Integrating Shaders into Your Game Engine

Bryan Dudash
NVIDIA
Developer Technology
Agenda

- Why shaders?
- What are shaders exactly?
  - Evolution of graphics
- Using Shaders
  - High Level Shading Languages
  - C++ side API and semantics
- Tools
Why Shaders?

- Pixel Shaders are the #1 feature that will visually differentiate next-gen titles
- Distinct materials
  - Great way to show detail without geometry
  - Not everything matte or plastic
  - Moving away from just Blinn/Phong
- Custom light types
  - Volumetric lights
  - Not limited to OpenGL fixed pipeline
No Shaders vs Shaders

Flat texture, single texture, vertex lighting, no shadow

Bump mapped, multi texture, per pixel lighting, soft shadow

Doom 3 courtesy of id Software. All Rights Reserved.
Per Pixel Lighting

- Bump mapping / Normal Mapping / Per-Pixel Lighting are synonyms
  - Blinn Diffuse Specular lighting
  - With Tangent space Bump mapping

- Instead of calculating lighting on a per-vertex normal, use a per-pixel normal instead
Two quads lit per pixel
Pipelined Architecture

CPU → Geometry Storage → Vertex Processor → Rasterizer → Fragment Processor → Frame buffer

Texture Storage + Filtering

Vertices → Pixels
What are Shaders?

- User-defined vertex and fragment processing
  - Custom animation, lighting, image processing, etc.

- Ubiquitous platform & API support
  - PCs, next-generation consoles, cellular phones
  - Direct3D, OpenGL, OpenGL-ES

- Programmed in C-like high level languages
  - HLSL (Direct3D)
  - GLSL (OpenGL)
  - GLSL-ES (OpenGL-ES)
  - Cg (OpenGL, OpenGL-ES)
Shader Taxonomy

- Hardware functionality often described relative to Direct3D shader models 1 – 3

- Newer shader models increase programmability

- SM 1: Fixed-point color blending, static dependent texturing, <= 16 operations

- SM 2: Floating-point arithmetic, programmable dependent texturing, <= 64 operations

- SM 3: Branching & subroutines, 1000s of operations
PC/DirectX Shader Model Timeline

- **1998**: DirectX 5, Riva 128
- **1999**: DirectX 6, Multitexturing, Riva TNT
- **2000**: DirectX 7, T&L TextureStageState, GeForce 256
- **2001**: DirectX 8, SM 1.x, GeForce 3, Cg
- **2002**: DirectX 9, SM 2.0, GeForceFX
- **2003**: DirectX 9.0c, SM 3.0, GeForce 6
- **2004**: Quake 3, Giants, Halo, Far Cry, UE3

All images courtesy of respective companies. All Rights Reserved.
DirectX 8, SM 1.x / OpenGL 1.4

- Programmable vertex shaders
  - Up to 128 floating-point instructions

- Programmable pixel shaders
  - Up to 16 fixed-point vector instructions and 4 textures
  - 3D texture support
  - Up to 1 level of dependent texturing

- Advanced Render-to-Texture support

- Example Hardware
  - GeForce 3, ATI Radeon 8500, XGI Volari V3, Matrox Parhelia
SM 1.x-era Game: Halo

- Vertex shaders used to add fresnel reflection to ice
- Pixel shaders used to add glow to sun
- Render-to-texture used to distort pistol scope
- Dependent texturing used to animate & light water
DirectX 7 vs DirectX 8

Halo courtesy of Microsoft. All Rights Reserved.
Cg – C for Graphics

- High-level language designed for real-time shaders
- Supported in major DCC apps (Maya, Max, XSI)
  - What artists see in tool chain matches in-game result
HLL vs Assembly

High-level source code

\[
\text{float3 } L = \text{normalize(lightPosition - position.xyz)}; \\
\text{float3 } H = \text{normalize}(L + \text{normalize(eyePosition - position.xyz)});
\]

\[
\text{color.xyz} = \text{Ke} + \text{Ka} * \text{globalAmbient} + \\
\text{Kd} * \text{lightColor} * \text{max}(\text{dot}(L, N), 0) + \\
\text{Ks} * \text{lightColor} * \text{pow}\left(\text{max}(\text{dot}(H, N), 0), \text{shininess}\right);
\]

\[
\text{color.w} = 1;
\]

Assembly

```
ADD R0.xyz, eyePosition.xyzx, -f[TEX0].xyzx; 
DP3R R0.w, R0.xyzx, R0.xyzx; 
RSQR R0.w, R0.w; 
MULR R0.xyz, R0.w, R0.xyzx; 
ADDR R1.xyz, lightPosition.xyzx, -f[TEX0].xyzx; 
DP3R R0.w, R1.xyzx, R1.xyzx; 
RSQR R0.w, R0.w; 
MADR R0.xyz, R0.w, R1.xyzx, R0.xyzx; 
MULR R1.xyz, R0.w, R1.xyzx; 
DP3R R0.w, R1.xyzx, f[TEX1].xyzx; 
MAXR R0.w, R0.w, {0}.x; 
SLER H0.x, R0.w, {0}.x; 
DP3R R1.x, R0.xyzx, R0.xyzx; 
RSQR R1.x, R1.x; 
MULR R0.xyz, R1.x, R0.xyzx; 
DP3R R0.x, R0.xyzx, f[TEX1].xyzx; 
MAXR R0.x, R0.x, {0}.x; 
POWR R0.x, R0.x, shininess.x; 
MOVXC HC.x, H0.x; 
MOVR R0.x(GT.x), {0}.x; 
MOVR R1.xyz, lightColor.xyzx; 
MULR R1.xyz, Kd.xyzx, R1.xyzx; 
MOVR R2.xyz, globalAmbient.xyzx; 
MOVR R3.xyz, Ke.xyzx; 
MADR R3.xyz, Ka.xyzx, R2.xyzx, R3.xyzx; 
MADR R3.xyz, R1.xyzx, R0.w, R3.xyzx; 
MOVR R1.xyz, lightColor.xyzx; 
MULR R1.xyz, Ka.xyzx, R1.xyzx; 
MADR R3.xyz, R1.xyzx, R0.x, R3.xyzx; 
MOVR o[COLR].xyz, R3.xyzx; 
MOVR o[COLR].w, {1}.x;
```
Impact of HLLs

- Dramatic increase in shader adoption
  - Tens of games per year to hundreds

- Shift in game development
  - Shaders become content requirement, not tech feature
  - “What do I want?”, not “what can I do?”
  - Gives control of the look of the game to artists

Unreal courtesy of Epic Games. All Rights Reserved.
DirectX 9, SM 2.0 / OpenGL 1.5

- Floating point pixel processing
  - 16/32-bit floating point shaders, render targets & textures
  - Up to 64 vector instructions and 16 textures
  - Arbitrary dependent texturing

- Longer vertex processing – 256 instructions

- Multiple Render Targets – up to 16 outputs per pixel

- Example Hardware
  - GeForce FX 5900, ATI Radeon 9700, S3 DeltaChrome
DirectX 9.0c, SM 3.0 / OpenGL 2.0

- Unified shader programming model
  - Pixel & vertex shader flow control
  - Infinite length vertex & pixel shaders
  - Vertex shader texture lookups

- Floating-point filtering & blending

- Geometry instancing

- Example Hardware
  - GeForce 6800, GeForce 7800 GTX
SM 3.0-era Game: Unreal Engine 3

- 16-bit FP blending for high dynamic range lighting
- 16-bit FP filtering accelerates glow and exposure FX
- Long shaders & flow control for virtual displacement mapping, soft shadows, iridescence, fog, etc.

Unreal Engine 3 courtesy of Epic Games. All Rights Reserved.
Using Shaders
“Effects”

- Direct3D FX and CgFX
  - ID3DXEffect or CGeffect
- Wrapper around pixel and vertex shaders
- Can Configure
  - Target shader version
  - Common case variables
- Can reference a library of shader functions
- Define multi-pass techniques
Define any variable naming you want
    Semantics make sure constants get set

float4x4 wvp : WorldViewProjection;

D3D SAS is standardized
    Supported by many applications
      FX Composer
      3D Studio Max

OpenGL semantics standardized for CgFX in 1.4
Annotations

- Custom data associated with any element of your HLSL or CgFX effect

```hlsl
sampler2D anisoTextureSampler <
    string file = "Art/stone-color.png";
> = sampler_state {
    generateMipMap = true;
    minFilter = LinearMipMapLinear;
    magFilter = Linear;
    WrapS = Repeat;
    WrapT = Repeat;
    MaxAnisotropy = 8;
};
```

- Allows you to provide hooks to set per object data
  - E.g. Used extensively by shader tools for UI controls
CgFX Semantics Demo
Demo Important Bits

- Tangent basis interpolated from vertex shader
- Single fragment shader for lighting
- An unsized array of light structures that is dynamically resized by the C++ side
- A handful of different light types that implement the light interface
  - Point light
  - Spot Light
  - Etc…
- Optional Bump mapping based on a constant
Single Lighting Function

- Sample Albedo map for base color
- Normalize interpolated vectors
  - Tangent space basis vectors
- Optionally perturb our normal based on a normal map
- Iterate over our lights and accumulate diffuse and specular
- Combine color and lighting values to produce final result
C++ Side

- Assign the light position through the effect given a handle to the variable
- Sets number of lights and light info based on program code dynamically
- Can also pick whether or not to use normal maps
  - When not using it, shader gets faster
- Any dynamic configuration can be represented as a uniform parameter or global constant
Shader Library

- Rather than writing each shader separately
- Code re-use is good!!
- Establish common interpolated values
  - Vertex to Fragment/Pixel program
  - e.g. At a base, COLOR0, TEXCOORD0 off limits
- Create a library of useful functions
  - Break everything out
  - Only costs compile time (can be preprocessed!)
Write with extensibility in Mind

- Quick hacks are for prototyping
- Same as regular code
- Establish guidelines for style
- Full preprocessor support
  - #ifdef #define etc
- Naming convention for techniques
- No Assembly!
Performance

- CPU bound, or Pixel Shader
- NVIDIA’s GPU Programming Guide
- NVIDIA provides a number of handy performance analysis tools
  - NVShaderPerf
  - NVPerfHUD
  - NVPerfKit
NVPerfHUD

What is NVPerfHUD?
How does it work?
Schedule
What is NVPerfHUD?

- Stands for: PERFormance Heads Up Display
- Overlays graphs and dialogs on top of your application
- Interactive HUD
What is NVPerfHUD?

- 4 different types of HUD
- Performance Dashboard
- Debug Console
- Frame Debugger
- Frame Profiler (New in 4.0)
How to use it

- Run your application with NVPerfHUD
- Use it as you normally do until you find:
  - Functional problem: use the debugger
  - Low FPS: use the profiler
Performance Dashboard
Performance Dashboard

FPS: 52.3  TRIS/Frame: 339400  Time: 28.7 ms/call

Press F1 for help

NVIDIA version: 4.0.321.1500
NVIDIA driver version: 8.14.10.7772
App name: C:\Program Files\Futuremark\3DMark03.exe

- Handshake with application OK
- WARNING: Possibly NON-PURE device
- DirectX 9.0TA/TA runtime detected.
- NVAPI found, enabling extended functionality.

Number of DP calls: 184
Performance Dashboard

FPS: 52.3  TRIS/Frame: 339400  Time: 28.7 ms

NVIDIA driver version: 4.14.10-7772
App name: C:\Program Files\Futuremark\3DMark03.exe

- Handshake with application OK
- WARNING: Porting NON-PURE device
- DirectX 3DRuntime runtime detected.
- NVDFMAPI found, enabling extended functionality.

Number of DP calls: 184

- Ms per frame  Driver time  CPU waits for GPU  GPU idle
Performance Dashboard

Resource monitor

Resources monitored

- Textures
- Volume Textures
- Cube textures
- Vertex Buffers
- Index buffers
- Stencil and depth surfaces
Performance Dashboard

- Speed control

- FPS: 52.3
- TRIs/Frame: 339400
- Time: 28.7
- Speed: ▶ 1.000
- Press F1 for help

- NVPerfHUD version: 4.0.321.1500
- NVIDIA driver version: 6.14.10.7772
- App name: C:\Program Files\Futuremark\
The simplified graphics pipeline

- Vertex Assembly
- Vertex Shader
- Pixel Shader
- Raster OPerations
Schedule

- Beta: August
- Release: September
NVPerfKit  Performance Analysis Toolkit

Complete Performance Instrumentation Solution
- Instrumented Driver
- NVIDIA Developer Control Panel (NVDevCPL)
- NVIDIA Plug-in for Microsoft PIX for Windows
- Direct access to performance counters via PDH
  - Support for PerfMon, Intel® VTune™, gDEBugger, and more
  - Access to performance signals inside your applications
- Includes code samples for OpenGL and Direct3D
- Opt-in security mechanism prevents unauthorized analysis
NVPerfKit  Instrumented Driver

- Provides GPU and Driver Performance Counters
- Supports OpenGL and Direct3D
- Supports SLI Counters
- Requires GeForce FX or later
- Significantly more counters available on GeForce 6 Series and later...
PerfMon already shows CPU counters. Now you can add NVPerfKit counters reported by the GPU hardware, OpenGL Driver and Direct3D Driver.
NVShaderPerf

- Same technology as Shader Perf panel in FX Composer
- Analyze DirectX and OpenGL Shaders
  - HLSL, GLSL, Cg, !!FP1.0, !!ARBfp1.0, VS1.x, VS2.x, VS3.x, PS1.x, PS2.x, PS3.x, etc.
- Shader performance regression testing on the entire family of NVIDIA GPUs, without rebooting!
Conclusion

- Use high-level shading languages
- Use FX files and Semantics
  - Either CgFX or D3D FX
- Use our tools
  - Tons of free tools
  - Tons of free examples
- Treat Shaders like C++ code
  - Good design can save tons of time in making your game look amazing!
Questions

- [http://developer.nvidia.com/CgTutorial](http://developer.nvidia.com/CgTutorial)
- Email: [bdudash@nvidia.com](mailto:bdudash@nvidia.com)
The Source for GPU Programming

developer.nvidia.com

- Latest News
- Developer Events Calendar
- Technical Documentation
- Conference Presentations
- GPU Programming Guide
- Powerful Tools, SDKs and more ...

Join our FREE registered developer program for early access to NVIDIA drivers, cutting edge tools, online support forums, and more.

developer.nvidia.com