Programming with OpenGL
Part 5: More GLSL

Ed Angel

Professor Emeritus of Computer Science

University of New Mexico

Objectives

- Coupling shaders to applications
  - Reading
  - Compiling
  - Linking
- Vertex Attributes
- Setting up uniform variables
- Example applications

Linking Shaders with Application

- Read shaders
- Compile shaders
- Create a program object
- Link everything together
- Link variables in application with variables in shaders
  - Vertex attributes
  - Uniform variables

Program Object

- Container for shaders
  - Can contain multiple shaders
  - Other GLSL functions

```
GLuint myProgObj;
myProgObj = glCreateProgram();
 /* define shader objects here */
glUseProgram(myProgObj);
glLinkProgram(myProgObj);
```

Reading a Shader

- Shaders are added to the program object and compiled
- Usual method of passing a shader is as a null-terminated string using the function **`glShaderSource`**
- If the shader is in a file, we can write a reader to convert the file to a string

Shader Reader

```
#include <stdio.h>

static char*
readShaderSource(const char* shaderFile)
{
    FILE* fp = fopen(shaderFile, "r");

    if ( fp == NULL ) { return NULL; }

    fseek(fp, 0L, SEEK_END);
    long size = ftell(fp);
```

Shader Reader (cont)

```
    fseek(fp, 0L, SEEK_SET);
    char* buf = new char[size + 1];
    fread(buf, 1, size, fp);

    buf[size] = '\0';
    fclose(fp);

    return buf;
}
```

Adding a Vertex Shader

```
GLuint vShader;
GLuint myVertexObj;
GLchar vShaderfile[] = "my_vertex_shader";
GLchar* vSource =
        readShaderSource(vShaderFile);
glShaderSource(myVertexObj,
          1, &vSource, NULL);
myVertexObj =
          glCreateShader(GL_VERTEX_SHADER);
glCompileShader(myVertexObj);
glAttachObject(myProgObj, myVertexObj);
```

Vertex Attributes

- Vertex attributes are named in the shaders
- Linker forms a table
- Application can get index from table and tie it to an application variable
- Similar process for uniform variables

Vertex Attribute Example

```
#define BUFFER_OFFSET( offset )
    ((GLvoid*) (offset))

GLuint loc =
  glGetAttribLocation( myProgObj,"vPosition”);
glEnableVertexAttribArray( loc );
glVertexAttribPointer( loc, 2, GL_FLOAT,
    GL_FALSE, 0, BUFFER_OFFSET(0) );
```

Uniform Variable Example

```
GLint angleParam;
angleParam = glGetUniformLocation(myProgObj,
     "angle");
/* angle defined in shader */

/* my_angle set in application */
GLfloat my_angle;
my_angle = 5.0 /* or some other value */

glUniform1f(angleParam, my_angle);
```

Double Buffering

- Updating the value of a uniform variable opens the door to animating an application
  - Execute glUniform in display callback
  - Force a redraw through glutPostRedisplay()
- Need to prevent a partially redrawn frame buffer from being displayed
- Draw into back buffer
- Display front buffer
- Swap buffers after updating finished

Adding Double Buffering

- Request a double buffer
  - glutInitDisplayMode(GLUT_DOUBLE)
- Swap buffers

```
void mydisplay()
{
        glClear(……);
        glDrawArrays();
        glutSwapBuffers();
}
```

Idle Callback

- Idle callback specifies function to be executed when no other actions pending
  - glutIdleFunc(myIdle);

```
void myIdle()
{
    // recompute display
    glutPostRedisplay();
}
```

Attribute and Varying Qualifiers

- Starting with GLSL 1.5 attribute and varying qualifiers have been replaced by in and out qualifiers
- No changes needed in application
- Vertex shader example:

```
#version 1.4
attribute vec3 vPosition;
varying vec3 color;
```

```
#version 1.5
in vec3 vPosition;
out vec3 color;
```

Adding Color

- If we set a color in the application, we can send it to the shaders as a vertex attribute or as a uniform variable depending on how often it changes
- Let's associate a color with each vertex
- Set up an array of same size as positions
- Send to GPU as a vertex buffer object

Setting Colors

```
typedef  vec3 color3;
color3 base_colors[4] = {color3(1.0, 0.0. 0.0), ….
color3 colors[NumVertices];
vec3 points[NumVertices];

//in loop setting positions

colors[i] = basecolors[color_index]
position[i] = …….
```

Setting Up Buffer Object

```
//need larger buffer

glBufferData(GL_ARRAY_BUFFER, sizeof(points) +
   sizeof(colors), NULL, GL_STATIC_DRAW);

//load data separately

glBufferSubData(GL_ARRAY_BUFFER, 0,
   sizeof(points), points);
glBufferSubData(GL_ARRAY_BUFFER, sizeof(points),
   sizeof(colors), colors);
```

Second Vertex Array

```
// vPosition and vColor identifiers in vertex shader


loc = glGetAttribLocation(program, “vPosition”);
glEnableVertexAttribArray(loc);
glVertexAttribPointer(loc, 3, GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET(0));

loc2 = glGetAttribLocation(program, “vColor”);
glEnableVertexAttribArray(loc2);
glVertexAttribPointer(loc2, 3, GL_FLOAT, GL_FALSE, 0,
    BUFFER_OFFSET(sizeofpoints));
```

Vertex Shader Applications

- Moving vertices
  - Morphing
  - Wave motion
  - Fractals
- Lighting
  - More realistic models
  - Cartoon shaders

Wave Motion Vertex Shader

```
•
in vec4 vPosition;
uniform float xs, zs, // frequencies
uniform float h; // height scale
void main()
{
  vec4 t = vPosition;
  t.y = vPosition.y
     + h*sin(time + xs*vPosition.x)
     + h*sin(time + zs*vPosition.z);
  gl_Position = t;
}
```

Particle System

```
in vec3 vPosition;
uniform mat4 ModelViewProjectionMatrix;
uniform vec3 init_vel;
uniform float g, m, t;
void main()
{
vec3 object_pos;
object_pos.x = vPosition.x + vel.x*t;
object_pos.y = vPosition.y + vel.y*t
       + g/(2.0*m)*t*t;
object_pos.z = vPosition.z + vel.z*t;
gl_Position =
  ModelViewProjectionMatrix*vec4(object_pos,1);
}
```

Pass Through Fragment Shader

```
/* pass-through fragment shader */

in vec4 color;
void main(void)
{
    gl_FragColor = color;
}
```

The University of New Mexico

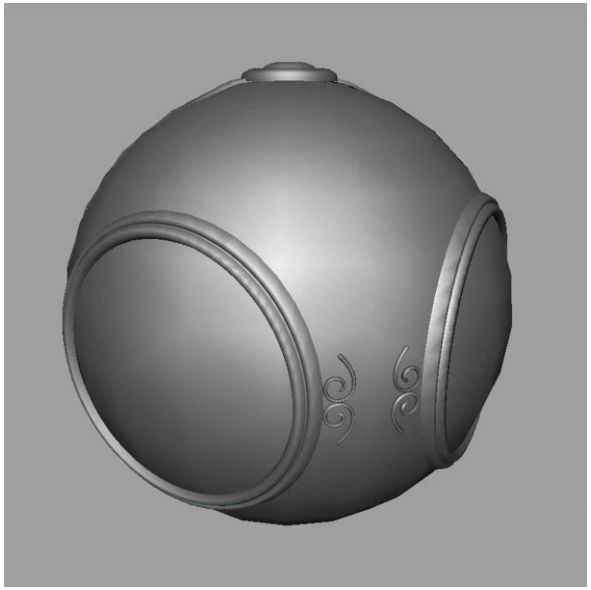
Vertex vs Fragment Lighting

per vertex lighting

per fragment lighting

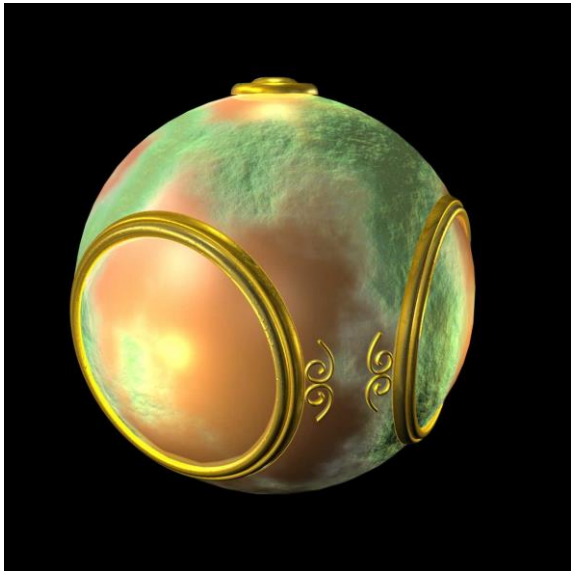
Fragment Shader Applications

Texture mapping



smooth shading

environment mapping



bump mapping