need exporter to bring tweaked shaders into your game engine.
Finally, here’s the latest code in action, showing a bunch of different CgFX shaders right in max, alongside a game engine using the same shaders. We’re showing this off all day, every day at the NVIDIA booth.
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.

Discuss similarities and differences among the different programs. Cg vs.
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

Discuss tradeoff for run-time versus pre-compile
CgFX Viewer

The CgFX Viewer can be used as a production resource and a code example for implementing CgFX.
I want to show the artists the tools involved in hardware shader design but also to provide a solid understanding of what hardware shaders are, how they work and how artists can use them to their advantage.

There are substantial differences between how shaders normally work in a 3D program and how they work in real-time. I want to point out these differences and basically give artists they need to use the tools.

Important for artist to know constraints. Seeing what is possible allows you to plan better what type of effects you want to create.
Cg is cross-API and cross-platform. It compiles to either DirectX or OpenGL. That makes Cg cross-platform. These tools run on non-NVIDIA cards as well as NVIDIA cards. Cg can be pre-compiled to assembly or can be built into your game engine.
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.*

This is the standard way of working and is similar to the way that Pixar and other film companies work. You’ll always be better off by having a good relationship with the programming staff.
Customizing Shader Parameters

- Customizable Parameters are specific to each effect
- Bitmaps can be swapped
- Color and numeric values can be changed
- Shader changed by selecting a new fx file
Lights and Animation

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

Multiple Techniques can be used

View the customized shader in either Direct3D or OpenGL
3. The Gritty Details of HW Shaders

- Overview of Shaders
- Hardware Shaders and Software Shaders
- Artist/Programmer teamwork

Overview of Shaders, be clear about what a shader is. Big differences between hardware and software shaders. Nobody likes teamwork.
Vertex and Pixel Shaders offer programmability so that surfaces can be made of unique and individual ‘stuff’.

Vertex and Pixel shaders offer programmability of surfaces so that the same model and texture data can produce many different looks. Unlike Fixed-function pipeline real-time, not everything is made of the same stuff.

Surface properties like reflectance, light dispersion, shininess, etc. can be programmed on a per vertex or per pixel level. That gives complete control over the look of surfaces.

Some effects like reflection and refraction just can’t be done with a simple texture maps and a fixed-function pipeline.

Programmable shaders are the stuff you put into the GriGri bag to make beautiful real-time 3D voodoo.
Vertex Shaders

Vertex Shaders are both Flexible and Quick

Not always obvious to an artist whether a shader is a vertex shader or a pixel shader.

In general, pixel shaders execute faster since they operate on a vertex level which is far less data-intensive than working on a pixel level.

Vertex shaders execute before pixel shaders and they can both move vertices and shade a surface. For shading, vertex shaders offer less quality than pixel shaders because they can only do linear interpretation between vertex values.
Pixel Shaders

Pixel shaders calculate the color for any given pixel. Generally the final pixel color is a function of the shading equation and the location of the camera, light, and pixel location in screen space. Most shaders don’t return values to the program. They just affect the final screen color. For some complex effects Multipass shaders may be used. Multipass shaders require CPU code to manage the effect.

Pixel shaders affect both diffuse components of the surface as well as the specular components. Of course, pixel shaders can also create any number of lighting models realistic and non-realistic.
Software Shaders are not for Real-time

Artist visualization. Not necessarily scientific.
You cannot expect a shader with an arbitrary level of complexity and flexibility to run in real-time
Software shaders are most often run with just a few maps and features enabled. (You can’t afford this type of inefficiency in real-time)
Rendering speed takes a back seat to the flexibility of a shader
Streamlined does not mean simplistic but rather efficient
Variability with a fixed number of parameters. You adjust hardware shaders by specifying different maps and parameter values.
You can't efficiently disable parts of shader you aren't using. Best to use different shaders.
Basic plumbing of the shader is determined by the graphics programmer on a project. He/she determines which parts of the shader are artist configurable.
Small Efficient Shaders

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

Two approaches to writing shaders. First is to have a few very configurable shaders used for everything. The other is to have many small, narrowly-focused shaders.

Hardware shaders more naturally fall into the second category since you want to maximize efficiency. You’ll likely have many slightly different shaders to work with.

Pixel shaders are generally math and texture intensive. More complex shaders become increasingly math-intensive. Shader efficiency is FAR more important than geometric efficiency.

Important also to consider that hardware likes data to be sent a certain way; x number of texture and math operations per clock, can’t just throw random data at full speed. Need to optimize. An extra texture may slow the shader down considerably.
Fallback Techniques

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

Scalability across hardware, platforms, and LOD levels
Hardware Shaders must generally support graphics chips with different capabilities
Fallback techniques are shaders targeted for different platforms or different levels of hardware capability
A single shader can contain several or no fallbacks. Techniques can be written in Cg or assembly.
The lighting model is only part of what makes each shader unique. Multiple lighting models can be combined in a given shader. Most lighting models are a combination of normal angle, eye angle and light direction. Most lighting models attempt to simulate realism with a limited amount of complexity. Non-photo-real lighting models are also common.
Lighting is a function of the shader. The lighting models in use are part of the shader but so are the actual light definitions.

Shaders can light a surface in any arbitrary way; realistic, self-illuminated, etc.

Shaders can be programmed to look for lights in a given scene. What information the shader uses from these lights is entirely dependent upon the shader. Some shaders may take a complete definition. Others may look only at the light position or light color.
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