Intro to GPU Programming (OpenGL Shading Language)

Cliff Lindsay

Ph.D. Student CS-WPI
Talk Summary

Topic Coverage

• Define Shading Languages (loosely)
• High Level View of GPU
• Functional Aspects of GPU
• Example Shaders (GPU programs)
Who Am I?

• Ph.D. Student @ WPI
• Advisor = Emmanuel
• Interests:
  – Computational Photography
  – Real-time Rendering
  – Photorealistic Rendering
  – GPU Algorithms

• Done: Published Papers, M.S. Thesis
Some Work We’ve Done

Samples
Q: What is a Programmable GPU & Why do we need it?

A:

• OpenGL Fixed Function: Can only select from pre-defined effects (90’s)
  – E.g. Only two shading models pre-defined

• Industry needs flexibility (new effects)

• GPU Shaders = programmability + access to GPU internals
History Of Real-Time Graphics

- **Virtua Fighter** (SEGA Corporation)
  - NV1
  - 50K triangles/sec
  - 1M pixel ops/sec
  - 1995

- **Dead or Alive 3** (Tecmo Corporation)
  - Xbox (NV2A)
  - 100M triangles/sec
  - 1G pixel ops/sec
  - 2001

- **Dawn** (NVIDIA Corporation)
  - GeForce FX (NV30)
  - 200M triangles/sec
  - 2G pixel ops/sec
  - 2003
Examples of New Effects

Complex Materials

Lighting Environments

Shadowing

Advanced Mapping
History of Shading Languages

Big Players

- **RenderMan** – Pixar, software based in toy story
- **Cg** – nVidia, 1st commercial SL
- **HLSL** – M$/NVidia, Cg & Xbox project (Cg/HLSL Fork)
- **GLSL** – SGI, ARB/3DLabs
- **Stanford RTSL** - Academic SLs

Several others more recently
The Motivation for Shading Languages

- Graphics hardware has become increasingly more powerful
- Programming powerful hardware with assembly code is hard
- Most GPUs support programs more than 1,000 assembly instructions long
- Programmers need the benefits of a high-level language:
  - Easier programming
  - Easier code reuse
  - Easier debugging

Assembly

```
RSQ R0, R0.x;
MUL R0, R0.x, c[11].xyzx;
MOV R1, c[3];
MUL R1, R1.x, c[0].xyzx;
DP3 R2, R1.xzyx, R1.xzyx;
RSQ R2, R2.x;
MUL R1, R2.x, R1.xzyx;
ADD R2, R0.xzyx, R1.xzyx;
DP3 R3, R2.xzyx, R2.xzyx;
RSQ R3, R3.x;
MUL R2, R3.x, R2.xzyx;
DP3 R2, R1.xzyx, R2.xzyx;
MAX R2, c[3].z, R2.x;
MOV R2.z, c[3].y;
MOV R2.w, c[3].w;
LIT R2, R2;
...
```

```
float3 cSpecular = pow(max(0, dot(Nf, H)), phongExp).xxx;
float3 cPlastic = Cd * (cAmbient + cDiffuse) + Cs * cSpecular;
```
Where Can I Use Shader Programming

- Students who learn Cg can apply their skills in a variety of situations
  - **Graphics APIs**
    - OpenGL
    - DirectX
  - **Operating Systems**
    - Windows
    - Linux
    - Mac OS
  - **Graphics Hardware**
    - NVIDIA GPUs
    - ATI GPUs
    - Other GPUs that support OpenGL and DirectX 9

Actual Screenshots of Same Shader on Different Platforms (2 of 5 on ATI Hardware)
Shader Pipeline

Like Traditional Hardware Graphics Pipeline

But:

• Has Programmable Stages
• Control Primitives In Pipeline
  – e.g. Skinning, Animation
• Not Limited In Rendering Style
  – Whatever Material Desired
• Decouple Rendering From Application
Shader Pipeline

Programmable Graphics Pipeline
Programmable Pipeline

Programmable Functionality
- Exposed via small programs
- Language similar to c/c++
- Hardware support highly variable

**Vertex Shaders**
- Input: Application geometry & per vertex attributes
- Transform input in a meaningful way

**Fragment Shaders**
- Input: Perspective Correct Attributes (interpolated)
- Transform input into color or discard
Recent Advances

- Geometry Shaders
- Texture Fetching Within Vertex Shaders
In General

**Some Fixed Functions Are Bypassed**

**Vertex Tasks**
- Vertex Transformation
- Normal Transformation, Normalization
- Lighting
- Texture Coordinate Generation and Transformation

**Fragment Tasks**
- Texture accesses
- Fog
- Discard Fragment
Rendering Pipeline

- All Operations Performed By Programmer
- Same Stages As Fixed Function
- Inject Code: **Vertex** & **Fragment** Programs

Image courtesy of Nvidia
Rendering Pipeline

Image courtesy of Nvidia
Anatomy Of GLSL: OpenGL State

**Built-in Variables**
- Always prefaced with `gl_
- Accessible to both vertex and fragment shaders

**Uniform Variables**
- Matrices (i.e. ModelViewMatrix, ProjectionMatrix, inverses, transposes)
- Materials (in MaterialParameters struct, ambient, diffuse, etc.)
- Lights (in LightSourceParameters struct, specular, position, etc.)

**Varying Variables**
- FrontColor for colors
- TexCoord[] for texture coordinates
Anatomy Of GLSL: Special Vars

**Vertex Shaders**
- Have access to several vertex attributes:
  - `gl_Color`, `gl_Normal`, `gl_Vertex`, etc.
- Also write to special output variables:
  - `gl_Position`, `gl_PointSize`, etc.

**Fragment Shaders**
- Have access to special input variables:
  - `gl_FragCoord`, `gl_FrontFacing`, etc.
- Also write to special output variables:
  - `gl_FragColor`, `gl_FragDepth`, etc.
Example: Phong Shader

Questions?

Goals
• Phong Illumination Review (1 slide)
• C/C++ Application Setup
• Vertex Shader
• Fragment Shader
• Debugging
Phong Shader Review

$Illum = ambient + diffuse + specular$

$= Ka \times I + Kd \times I \times (\cos \theta) + Ks \times I \times \cos^n(\phi)$

[Diagram Courtesy of E. Agu]
Phong Shader: Setup Steps

Step 1: Create Shaders
Create handles to shaders

Step 2: Specify Shaders
load strings that contain shader source

Step 3: Compiling Shaders
Actually compile source (check for errors)

Step 4: Creating Program Objects
Program object controls the shaders

Step 5: Attach Shaders to Programs
Attach shaders to program obj via handle

Step 6: Link Shaders to Programs
Another step similar to attach

Step 7: Enable Program
Finally, let GPU know shaders are ready
Phong Shader: App Setup

GLhandleARB phongVS, phongkFS, phongProg; // handles to objects

// Step 1: Create a vertex & fragment shader object
phongVS = glCreateShaderObjectARB(GL_VERTEX_SHADER_ARB);
phongFS = glCreateShaderObjectARB(GL_FRAGMENT_SHADER_ARB);

// Step 2: Load source code strings into shaders
glShaderSourceARB(phongVS, 1, &phongVS_String, NULL);
glShaderSourceARB(phongFS, 1, &phongFS_String, NULL);

// Step 3: Compile the vertex, fragment shaders.
glCompileShaderARB(phongVS);
glCompileShaderARB(phongFS);

// Step 4: Create a program object
phongProg = glCreateProgramObjectARB();

// Step 5: Attach the two compiled shaders
glAttachObjectARB(phongProg, phongVS);
glAttachObjectARB(phongProg, phongFS);

// Step 6: Link the program object
glLinkProgramARB(phongProg);

// Step 7: Finally, install program object as part of current state
glUseProgramObjectARB(phongProg);
This Shader Does
• Gives eye space location for v
• Transform Surface Normal
• Transform Vertex Location

varying vec3 N;
varying vec3 v;

void main(void)
{
    v = vec3(gl_ModelViewMatrix * gl_Vertex);
    N = normalize(gl_NormalMatrix * gl_Normal);
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}

Created For Use Within Frag Shader
(Update OpenGL Built-in Variable for Vertex Position)
varying vec3 N;
varying vec3 v;

void main (void)
{
    // we are in Eye Coordinates, so EyePos is (0,0,0)
    vec3 L = normalize(gl_LightSource[0].position.xyz - v);
    vec3 E = normalize(-v);
    vec3 R = normalize(-reflect(L,N));

    //calculate Ambient Term:
    vec4 Iamb = gl_FrontLightProduct[0].ambient;

    //calculate Diffuse Term:
    vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

    // calculate Specular Term:
    vec4 Ispec = gl_FrontLightProduct[0].specular
        * pow(max(dot(R,E),0.0), gl_FrontMaterial.shininess);

    // write Total Color:
    gl_FragColor = gl_FrontLightModelProduct.sceneColor + Iamb + Idiff + Ispec;
}

Worcester Polytechnic Institute
**Many things will silently fail during setup**

- No good automatic debugging tools for GLSL yet exist
- Common show-stoppers:
  - Typos in shader source
  - Assuming implicit type conversion
  - Attempting to pass data to undeclared varying/uniform variables
- Extremely important to check error codes, use status functions like:
  - `glGetObjectParameter{I|f}vARB(GLhandleARB shader, GLenum whatToCheck, GLfloat *statusVals)`
- Subtle Problems
  - Type over flow
  - Shader too long
  - Use too many registers
Phong Shader: Demo

Click Me!
GPU: More Than RT Pipeline

• Character Animation
• Ray Tracing
• General Purpose Programming
• Game Physics
Future Of GPUs

• Super Computers On The Desktop
  – GPUs = Order Of Magnitude Than CPUs

• Mobile Computing
  – Realistic Rendering On Phones
    – Mobile Applications:
      • Automotive Computing
      • Wearable Computers
      • Cameras, Phones, E-paper, Bots, ...
Questions?
References

- OpenGL Shading Language (Orange Book), Randi Rost, 2004
- Intro GLSL, Talk Slides Randi Rost 3DLabs, 2005
- Intro GLSL, Teaching Slide, Mike Bailey (my undergrad graphics teacher) U of O, 2006
- Intro GLSL, Teaching Slides, Keith O’connor, GV2 (U of Dublin)
- OpenGL Shading Language, Teaching Slides, Jerry Talton, Stanford, 2006
Shader Vertex Processing

All value are inputs to Shaders

Attribute0
Attribute1
Attribute2
...
Attribute$n$
EdgeFlag

Uniform
Texture

Vertex Shader

Temporaries

Position
ClipVertex
PointSize
Varying0
Varying1
Varying2
...
Varying$n$
EdgeFlag
Shader Fragment Processing

Same as vertex, all values are input into shader