Lecture 14 of 41

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 2e – see http://bit.ly/eUq45; Direct3D handout
Next class: §4.1 – 4.3, Eberly 2e; Computer-Generated Animation handout
Reference – Christen tutorials: http://www.clockworkcoders.com/oglsli/
NeHe article #21 (NB: not an old lesson): http://bit.ly/gI9g47
Toymaker shading tutorial, K. Ditchburn: http://bit.ly/gBScYK

Lecture Outline

- Reading for Last Class: §3.1, Eberly 2e
- Reading for Today: §3.2 – 3.4, Eberly 2e; Direct3D handout
- Reading for Next Class: §4.1 – 4.3, Eberly 2e; CGA handout
- Last Time: Shaders and Programmable Hardware
  - Hardware rendering & APIs for programmable hardware
  - Vertex shaders: vertex attributes to illumination at vertices
  - Pixel shaders: lit vertices to pixel colors, transparency
- Today: Shader Languages, 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/gBScYK
- Coming Up: Computer-Generated Animation Demos, Videos
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 FVF we will be using
- Set the index buffer we will be using
- Call the required DrawPrimitive function

```cpp
void CGfxEntityCube::Render()
{
    gD3dDevice->SetMaterial( &m_material );
    gD3dDevice->SetTexture(0,NULL);
    gD3dDevice->SetStreamSource( 0, m_vb,0, sizeof(CUBEVERTEX) );
    gD3dDevice->SetFVF( D3DFVF_CUBEVERTEX );
    gD3dDevice->SetIndices( m_ib);

    // draw a triangle list using 24 vertices and 12 triangles
    gD3dDevice->DrawIndexedPrimitive( D3DPT_TRIANGLELIST,0,0,24,0,12);
}
```
Acknowledgements

Nathan H. Bean
Instructor
Outreach Coordinator
Department of Computing and Information Sciences
Kansas State University
http://bit.ly/gC3wH

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

Direct3D material from slides © 2006 – 2010 N. Bean, Kansas State University
http://bit.ly/gC3wH

Review:
Shader Languages Overview

OpenGL State Machine (Simplified) from Wikipedia: Shader
http://bit.ly/F8iBP

• HLSL: Shader language and API developed by Microsoft, only usable from within a DirectX application.

• Cg: Shader language and API developed by Nvidia, usable from within a DirectX and OpenGL application. Cg has a stark resemblance to HLSL.

• GLSL: Shader language and API developed by the OpenGL consortium and usable from within an OpenGL application.

© 2009 Koen Samyn
http://knol.google.com/k/hlsl-shaders
Review:
Programmable Hardware


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 (texture should appear to be projected onto the scene, as if from a slide projector)

```c
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;
}

float4 FSProjTexture(float4 Tex : TEXCOORD0) : COLOR
{
    return tex2Dproj(ProjTexMapSampler, Tex);
}
```

Review: HLSL Vertex Shader Example

```c
// This is used by 3dsmax to load the correct parser
string ParamID = "0x0";
// D3D material specific
float4x4 wvp : WORLDVIEWPROJ;
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

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

```glsl
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

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

Review: GLSL Vertex Shader Example

- Diffuse Shader (NeHe GLSL Example)

```
// OffSpec Shader
The diffuse lighting model is one common used lighting model. It’s a little bit harder to implement:
Vertex Shader:

varying vec3 normal;
varying vec3 vertex_to_light_vector;
void main()
{
  // Transforming The Vertex
  gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
  // Transforming The Normal To ModelView-Space
  normal = gl_NormalMatrix * gl_Normal;
  // Transforming The Vertex Position To ModelView-Space
  vec3 vertex_position_in_model_space = gl_ModelViewMatrix * gl_Vertex;
  // Calculating The Vertex Position To The Light Position
  vertex_to_light_vector = vertex_position_in_model_space - gl_LightSource[0].position;
  // Calculating The Normal To The Surface Normal
  normal_to_model_normal = normal;
  // Calculating the Normal To The Surface Normal
  normal_to_model_normal = normal;
  // Calculating the Vertex Position To The Light Position
  vertex_to_light_vector = vertex_position_in_model_space - gl_LightSource[0].position;
```

- Machine Problems, Projects: Will Use Combination of Shaders
Consider BRDF (Especially $N \cdot L$) From Last Lecture

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

### Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [1]

**Vertex Shader**
```glsl
void main(void)
{
    vec4 a = gl_Vertex;
    a.x = a.x * 0.5;
    a.y = a.y * 0.5;
    gl_Position = gl_ModelViewProjectionMatrix * a;
}
```

Q: What does this do?
A: Incoming x and y components are scaled with a factor 0.5.
Scaled vertex is transformed with concatenated modelview and projection matrix.

**Fragment Shader**
```glsl
void main (void)
{
    gl_FragColor = vec4 (0.0, 1.0, 0.0, 1.0);
}
```

Q: What does this do?
A: Makes everything green!
Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [2]

aShaderManager Class (Christen, 2003)

// globals:
aShaderManager shadermanager;
aShaderObject* shader;

// init:
shader = shadermanager.loadFromFile("test.vert","test.frag");

// draw:
shader->begin();
glutSolidSphere(1.0,32,32);
shader->end();

Methods

// load vertex/fragment shader from file
aShaderObject* loadFromFile(char* vertexFile, char* fragmentFile);
aShaderObject* loadfromMemory(const char* vertexMem, const char* fragmentMem);
bool free(aShaderObject* o);

loadFromFile: loads shader source code from file, compiles, links and returns a shader object.
loadFromMemory: instead of using files you can specify memory addresses containing a null
terminated char array of the program string.
free: if you don’t need a shader anymore you can free it. Usually this function isn’t needed, the destructor of
the ShaderManager frees memory.

© 2003 – 2005 M. Christen, ClockworkCoders.com

Without this framework:
… see next slide
(Christen, 2003)

Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [3]

AppInit Function (Christen, 2003) – Part 1 of 3

#include "../cwc/aGLSL.h"
#include <GL/glut.h>
#include <iostream>

using namespace std;

// Shader and Program Objects:
aShaderObject* myShader = 0;
aVertexShader* myVertexShader = 0;
aFragmentShader* myFragmentShader = 0;

void AppInit(void)
{
  // it is important a GL context is available before creating
  // shader and Program objects.
  myShader = new aShaderObject;
  myVertexShader = new aVertexShader;
  myFragmentShader = new aFragmentShader;
}
Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [4]

AppInit Function (Christen, 2003) – Part 2 of 3

// Load Vertex Program
if (myVertexShader->load("simple.vert") != 0)
{
    cout << "can't load vertex shader\n";
    exit(-1); // if file is missing: major error, better exit
}

// Load Fragment Program
if (myFragmentShader->load("simple.frag") != 0)
{
    cout << "can't load fragment shader\n";
    exit(-1); // if file is missing: major error, better exit
}

// Compile Vertex Program, if it fails output log
if (!myVertexShader->compile())
{
    cout << "***COMPILER ERROR:
    cout << myVertexShader->getCompilerLog() << endl;
}

// Compile fragment Program, if it fails output log
if (!myFragmentShader->compile())
{
    cout << "***COMPILER ERROR:\n    cout << myFragmentShader->getCompilerLog() << endl;
}

// Add (compiled Programs) to object
myShader->addShader(myVertexShader);
myShader->addShader(myFragmentShader);

// Link the Program
if (!myShader->link())
{
    cout << "**LINKER ERROR\n    cout << myShader->getLinkerLog() << endl;
}
} // AppInit()

Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [5]

AppInit Function (Christen, 2003) – Part 3 of 3

// Compile fragment Program, if it fails output log
if (!myFragmentShader->compile())
{
    cout << "***COMPILER ERROR:\n    cout << myFragmentShader->getCompilerLog() << endl;
}

// Add (compiled Programs) to object
myShader->addShader(myVertexShader);
myShader->addShader(myFragmentShader);

// Link the Program
if (!myShader->link())
{
    cout << "**LINKER ERROR\n    cout << myShader->getLinkerLog() << endl;
}
} // AppInit()
Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [6]

DrawFrame & AppExit (Christen, 2003)

//*************************************************************
// Draw one Frame (don't swap buffers)
void DrawFrame(void)
{
    myShader->begin();
    glutSolidTeapot(1.0);
    myShader->end();
}
//*************************************************************

// use AppExit to clean up
void AppExit(void)
{
    if (myShader!=0) delete myShader;
    if (myVertexShader!=0) delete myVertexShader;
    if (myFragmentShader!=0) delete myFragmentShader;
}

Loading Programs without aShaderManager class (Steps 1 – 2a of 3)

aGLSL.h defines:
aVertexShader: Class to manage vertex shader programs
aFragmentShader: Class to manage fragment shader programs
aShaderObject: Class to manage shader objects

Step 1: Declare shader programs and shader objects
aShaderObject* myShader = 0;
aVertexShader* myVertexShader = 0;
aFragmentShader* myFragmentShader = 0;

Step 2: Load, add and compile/link programs in AppInit()

Reserve memory and initialize objects (it is important an OpenGL context already exists when you do this)
myShader = new aShaderObject;
myVertexShader = new aVertexShader;
myFragmentShader = new aFragmentShader;
Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [8]

Loading Programs without aShaderManager class (Steps 2b – 2d of 3)

Step 2: Load, add and compile/link programs in AppInit() (continued)

Load:
   myVertexShader->load("simple.vert");
   myFragmentShader->load("simple.frag");

Compile:
   myVertexShader->compile();
   myFragmentShader->compile();

You can access compiler errors with:
   char* getCompilerLog(void);
   (getCompilerLog is defined in aVertexShader and aFragmentShader)

Add (compiled) programs to the object and link it:
   myShader->addShader(myVertexShader);
   myShader->addShader(myFragmentShader);
   myShader->link();

You can access linker errors with "myShader->getLinkerLog()" similar to compilerLog.

Tutorial 1: Loading, Compiling, & Linking OGLSL Programs [9]

Loading Programs without aShaderManager class (Step 3 of 3)

Step 3: Use shader (see rest of Tutorials, #2 – 10!)

   myShader->begin();
   ... {draw something with GL} ... 
   myShader->end();

Adapted material © 2003 – 2005 M. Christen, ClockworkCoders.com
http://bit.ly/i5IFL0
#5 Phong shading in OpenGL
http://bit.ly/5DFl4
#6 Texture Mapping
#7 Color Keys (Transparency Masks)
#8 Multitexturing
http://bit.ly/q6gYoY
#9 Procedural Texture (Procedural Materials)
#10 Cartoon Shading

Adapted from material © 2003 – 2005 M. Christen, ClockworkCoders.com
http://bit.ly/i5IFL0
Computing & Information Sciences
Kansas State University
Shading in Direct3D [1]: Writing Vertex Shaders

Defining TVertex data structure: position/normal/texture tuple:

```cpp
struct TVertex {
    D3DXVECTOR3 position;
    D3DXVECTOR3 Normal;
    D3DXVECTOR3 Tex;
};
```

Vertex declaration to describe this structure:

```cpp
const D3DVERTEXELEMENT9 dec[4] = {
    {0, 0, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_POSITION, 0},
    {0, 12, D3DDECLTYPE_FLOAT3, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_NORMAL, 0},
    {0, 24, D3DDECLTYPE_FLOAT2, D3DDECLMETHOD_DEFAULT, D3DDECLUSAGE_TEXCOORD, 0},
    D3DDECL_END()};
```

Each line corresponds to one of the elements in TVertex. The data in each line is:

<table>
<thead>
<tr>
<th>WORD Stream</th>
<th>WORD Offset</th>
<th>BYTE Type</th>
<th>BYTE Method</th>
<th>BYTE Usage</th>
<th>BYTE UsageIndex</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

We need to tell Direct3D about our vertex declaration using the following call:

```cpp
IDirect3DVertexDeclaration9 m_vertexDeclaration;
gDevice->CreateVertexDeclaration(dec, &m_vertexDeclaration);
```

To render:

```cpp
gDevice->SetStreamSource(0, m_vb, 0, sizeof(TVertex));
gDevice->SetVertexDeclaration(m_vertexDeclaration);
```

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

Global:

```cpp
float4x4 matWorld : WORLD;
```

In application:

```cpp
dxEffect->SetMatrix("matWorld", &matWorld);
```

Shader output structure: transformed vertex position & texture coordinate

```cpp
struct VS_OUTPUT {
    float4 Pos : POSITION;
    float2 tex : TEXCOORD0;
};
```

```cpp
VS_OUTPUT VS(float4 Pos : POSITION, float2 tex : TEXCOORD0) {
    float angle = (time % 360) * 2; // angle to use for sine wave
    Pos.z = sin(Pos.x * angle); // wave effect
    // take y position of vertex into account
    Pos.z += sin(Pos.y / 2 * angle);
    // make left edge look as if it is attached to a flagpole
    Pos.z += Pos.x * 0.03f;
}
```

Shading in Direct3D [2]: Using Vertex Shaders

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

Global:

```cpp
float4x4 matWorld : WORLD;
```

In application:

```cpp
dxEffect->SetMatrix("matWorld", &matWorld);
```

Shader output structure: transformed vertex position & texture coordinate

```cpp
struct VS_OUTPUT {
    float4 Pos : POSITION;
    float2 tex : TEXCOORD0;
};
```

```cpp
VS_OUTPUT VS(float4 Pos : POSITION, float2 tex : TEXCOORD0) {
    float angle = (time % 360) * 2; // angle to use for sine wave
    Pos.z = sin(Pos.x * angle); // wave effect
    // take y position of vertex into account
    Pos.z += sin(Pos.y / 2 * angle);
    // make left edge look as if it is attached to a flagpole
    Pos.z += Pos.x * 0.03f;
}
```
**Shading in Direct3D:**
**Pixel Shading & Rest of Pipeline**

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

Adapted from Toymaker © 2004 – 2010 K. Ditchburn, Teesside University
http://bit.ly/gBScYK

Per-Pixel Shader for Diffuse Lighting

**Next: CGA Videos (Demos & Trailers)**

- **Monsters Inc.** (2001)
- **Monsters Inc. 2** (2012) © Disney/Pixar
- **Kung-Fu Panda** © 2008 DreamWorks Animation SKG
- **Happy Feet** © 2006 Warner Brothers
- **Toy Story** (1995)
  - **Toy Story 2** (1999)
  - **Toy Story 3** (2010) © Disney/Pixar
- **Shrek** (2001)
  - **Shrek 2** (2004)
  - **Shrek the Third** (2007)
  - **Shrek Forever After** (2010) © DreamWorks Animation SKG
- **Toy Story 3** (2010) © Disney/Pixar
- **I WANT YOU FOR PIXAR**
- **Luxo Jr.** © 1986 Pixar Animation Studios
- **Tron: Legacy** © 2010 Walt Disney Pictures
- **Wall-E** © 2008 Disney/Pixar

Summary

- Last Time: Shaders and Programmable Hardware
  - Hardware rendering & APIs for programmable hardware
  - Vertex shaders: vertex attributes to illumination at vertices
  - Pixel shaders: lit vertices to pixel colors, transparency

- Shader Languages, Especially OGLSL
  - OpenGL Shading Language (OGLSL or GLSL) – main topic
  - High-Level Shading Language (HLSL) & Direct3D

- Shading in Direct3D
  - Tutorials from K. Ditchburn based on Direct3D 10, HLSL
  - For more info, see Toymaker site: [http://bit.ly/gBScYK](http://bit.ly/gBScYK)

- Still to Come: Pixar’s RenderMan Specification, Renderer, SL

- Many More Shading Techniques: Explore for Yourself!
  - References: GLSL, HLSL/D3D, Cg, Renderman (Apodaca & Gritz)
  - Graphics Gems & GPU Gems series (some chapters online)
  - Shader books by Wolfgang Engels

Terminology

- OpenGL Architecture Review Board – Standards Committee 2002 - 2006

- OpenGL Shading Language (OGLSL aka GLSL)
  - 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
  - Shader Model n: used with D3D version (SM5 = Direct3D 11, etc.)

- 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)