



GLSL Primer (for version 3.3)

CS334

Daniel G. Aliaga
Department of Computer Science
Purdue University

[Slides thanks to Ed Angel &
Dave Shreiner]



GLSL Data Types

- Scalar types: `float, int, bool`
- Vector types: `vec2, vec3, vec4`
`ivec2, ivec3, ivec4`
`bvec2, bvec3, bvec4`
- Matrix types: `mat2, mat3, mat4`
- Texture sampling: `sampler1D, sampler2D,`
`sampler3D, samplerCube`
- C++ Style Constructors
`vec3 a = vec3(1.0, 2.0, 3.0);`
`mat4 b = mat4(5.0); // fill the diagonal with 5.0`



Operators

- Standard C/C++ arithmetic and logic operators
- Overloaded operators for matrix and vector operations

```
mat4 m;  
vec4 a, b, c;
```

```
b = a*m;  
c = m*a;
```



Components and Swizzling

- Access vector components using either:
 - `[]` (c-style array indexing)
 - `xyzw`, `rgba` or `strq` (named components)
- For example:

```
vec3 v;  
v[1], v.y, v.g, v.t
```

 - all refer to the same element
- Component swizzling:

```
vec3 a, b;  
a.xy = b.yx;
```



Qualifiers

- `in, out`

- Copy vertex attributes and other variable into and out of shaders

```
in  vec2 texCoord;
```

```
out vec4 color;
```

- `uniform`

- shader-constant variable from application

```
uniform float time;
```

```
uniform vec4 rotation;
```



Functions

- Built in
 - Arithmetic: `sqrt`, `pow`, `abs`
 - Trigonometric: `sin`, `asin`, `radians`
 - Graphical: `length`, `reflect`, `dot`, `normalize`
- User defined



Built-in Variables

- `gl_Position`
 - (required) output position from vertex shader
 - Receive homogeneous vertex position
- `gl_FragCoord`
 - input fragment position
- `gl_FragDepth`
 - input depth value in fragment shader

Simple Vertex Shader for Cube Example



```
#version 330 core

in vec4 vPosition;
in vec4 vColor;

out vec4 color;

void main()
{
    color = vColor;
    gl_Position = vPosition;
}
```


The Simplest Fragment Shader



```
#version 330 core

in vec4 color;

out vec4 fColor; // fragment's final color

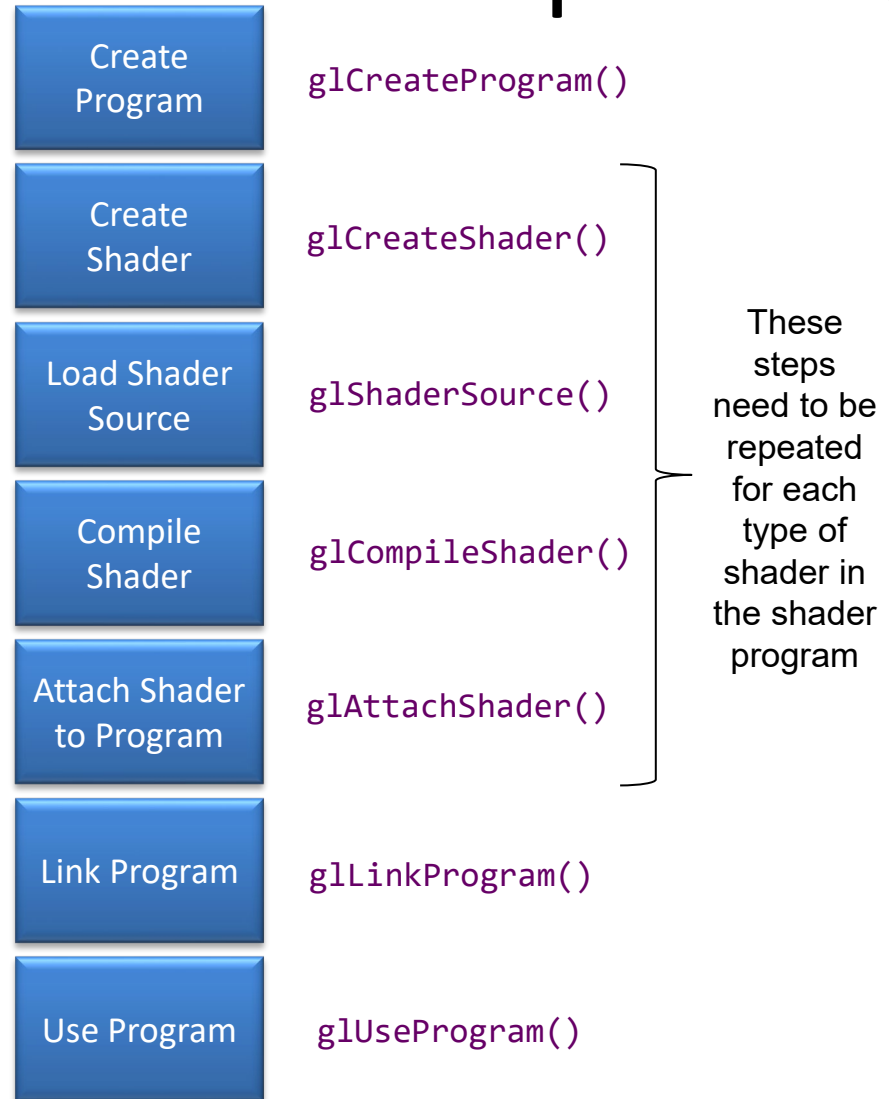
void main()
{
    fColor = color;
}
```

Note: `gl_FragColor` used to be the output color but that is deprecated in the latest version...



Getting Your Shaders into OpenGL

- Shaders need to be compiled and linked to form an executable shader program
- OpenGL provides the compiler and linker
- A program must contain
 - vertex and fragment shaders
 - other shaders are optional



Associating Shader Variables and Data



- Need to associate a shader variable with an OpenGL data source
 - vertex shader attributes → app vertex attributes
 - shader uniforms → app provided uniform values
- OpenGL relates shader variables to indices for the app to set
- Two methods for determining variable/index association
 - specify association before program linkage
 - query association after program linkage

Determining Locations After Linking



- Assumes you already know the variables' names

```
GLint loc = glGetUniformLocation(
program, "name" );
```

```
GLint loc = glGetUniformLocation(
program, "name" );
```



Initializing Uniform Variable Values

- Uniform Variables

```
glUniform4f( index, x, y, z, w );
```

```
GLboolean  transpose = GL_TRUE;
```

```
// Since we're C programmers
```

```
GLfloat  mat[3][4][4] = { ... };
```

```
glUniformMatrix4fv( index, 3, transpose,  
mat );
```



A Cube Program

```
int main( int argc, char **argv )
{
    glutInit( &argc, argv );
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH );
    glutInitWindowSize( 512, 512 );
    glutCreateWindow( "Color Cube" );

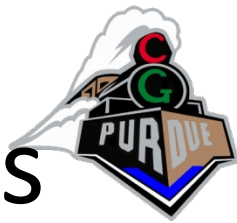
    glewInit();
    init();

    // Setup SHADERS
    // ...
    // ...

    glutDisplayFunc( display );
    glutKeyboardFunc( keyboard );
    glutMainLoop();

    return 0;
}
```

Cube Program's GLUT Callbacks



```
void display( void )
{
    glClear( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT );
    glDrawArrays( GL_TRIANGLES, 0, NumVertices );
    glutSwapBuffers();
}
```

```
void keyboard( unsigned char key, int x, int y )
{
    switch( key ) {
        case 033: case 'q': case 'Q':
            exit( EXIT_SUCCESS );
            break;
    }
}
```



Vertex Shader Examples

- A vertex shader is initiated by each vertex output by `glDrawArrays()`
- A vertex shader must output a position in clip coordinates to the rasterizer
- Basic uses of vertex shaders
 - Transformations
 - Lighting
 - Moving vertex positions



Demos

- <http://glslsandbox.com/>