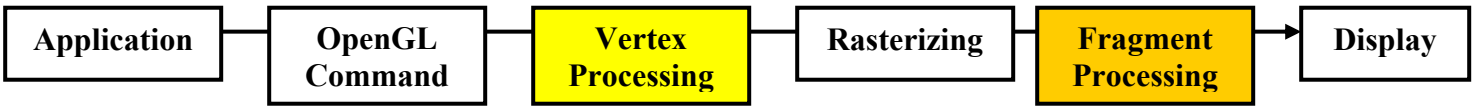
Fundamentals of Using the OpenGL Shading Language (GLSL)

Mike Bailey
Oregon State University

Where Do Shaders Fit in the Graphics Pipeline?



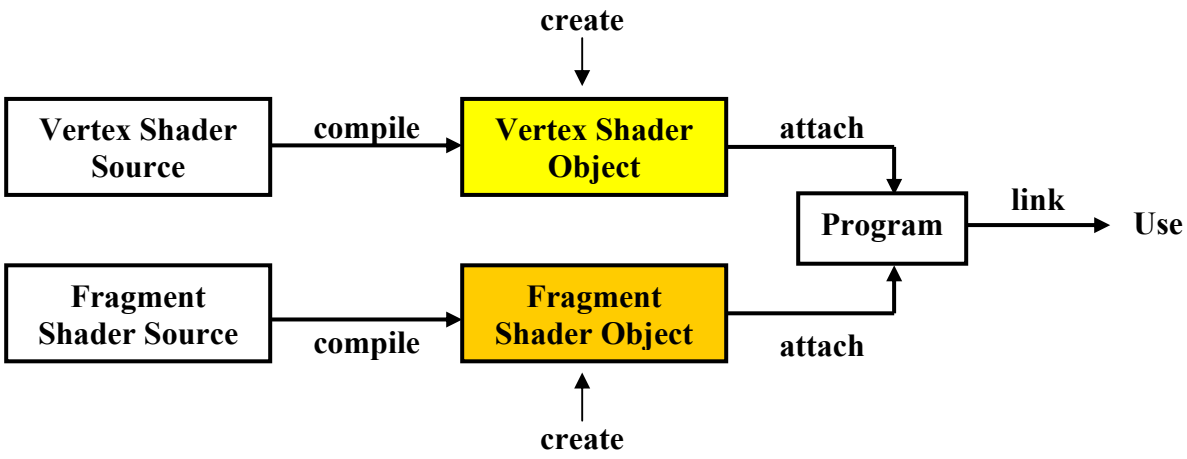
How Do I Read a Shader Source File Into a String?

```
#include <stdio.h>

FILE *fp = fopen( filename, "rb" );
fseek( fp, 0, SEEK_END );
int numBytes = ftell( fp );
GLcharARB * str = new GLubyte [numBytes+1];
rewind( fp );
fread( str, sizeof(GLcharARB), numBytes, fp );
fclose( fp );
str[numBytes] = '\0';
```

(Note: the variable *str* is re-used in the code below.)

How Do I Read, Compile, and Link Shaders?



```
int           status;
int           logLength;
GLcharARB *   log;
Glint         location;
```

Create the two shader objects:

```
GLhandleARB vertShader = glCreateShaderObjectARB( GL_VERTEX_SHADER_ARB );
GLhandleARB fragShader = glCreateShaderObjectARB( GL_FRAGMENT_SHADER_ARB );
```

Load and compile the vertex shader:

```
glShaderSourceARB( vertShader, 1, (const char**)&str, NULL );
glCompileShaderARB( vertShader );
CheckGlErrors( "Vertex Shader 1" );
glGetObjectParameterivARB( vertShader, GL_OBJECT_COMPILE_STATUS_ARB, &status );
if( status == GL_FALSE )
{
       fprintf( stderr, "Vertex shader compilation failed.\n" );
       glGetObjectParameterivARB( vertShader, GL_OBJECT_INFO_LOG_LENGTH_ARB, &logLength );
       log = new GLcharARB [logLength];
       glGetInfoLogARB( vertShader, logLength, NULL, log );
       fprintf( stderr, "\n%s\n", log );
       delete [] log;
       exit( 1 );
}
CheckGlErrors( "Vertex Shader 2" );
```

Load and compile the fragment shader:

```
glShaderSourceARB( fragShader, 1, (const char**)&str, NULL );
glCompileShaderARB( fragShader );
CheckGlErrors( "Fragment Shader 1" );
glGetObjectParameterivARB( fragShader, GL_OBJECT_COMPILE_STATUS_ARB, &status );
if( status == GL_FALSE )
{
       fprintf( stderr, "Fragment shader compilation failed.\n" );
       glGetObjectParameterARB( fragShader, GL_OBJECT_INFO_LOG_LENGTH_ARB, &logLength );
       log = new GLcharARB [logLength];
       glGetInfoLogARB( fragShader, logLength, NULL, log );
       fprintf( stderr, "\n%s\n, log );
       delete [] log;
       exit( 1 );
}
CheckGlErrors( "Fragment Shader 2" );
```

Create the program object and attach the two shaders to it:

```
GLhandleARB program = glCreateProgramObjectARB();
glAttachObjectARB( program, vertShader );
glAttachObjectARB( program, fragShader );
```

Link the program object:

```
glLinkProgram( program );
CheckGlErrors( "Shader Program 1" );
glGetObjectParameterivARB( program, GL_OBJECT_LINK_STATUS_ARB, &status );
if( status == GL_FALSE )
{
	fprintf( stderr, "Link failed.\n" );
	glGetObjectParameterivARB( program, GL_OBJECT_INFO_LOG_LENGTH_ARB, &logLength );
	log = new GLcharARB [logLength];
	glGetInfoLogARB( fragShader, logLength, NULL, log );
	fprintf( stderr, "\n%s\n", log );
	delete [] log;
	exit( 1 );
}
CheckGlErrors( "Shader Program 2" );
```

Make this program object (and its two shaders) active:

```
glUseProgramObjectARB( program );
```

How Do I Detach a Shader from a Program Object?

```
glDetachObjectARB( GLhandleARB program, GLhandleARB shader );
```

How Do I Delete a Shader or Program Object (and re-claim its memory)?

```
glDeleteObjectARB( GLhandleARB programOrShader );
```

How Do I Pass Information Into a Shader?

Uniform Variables, which cannot be set within a glBegin-glEnd:

```
float lightLoc[3] = { 0., 100., 0. };
location = glGetUniformLocationARB( program, "LightLocation" );
if( location < 0 )
	fprintf( stderr, "Cannot find Uniform variable 'LightLocation'\n" );

glUniform3fvARB( location, 3, lightLoc );
```

Per-vertex Attribute Variables, which can be set within a glBegin-glEnd:

```
location = glGetAttribLocationARB( program, "abArray" );
if( location < 0 )
      fprintf( stderr, "Cannot find Attribute variable 'abArray'\n" );

glBegin( GL_TRIANGLES );
      glVertexAttrib2fARB( location, a0, b0 );
      glVertex3f( x0, y0, z0 );
      glVertexAttrib2fARB( location, a1, b1 );
      glVertex3f( x1, y1, z1 );
      glVertexAttrib2fARB( location, a2, b2 );
      glVertex3f( x2, y2, z2 );
glEnd();
```

Note: Neither `glGetUniformLocationARB` **nor** `glGetAttribLocationARB` **can be called until after program is linked.**

How Do I Do Texture Mapping?

Determining the number of hardware texture units:

```
int numVertexTextureUnits, numFragmentTextureUnits;

glGetIntegerv( GL_MAX_TEXTURE_IMAGE_UNITS_ARB,        &numFragmentTextureUnits );
glGetIntegerv( GL_MAX_VERTEX_TEXTURE_IMAGE_UNITS_ARB, &numVertexTextureUnits   );
fprintf( stderr, "Have %d vertex texture units, and %d fragment texture units\n",
      numVertexTextureUnits, numFragmentTextureUnits );
```

(For reference, the NVIDIA Quadro FX 3400 has 4 vertex texture units and 16 fragment texture units.)

Setting up the texture in the first place:

```
int earthTex;
glGenTextures( 1, &earthTex );

glBindTexture( GL_TEXTURE_2D, earthTex );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR );
glTexParameteri( GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );
glTexImage2D( GL_TEXTURE_2D, 0, GL_RGB, w, h, 0, GL_RGB, GL_UNSIGNED_BYTE, rgb );
```

Binding the texture to a specific hardware texture unit:

```
Glint texLoc = glGetUniformLocationARB( program, "EarthTexture" );
glUniform1iARB( texLoc, 0 );        // use texture unit #0
```

Getting the shader to access the texture that is bound to a specific hardware texture unit:

```
uniform sampler2D EarthTexture;
void main(void)
{
      vec3 rgb = vec3( texture2D( EarthTexture, gl_TexCoord[0].st ) );
      gl_FragColor = vec4( rgb, 1.0 );
}
```

Drawing with this texture:

```
glActiveTexture( GL_TEXTURE0 );
glBindTexture( GL_TEXTURE_2D, earthTex );
glUseProgramObjectARB( program );
glEnable( GL_TEXTURE_2D );

glBegin( GL_QUADS );
      •••
glEnd();


glBegin( GL_TRIANGLE_STRIP );
      •••
glEnd();
```

How Do I Emulate Various Texture Environment Modes?

GL_REPLACE:

```
color = texture2D( tex, gl_TexCoord[0].st );
```

GL_MODULATE:

```
color *= texture2D( tex, gl_TexCoord[0].st );
```

GL_DECAL:

```
vec4 texture = texture2D( tex, gl_TexCoord[0].st );
vec3 col3 = mix( color.rgb, texture.rgb, texture.a );
color = vec4( col3, color.a );
```

GL_BLEND:

```
vec4 texture = texture2D( tex, gl_TexCoord[0].st );
vec3 col3 = mix( color.rgb, gl_TextureEnvColor[0].rgb, texture.rgb );
color = vec4( col3, color.a * texture.a );
```

GL_ADD:

```
vec4 texture = texture2D( tex, gl_TexCoord[0].st );
color.rgb += texture.rgb;
color.a *= texture.a;
color = clamp( color, 0., 1. );
```

Handy Built-in Functions

```
sin( radians );
cos( radians );
atan( y, x );

pow( x, toTheY );
log2( x );
sqrt( x );
inversesqrt( x );
abs( x );
sign( x );
fract( x );
mod( x, y );

min( x, y );
max( x, y );
clamp( x, min, max );
mix( x, y, t );
step( edge, x );
smoothstep( edge0, edge1, x );

length( x );
distance( p0, p1 );
dot( x, y );
cross( x, y );
normalize( x );
ftransform();
faceforward( N, I, Nref );
reflect( I, N );
```

How Do I Easily Check for OpenGL Errors?

(Note that this is not just for shaders – this can, and should, be used throughout your program!)

```
void
CheckGlErrors( const char* caller )
{
	unsigned int glerr = glGetError();

	if( glerr == GL_NO_ERROR )
		return;

	fprintf( stderr, "GL Error discovered from caller `%s`: ", caller );

	switch( glerr )
	{
		case GL_INVALID_ENUM:
			fprintf( stderr, "Invalid enum.\n" );
			break;

		case GL_INVALID_VALUE:
			fprintf( stderr, "Invalid value.\n" );
			break;

		case GL_INVALID_OPERATION:
			fprintf( stderr, "Invalid Operation.\n" );
			break;

		case GL_STACK_OVERFLOW:
```

```
                        fprintf( stderr, "Stack overflow.\n" );
                        break;

                case GL_STACK_UNDERFLOW:
                        fprintf(stderr, "Stack underflow.\n" );
                        break;

                case GL_OUT_OF_MEMORY:
                        fprintf( stderr, "Out of memory.\n" );
                        break;

                default:
                        fprintf( stderr, "Unknown OpenGL error: %d (0x%0x)\n",
                                glerr, glerr );
        }
}
```

How Do I Tell If the GLSL Shader Extensions are Included?

```
fprintf( stderr, "Version = '%s'\n", glGetString( GL_VERSION ) );

char *list = (char *) glGetString( GL_EXTENSIONS );

if( strstr( list, "GL_ARB_shader_objects " ) != NULL )
        fprintf( stderr, "Has the GL_ARB_shader_objects Extension\n" );
else
        fprintf( stderr, "Does not have the GL_ARB_shader_objects Extension\n" );
```

How Do I Make the GLSL Extensions Work?

This defines what extensions are available:

```
#include <GL/glext.h>
```

How to tell if glext.h thinks the GLSL extensions are available:

```
#ifdef GL_ARB_shader_objects
```

Generated by extgen.cpp – don't modify, just include:

```
#include <GL/glh_genext.h>
```

Include this near the top of the program – it declares the extension procedures:

```
#include "glsl_procdefines.h"
```

Include this after each window has been created – it assigned the procedures to a rendering context:

```
#include "glsl_procassigns.h"
```

What's Really Behind the OpenGL Extensions?

#include <GL/glext.h> :

```
#define GL_ARB_shader_objects 1

extern GLhandleARB APIENTRY glCREATESHADEROBJECTARBPROC) (GLenum shaderType);

extern GLhandleARB APIENTRY glCreateShaderObjectARB (GLenum);
```

#include <GL/glh_genext.h> :

```
GLH_EXTERN PFNGLCREATESHADEROBJECTARBPROC GLH_EXT_NAME(glCreateShaderObjectARB) GLH_INITIALIZER;
```

#include "glsl_procdefines.h" :

```
PFNGLCREATESHADEROBJECTARBPROC glCreateShaderObjectARB;
```

#include "glsl_procassigns.h" :

```
glCreateShaderObjectARB = (PFNGLCREATESHADEROBJECTARBPROC) wglGetProcAddress( "glCreateShaderObjectARB" );
```

Shader-in-a-Box

At the top of the program:

```
#include <windows.h>
#include <GL/glu.h>
#include <GL/gl.h>
#include <GL/glut.h>
#include "glui.h"

#include <GL/glh_genext.h>
#include "glsl_procdefines.h"
```

In the function prototype declarations:

```
void            CheckGlErrors( const char* );
GLhandleARB     LoadVertexShader( const char *, GLhandleARB );
GLhandleARB     LoadFragmentShader( const char *, GLhandleARB );
int             LoadShader( const char *, GLhandleARB, GLhandleARB );
int             LinkProgram( GLhandleARB, GLhandleARB, GLhandleARB );
```

In the global variable declarations:

```
GLhandleARB     VertShader;
GLhandleARB     FragmentShader;
GLhandleARB     Program;
```

In InitGraphics(), after each window has been created:

```
#include "glsl_procassigns.h"


Program = glCreateProgramObjectARB();
CheckGlErrors( "InitGraphics: glCreateProgramObjectARB" );

VertShader = LoadVertexShader( "???.vert", Program );
CheckGlErrors( "InitGraphics: LoadVertexShader" );

FragmentShader = LoadFragmentShader( "???.frag", Program );
CheckGlErrors( "InitGraphics: LoadFragmentShader" );

LinkProgram( Program, VertShader, FragmentShader );
CheckGlErrors( "InitGraphics: LinkProgram" );
```

Before drawing vertices:

```
glUseProgramObjectARB( Program );
```

Routine source:

```
#include "glsl_support.h"
```