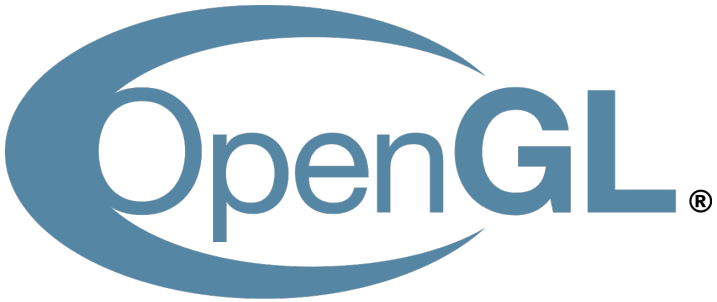
Intro to GLSL and Shaders

Dane Christensen, Brigham H. Keys, Esq.

Brigham Young University - Idaho

bkeys@bkeys.org

November 11, 2015



What are shaders?

- There are multiple shading langauges, such as HLSL, GLSL, and Kronos is releasing SPIR-V later this year.
- Just like anything you write in C++, shaders have source code, files, and are compiled.
- GLSL's syntax is very similar to the C programming language, so hopefully the learning curve will be minimal.
- Shaders are stored in IDs, which are ints.
- In this slide show we present how to compile a shader in it's own seperate source file, as it's own seperate program.
- Shaders reside entirely in the GPU when they are ran

Types of Shaders

There are two different kinds of shaders that together create a shader program, both are required for the shader to work. Chances are you are not going to need to write your own shader, as there are plenty of well written shaders online.

Vertex Shaders

States the position of a vertex (Very similar to a GL translation). Earlier when we used glVertex3f, that was a vertex shader. It tells OpenGL which points in 3D space we want to shade inbetween.

Fragment Shaders

Tells OpenGL, how we want to shade pixels on our screen. OpenGL is a very low level API so we can shade on a per-pixel basis if we wish.

About making a shader program

- Compiling a shader successfully gets us an ID for that shader.
- Requiring both a fragment shader, and a vertex shader.
- Compiler errors are possible, but checking for then is optional.
- A shader program is created by linking the fragment shader to the vertex shader.
- The shader program itself has an ID.
- Very rarely can a shader not be re-used

Examples of GLSL Program

Vertex Shader

```
File Edit Options Buffers Tools Help
attribute vec2 coord2d;
attribute vec3 v_color;
varying vec3 f_color;

void main(void) {
  gl_Position = vec4(coord2d, 0.0, 1.0);
  f_color = v_color;
}
```

Fragment Shader

```
File Edit Options Buffers Tools Help
uniform float fade;
varying vec3 f_color;

void main(void) {
  gl_FragColor = vec4(f_color.x, f_color.y, f_color.z, fade);
}
```

Compute Shaders

Most modern GPUs have many tiny cores optimized for smaller calculations. Since shaders reside entirely on the GPU, we can run a shader that performs only mathmatical calculations, *using the GPU for tasks that are not graphical at all*. APIs like Nvidia's CUDA give the developer an interface to do this in.

What is a compute shader?

- Do not have to shade anything on the screen at all.
- The shader mostly performs various mathmatical calculations.
- It enables the developer to put the load on the GPU for performing tasks better optimized for a GPU, or anything they want to put in the shader

Compiling into a shader

Getting the program ID

The function needed to compile and get the ID for the shader is
GLuint glCreateShader(GL_VERTEX_SHADER);

Create the shader program

The function to bind the source code to the shader (this must happen before compilation)
void glShaderSource(GLuint shader, GLsizei count, const GLchar **string, const GLint *length);

Create our shader!

Now we compile ths shader with:
glCompileShader(GLuint_ID);

Using our shader we created

Let OpenGL know you are using a shader

void GLuint glCreateProgram();
This behaves very much like a constructor.

We now attach both shaders to the program

void glAttachShader(program_ID, shader_ID);

Create our shader program, put it together

void glLinkProgram(program_ID);

Place this before you begin to render

void glUseProgram(program_ID);

References

A complete class for shader programs can be found in the following link under the files shader.cpp and shader.h
`https://github.com/BennyQBD/ModernOpenGLTutorial`

 https://www.opengl.org/ (2015)