Surface Detail 5 of 5: Shading Languages

OGLSL, Direct3D Shading

William H. Hsu
Department of Computing and Information Sciences, KSU

Public mirror web site: http://www.kddresearch.org/Courses/CIS636
Instructor home page: http://www.cis.ksu.edu/~bhsu

Readings:
Today: Section 3.2 – 3.4, Eberly 2– see http://bit.ly/vL7Str; Direct3D handout
Next class: §4.1 – 4.3, Eberly 2– Computer-Generated Animation handout
References – Christian tutorials: http://www.clockworkcoders.com/oglsl/
Toymaker shading tutorial, K. Ditchburn: http://bit.ly/gBScYK

Introduction to
CIS 536/636

Toymaker

Review:
Drawing in Direct3D

- Specify the material we wish to use for the following triangles
- Specify the texture we wish to use (if we want one or NULL if not)
- Set the stream source to our vertex buffer
- Set the PFV we will be using
- Set the index buffer we will be using
- Call the required DrawPrimitive function

void CGfxEntityCube::Render()
{
// draw a triangle list using 24 vertices and 12 triangles
   gD3dDevice->DrawIndexedPrimitive( D3DPT_TRIANGLELIST,0,0,24,0,12);
   gD3dDevice->SetIndices( m_ib);
   gD3dDevice->SetFVF( D3DFVF_CUBEVERTEX ) ;
   gD3dDevice->SetStreamSource( 0, m_vb,0, sizeof(CUBEVERTEX) ) ;
   gD3dDevice->SetTexture(0,NULL);
   
   // more code...
}

Shader Languages Overview

- OGLSL: Shader language and API developed by OpenGL, usable from within a Direct3D application
- GLSL: Shader language and API developed by the OpenGL consortium and usable from within an OpenGL application, GLSL may have a stark resemblance to OGLSL.
- HLSL: Shader language and API developed by Microsoft, only usable from within a Direct3D application
- Cg: Shader language and API developed by Nvidia, usable from within a DirectX and OpenGL application. Cg has a stark resemblance to HLSL.

Acknowledgements

Nathan H. Bean
Instructor
Teaching Assistant Coordinator
Department of Computing and Information Sciences
Kansas State University
 http://www.cis.ksu.edu/~bhsu

Andy van Dam
T. J. Watson University Professor of Technology and Education & Professor of Computer Science
Brown University
 http://www.cs.brown.edu/~andv

As others have noted, there is a lot of useful code available on the Internet. Also, it is not clear how much code is available from the course. A lot of useful code is available from the course.

Review:
Shading in Direct3D

- Hardware rendering & APIs for programmable hardware
- Vertex shaders: vertex attributes to illumination at vertices
- Pixel shaders: vertices to pixel colors, transparency

Today: Shaders, Especially OGLSL

- OpenGL Shading Language (OGLSL or GLSL) – main topic
- High-Level Shading Language (HLSL) & Direct3D
- Pixar’s RenderMan

Shading in Direct3D

Tutorials from K. Ditchburn based on Direct3D 10, HLSL

For more info, see Toymaker site: http://bit.ly/jygqZ7

Coming Up: Computer-Generated Animation Demos, Videos
**Review: Programmable Hardware**

- High-Level Shader Language (HLSL) is Microsoft’s language for programming GPUs
- Looks like C
- Example vertex and pixel shader for projective texturing (feature should appear to be projected onto the scene, as if from a slide projector)

```plaintext
// vertex shader
struct VS_OUTPUTPROJTEX // output structure
{
    float4 Pos : POSITION;
    float4 Tex : TEXCOORD0;
}

VS_OUTPUTPROJTEX VSProjTexture(float4 Pos : POSITION, float3 Normal : NORMAL)
{
    VS_OUTPUTPROJTEX Out = (VS_OUTPUTPROJTEX)0;
    Out.Pos = mul(Pos, matWorldViewProj); // transform Position
    Out.Tex = mul(ProjTextureMatrix, Pos); // project texture coordinates
    return Out;
}

// pixel shader
float4 PSProjTexture(float4 Tex : TEXCOORD0) : COLOR
{
    return tex2Dproj(ProjTexMapSampler, Tex);
}
```

Adapted from slide 2003 Wolfgang Engel, http://www.wolfgang-engel.info


---

**Review: HLSL Overview**

- High-Level Shader Language (HLSL) is Microsoft’s language for programming GPUs
- Looks like C
- Example vertex and pixel shader for projective texturing (feature should appear to be projected onto the scene, as if from a slide projector)

```plaintext
// vertex shader
struct VS_OUTPUT
{
    float4 Pos : POSITION;
    float4 Col : COLOR0;
}

VS_OUTPUT VS(float3 Pos : POSITION)
{
    VS_OUTPUT Out = (VS_OUTPUT)0;
    float4 hPos = float4(Pos, 1);
    Out.Pos = mul(hPos, wvp);
    Out.Col = float4(1, 1, 1, 1);
    return Out;
}

technique Default
{
    pass P0
    {
        // shaders
        CullMode = None;
        VertexShader = compile vs_2_0 VS();
    }
}
```

© 2009 Koen Samyn

http://knol.google.com/k/hlsl-shaders

---

**Review: OpenGL Fixed Function Pipeline**

- OpenGL FFP Diagram (for v1.5)
- New Function: Fragment (Pixel-Level) Shaders
  - Programmable pipeline – like HLSL, Cg
  - Compiles to shader objects
  - Runs on hardware: ATI Radeon 9x00+, nVidia GeForce 5x00+

---

**Review: GLSL Hybrid Shader Example**

**Vertex Shader**

```
varying float xpos;
varying float ypos;
varying float zpos;
void main(void)
{
    xpos = clamp(gl_Vertex.x,0.0,1.0);
    ypos = clamp(gl_Vertex.y,0.0,1.0);
    zpos = clamp(gl_Vertex.z,0.0,1.0);
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

**Fragment Shader**

```
varying float xpos;
varying float ypos;
varying float zpos;
void main(void)
{
    gl_FragColor = vec4(xpos, ypos, zpos, 1.0);
}
```

© 2003 – 2005 M. Christen, ClockworkCoders.com


---

**Review: GLSL Vertex Shader Example**

**Vertex Shader**

```
// vertex shader
varying float xpos;
varying float ypos;
varying float zpos;
void main(void)
{
    xpos = clamp(gl_Vertex.x,0.0,1.0);
    ypos = clamp(gl_Vertex.y,0.0,1.0);
    zpos = clamp(gl_Vertex.z,0.0,1.0);
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
}
```

© 2003 – 2006 W. Cristian, ClockworkCoders.com

http://bit.ly/gi9g47

---

**Review: GLSL Vertex Shader Example**

**Diffuse Shader (Nahe GLSL Example)**

```
// diffuse shader
uniform sampler2D diffuseMap;
uniform float diffuseIntensity;
void main(intensity)
{
    vec4 diffuseColor = texture2D(diffuseMap, texCoord);
    intensity *= diffuseIntensity;
    gl_FragColor = vec4(diffuseColor.rgb * intensity, diffuseColor.a);
}
```

Machine Problems, Projects: Will Use Combination of Shaders

© 2004 F. Rudolf, NeHe Productions

http://bit.ly/gi9g47

---
A shader manager frees memory.

```c++
void free(aShaderObject* o);
```

A shader manager loads shaders from memory or files.

```c++
aShaderObject* loadfromMemory(const char* vertexMem, const char* fragmentMem);
aShaderObject* loadfromFile(char* vertexFile, char* fragmentFile);
```

This is the easy way!

```c++
vec4 a = gl_Vertex;
vec4 a = gl_ModelViewProjectionMatrix * a;
a.y = a.y * 0.5;
a.x = a.x * 0.5;
```

This is how we write vertex shader code

```c++
void main(void)
{
    // Scaled vertex is transformed with concatenated modelview and projection matrix.
    // Incoming x and y components are scaled with a factor 0.5

    gl_Position = gl_ModelViewProjectionMatrix * a;
    a.y = a.y * 0.5;
    a.x = a.x * 0.5;
}
```

Result: Diffuse Term for Phong Shading (One Light, No Specular)

```
void AppInit(void)
{
    // Shader and Program objects.
    using namespace std;
    include ../cwc/aGLSL.h;

    aShaderManager shadermanager;
    aVertexShader myVertexShader;
    aFragmentShader myFragmentShader;

    // shader and Program objects.
    shadermanager.init();
    myVertexShader.init();
    myFragmentShader.init();

    // Load Vertex Program
    myVertexShader.loadfromFile("simple.vert")
    // Load Fragment Program
    myFragmentShader.loadfromFile("simple.frag")

    // Key function entry point
    // Compiles a shader object directly from source.
    if (myVertexShader.compile())
    {
        cout << "can't compile vertex shader!
    }
    else
    {
        cout << "can't load vertex shader!
    }

    // Compiles a shader object directly from source.
    if (myFragmentShader.compile())
    {
        cout << "can't compile fragment shader!
    }
    else
    {
        cout << "can't load fragment shader!
    }

    shadermanager.addPrograms(myVertexShader, myFragmentShader);
    shadermanager.compilePrograms();
    shadermanager.linkPrograms();
    shaderprocessor = shadermanager.loadfromFile("test.vert", "test.frag");
}
```

This is the easy way!

```c++
vec4 a = gl_Vertex;
vec4 a = gl_ModelViewProjectionMatrix * a;
```

Result: Diffuse Term for Phong Shading (One Light, No Specular)
Writing Vertex Shaders

This example shows how to manipulate the vertex position data to create the effect of a fluttering flag.

- **Shade**
- **VertexShader**
- **dxEffect**

**Code**

```
float4x4 matWorld : WORLD;
VS_OUTPUT VS(float4 Pos : POSITION, float2 tex : TEXCOORD0)
{
    float4 x = Pos;
    // make left edge look as if it is attached to a flagpole
    Pos.z += sin(Pos.y/2+angle);
    // take y position of vertex into account
    Pos.z *= Pos.x * 0.09f;
    // make flag flutter
    Pos.x = Pos.y = Pos.z = 0;
    return x;
}
```

**Expected Results**
- Fluttering flag

**Adapted from**
Toymaker © 2004 – 2010 K. Ditchburn, Teesside University
Shading in Direct3D: Pixel Shading & Rest of Pipeline

- **Application**
  - Scene management
  - Vertices, tessellation
- **Vertex Operations**
  - Transformation and Lighting (T&L)
  - Cutting, Clipping
- **Pixel Operations**
  - Triangle setup and rasterization
  - Shading, multitexturing
  - Fog, alpha test, depth buffering, antialiasing
  - Display

**Summary**

- **Last Time: Shaders and Programmable Hardware**
  - Hardware rendering & APIs for programmable hardware
  - Vertex shaders: vertices attributes to illumination at vertices
  - Pixel shaders: lit vertices to pixel colors, transparency
- **Shader Languages, Especially OGLSL**
  - Introduced in 2002 by OpenGL ARB, part of OpenGL since v1.4
  - GLSL compiler builds programs to load on OpenGL graphics cards
  - AppInit(): function to compile(), addShader(), and link()

**Terminology**

- **OpenGL Architecture Review Board – Standards Committee 2002 - 2006**
  - Introduced in 2002 by OpenGL ARB, part of OpenGL since v1.4
  - GLSL compiler builds programs to load on OpenGL graphics cards
  - AppInit(): function to compile(), addShader(), and link()
- **High-Level Shading Language**
  - Microsoft's programmable pipeline
  - Used in tandem with Direct3D
- **Other Shader Languages (SLs)**
  - Cg – Nvidia's general-purpose SL (surveyed in Lecture 13)
  - Gelato – Nvidia's production render farm SL (not covered)
  - **RenderMan** – Pixar's specification, renderer, or SL (surveyed later)