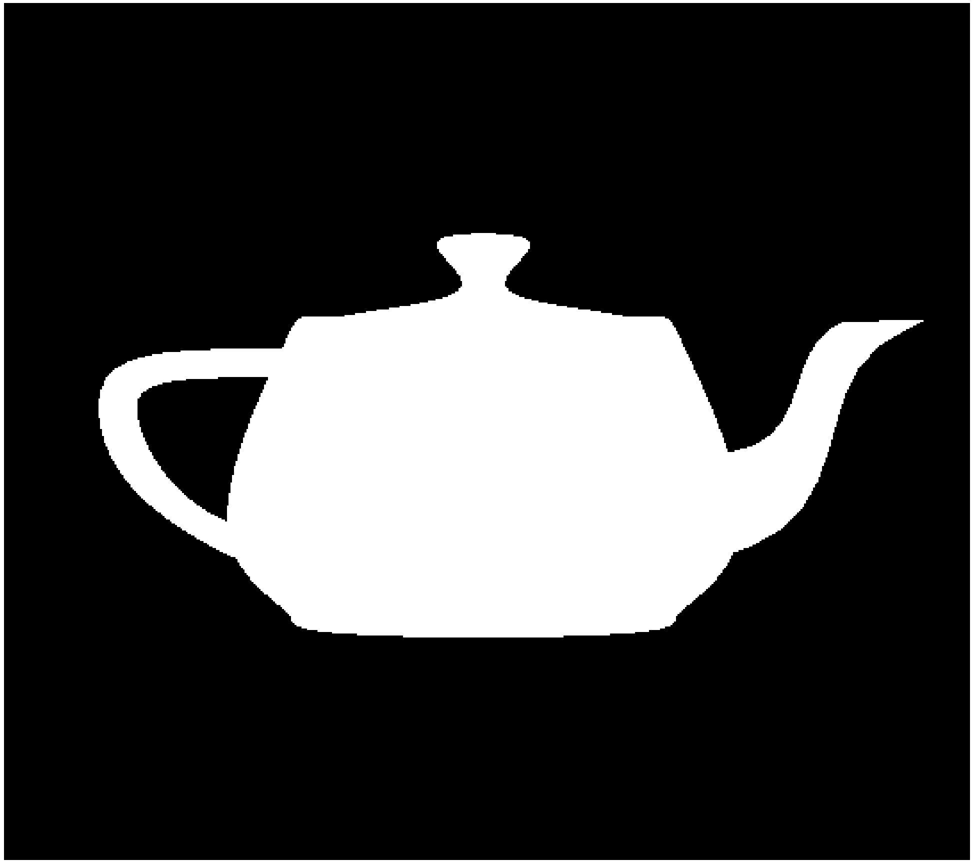
Computer Graphics

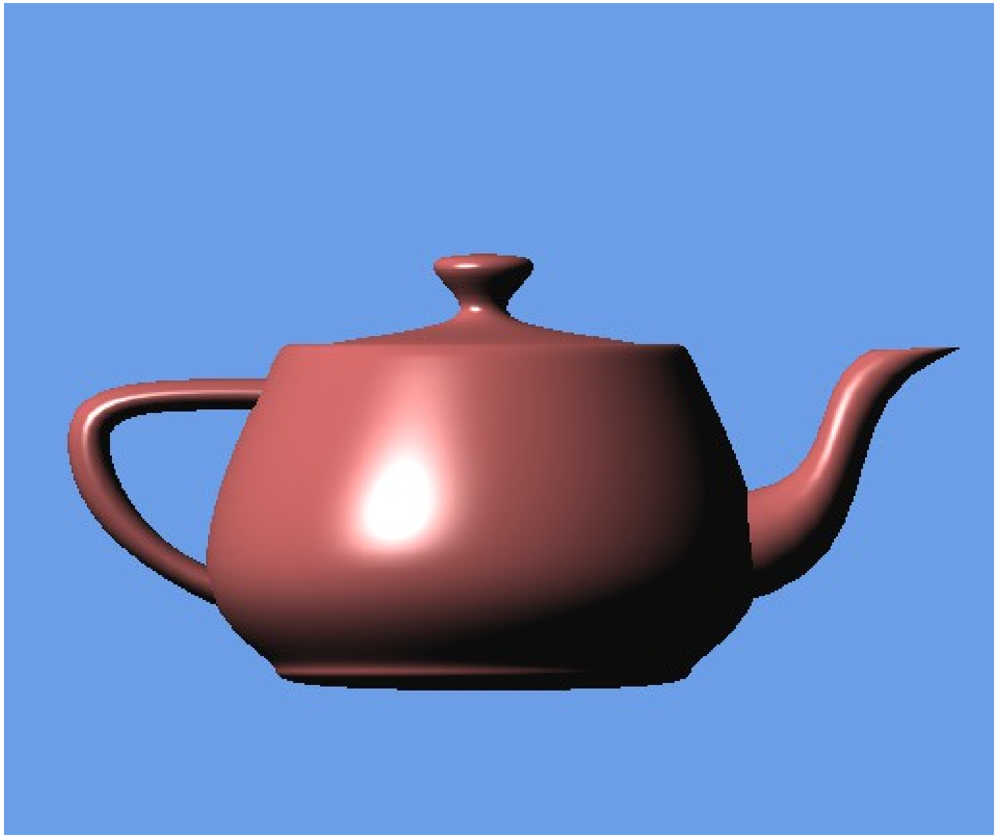
Assignment One

Objective

- Input

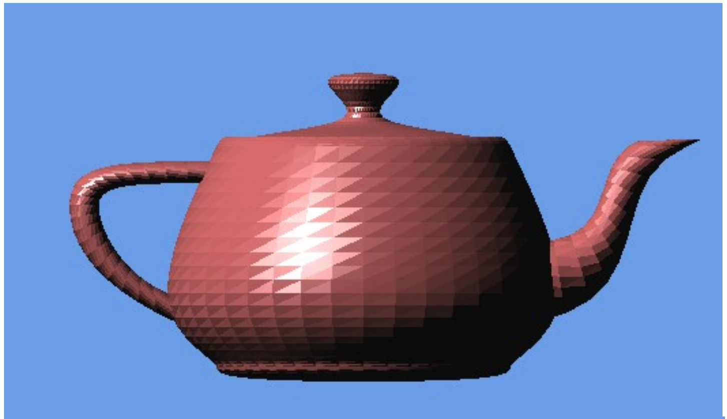


- Output

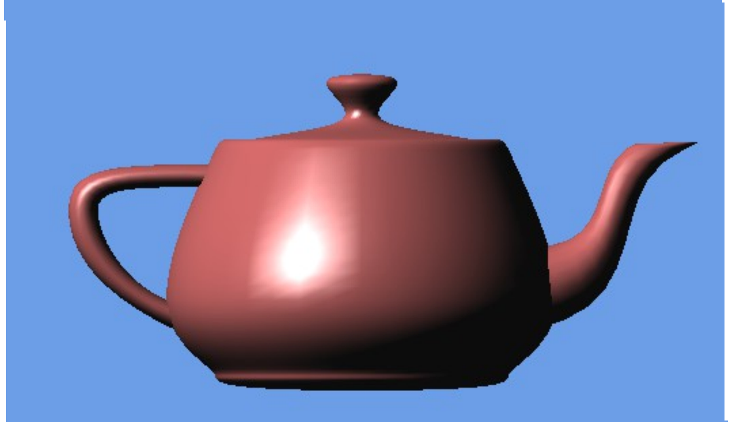


Objective

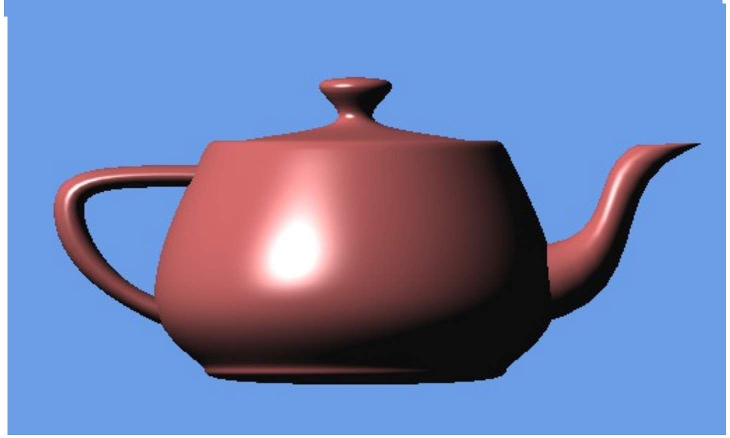
- Flat shading



- Gouraud shading

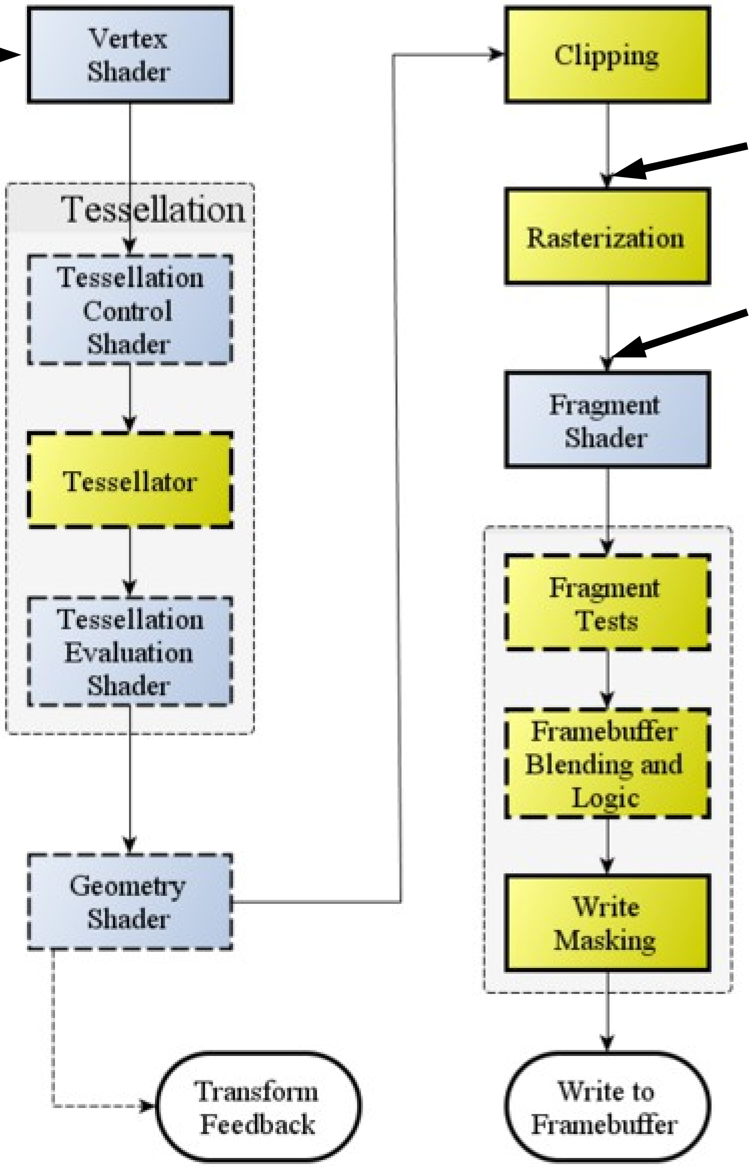


- Phong shading



OpenGL Pipeline

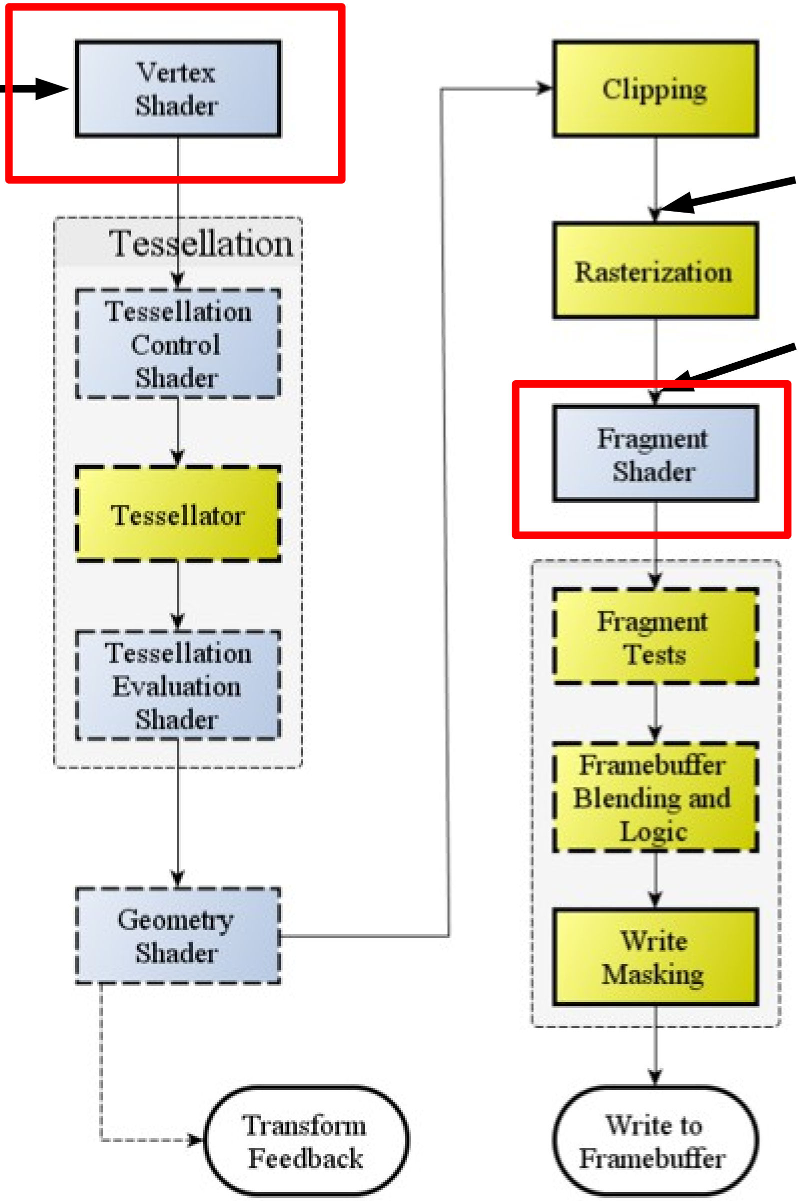
Vertex Data



Viewport transform to window space

Primitive parameter interpolation

GLSL Shaders

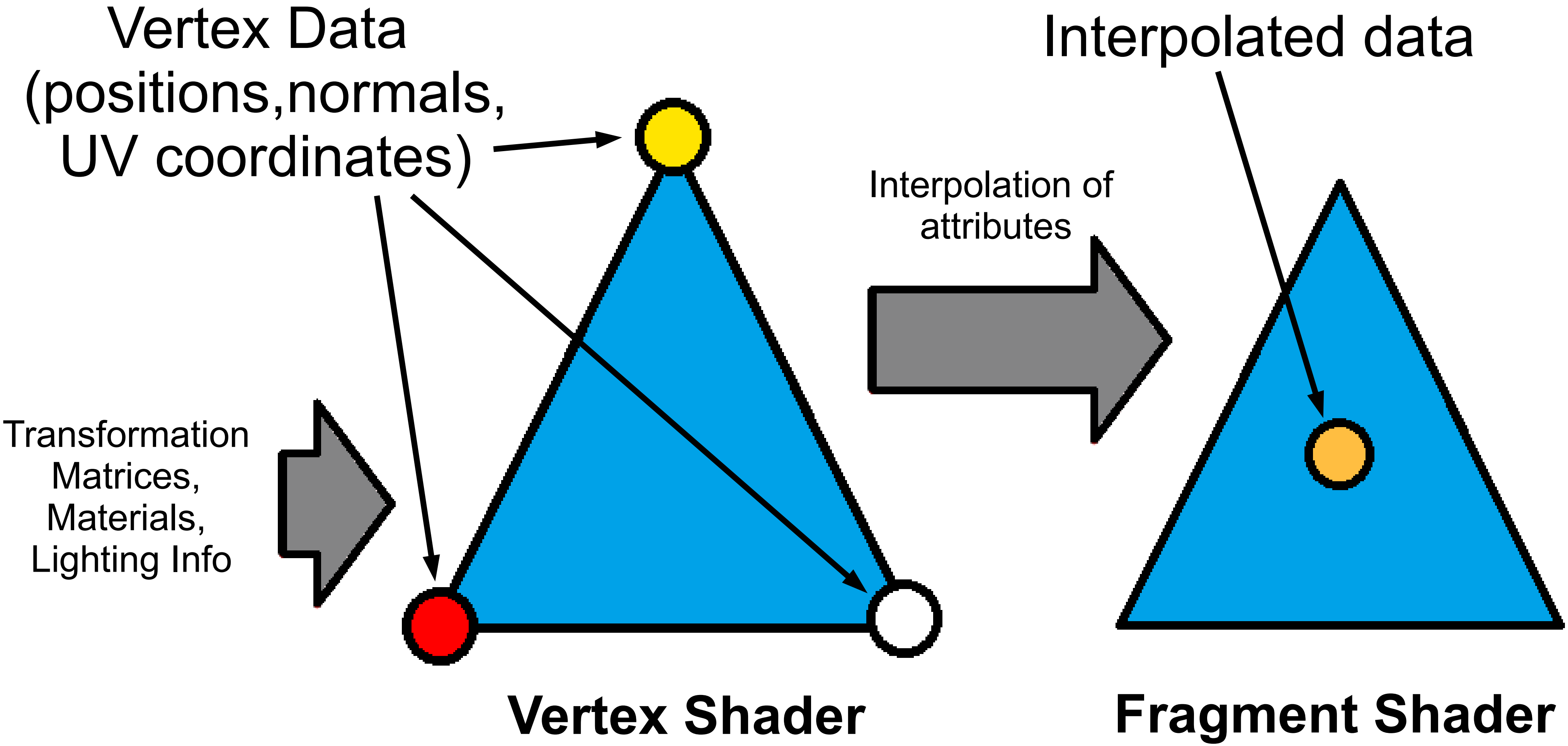


Vertex Data

Viewport transform to window space

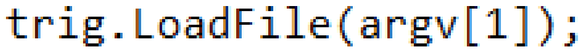
Primitive parameter interpolation

GLSL Shaders



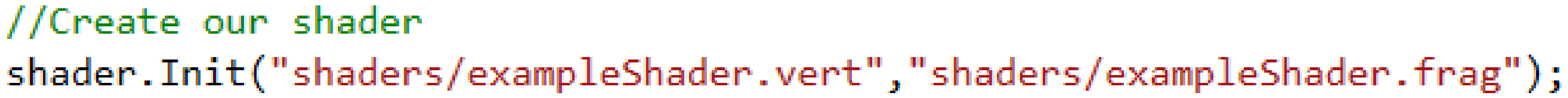
Source Code

- TriangleMesh class
  - Loads “teapot.obj”, holds mesh info



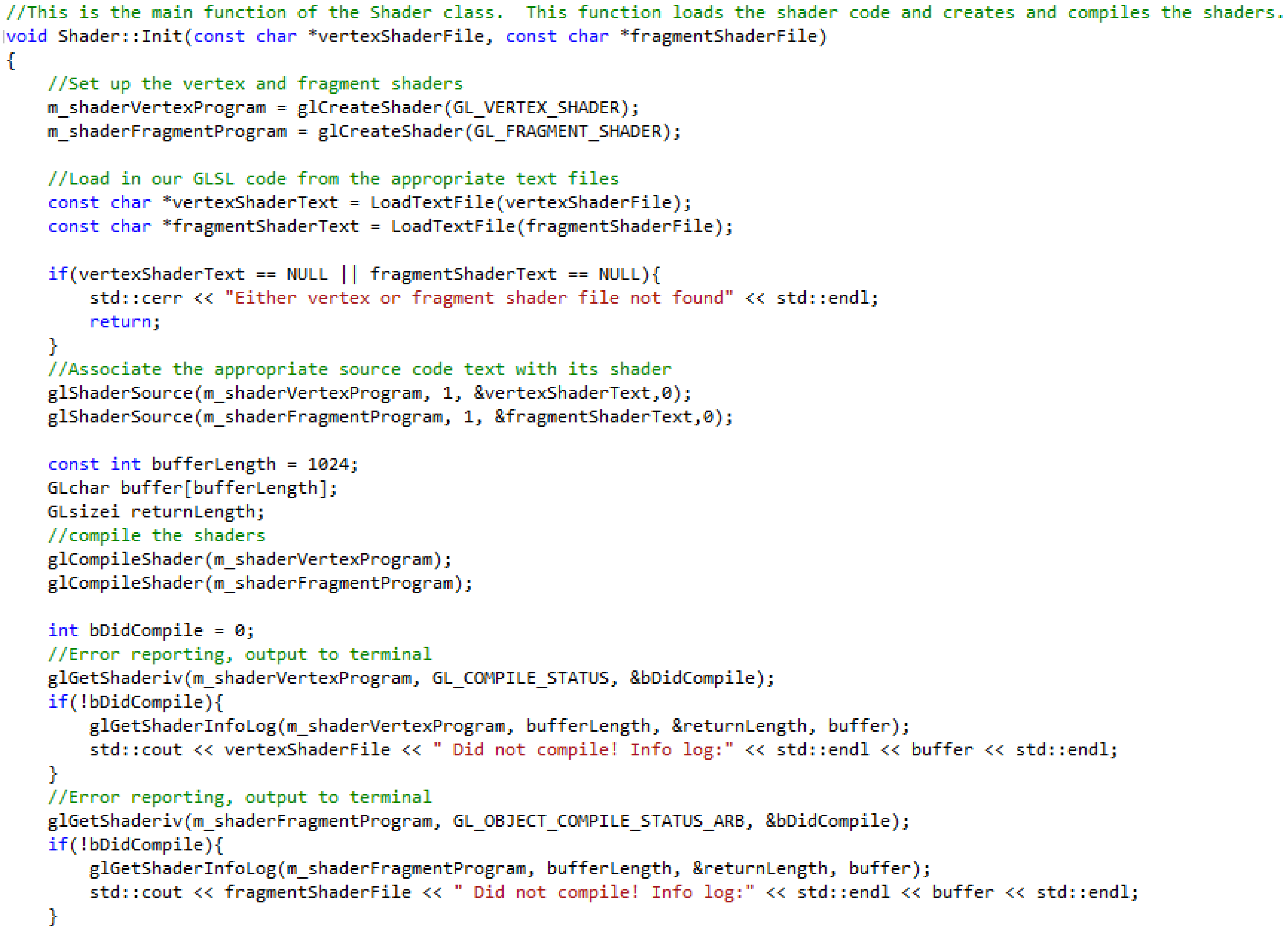
```
trig.LoadFile(argv[1]);
```

    - No Normal information, these must be computed
    - Use Triangles structures for this
    - Remember to normalise!
- Shader class
  - Init function loads vertex and fragment shader programs from text files:



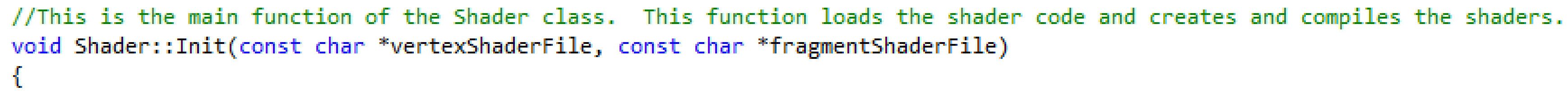
```
//Create our shader
shader.Init("shaders/exampleShader.vert","shaders/exampleShader.frag");
```

Source Code



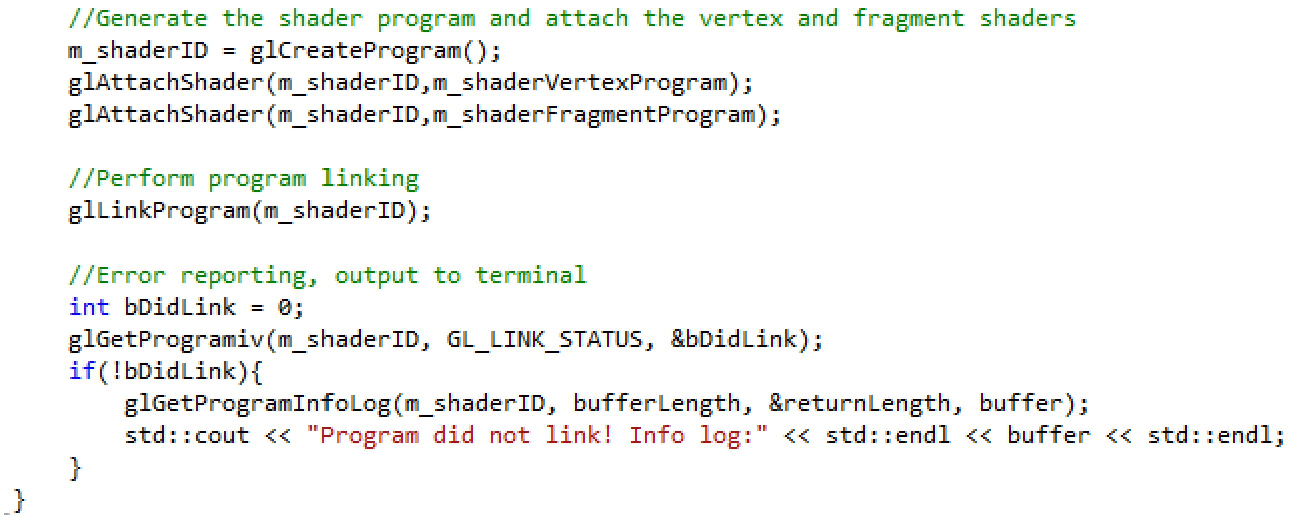
```
//This is the main function of the Shader class.  This function loads the shader code and creates and compiles the shaders.
void Shader::Init(const char *vertexShaderFile, const char *fragmentShaderFile)
{
    //Set up the vertex and fragment shaders
    m_shaderVertexProgram = glCreateShader(GL_VERTEX_SHADER);
    m_shaderFragmentProgram = glCreateShader(GL_FRAGMENT_SHADER);

    //Load in our GLSL code from the appropriate text files
    const char *vertexShaderText = LoadTextFile(vertexShaderFile);
    const char *fragmentShaderText = LoadTextFile(fragmentShaderFile);

    if(vertexShaderText == NULL || fragmentShaderText == NULL){
        std::cerr << "Either vertex or fragment shader file not found" << std::endl;
        return;
    }
    //Associate the appropriate source code text with its shader
    glShaderSource(m_shaderVertexProgram, 1, &vertexShaderText,0);
    glShaderSource(m_shaderFragmentProgram, 1, &fragmentShaderText,0);

    const int bufferLength = 1024;
    GLchar buffer[bufferLength];
    GLsizei returnLength;
    //compile the shaders
    glCompileShader(m_shaderVertexProgram);
    glCompileShader(m_shaderFragmentProgram);

    int bDidCompile = 0;
    //Error reporting, output to terminal
    glGetShaderiv(m_shaderVertexProgram, GL_COMPILE_STATUS, &bDidCompile);
    if(!bDidCompile){
        glGetShaderInfoLog(m_shaderVertexProgram, bufferLength, &returnLength, buffer);
        std::cout << vertexShaderFile << " Did not compile! Info log:" << std::endl << buffer << std::endl;
    }
    //Error reporting, output to terminal
    glGetShaderiv(m_shaderFragmentProgram, GL_OBJECT_COMPILE_STATUS_ARB, &bDidCompile);
    if(!bDidCompile){
        glGetShaderInfoLog(m_shaderFragmentProgram, bufferLength, &returnLength, buffer);
        std::cout << fragmentShaderFile << " Did not compile! Info log:" << std::endl << buffer << std::endl;
    }
```

Source Code



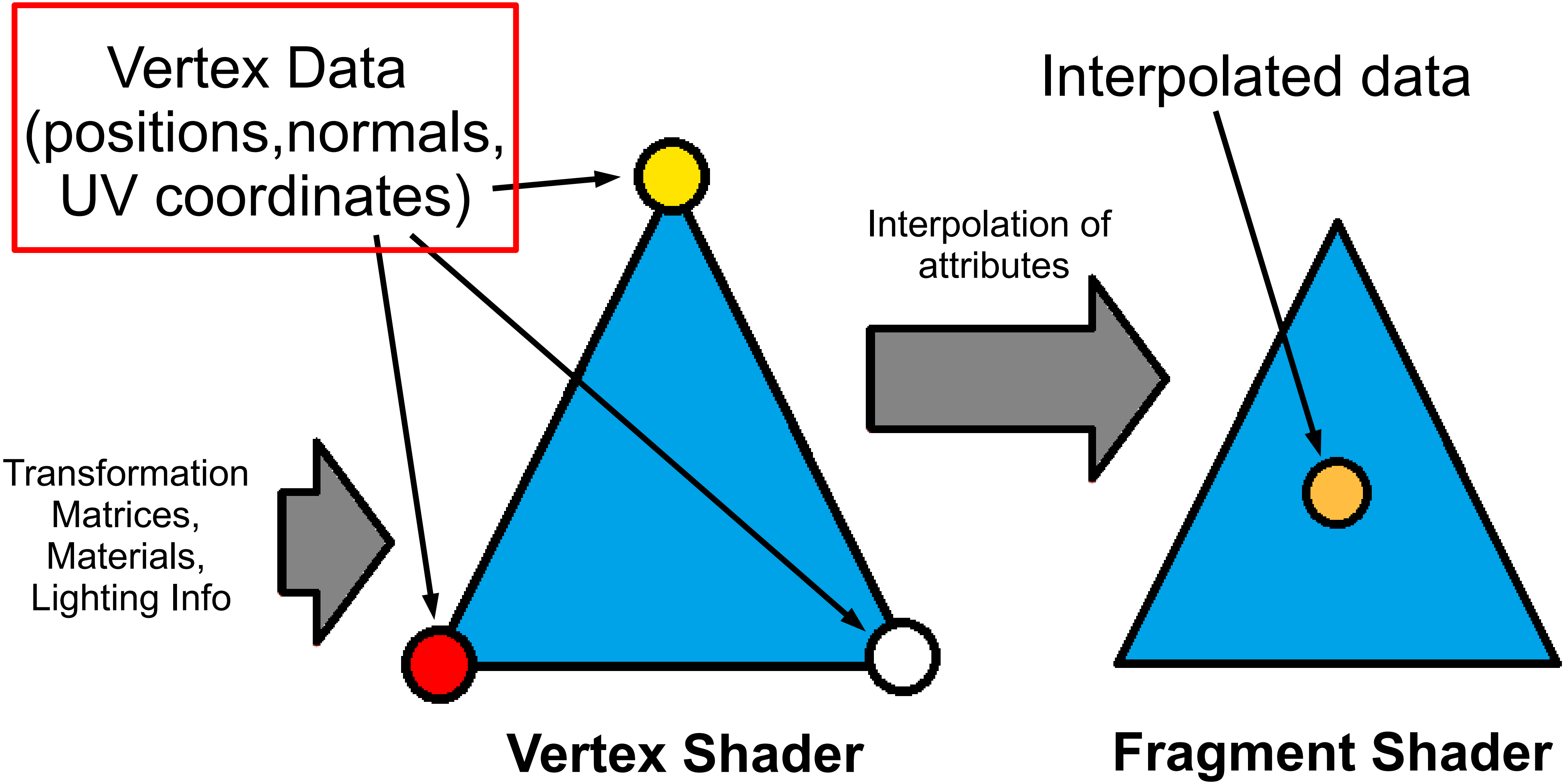
```
//This is the main function of the Shader class.  This function loads the shader code and creates and compiles the shaders.
void Shader::Init(const char *vertexShaderFile, const char *fragmentShaderFile)
{
```

....



```
    //Generate the shader program and attach the vertex and fragment shaders
    m_shaderID = glCreateProgram();
    glAttachShader(m_shaderID,m_shaderVertexProgram);
    glAttachShader(m_shaderID,m_shaderFragmentProgram);

    //Perform program linking
    glLinkProgram(m_shaderID);

    //Error reporting, output to terminal
    int bDidLink = 0;
    glGetProgramiv(m_shaderID, GL_LINK_STATUS, &bDidLink);
    if(!bDidLink){
        glGetProgramInfoLog(m_shaderID, bufferLength, &returnLength, buffer);
        std::cout << "Program did not link! Info log:" << std::endl << buffer << std::endl;
    }
}
```

GLSL Shaders



Vertex Data (positions,normals, UV coordinates)

Transformation Matrices, Materials, Lighting Info

Interpolation of attributes

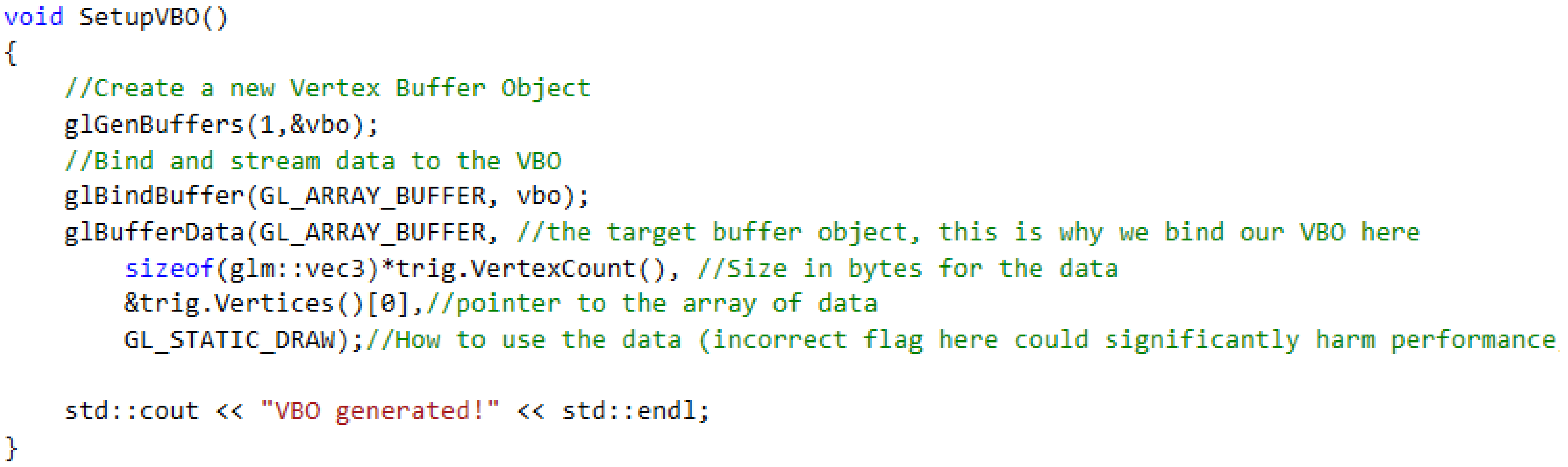
Interpolated data

Vertex Shader

Fragment Shader

Source Code

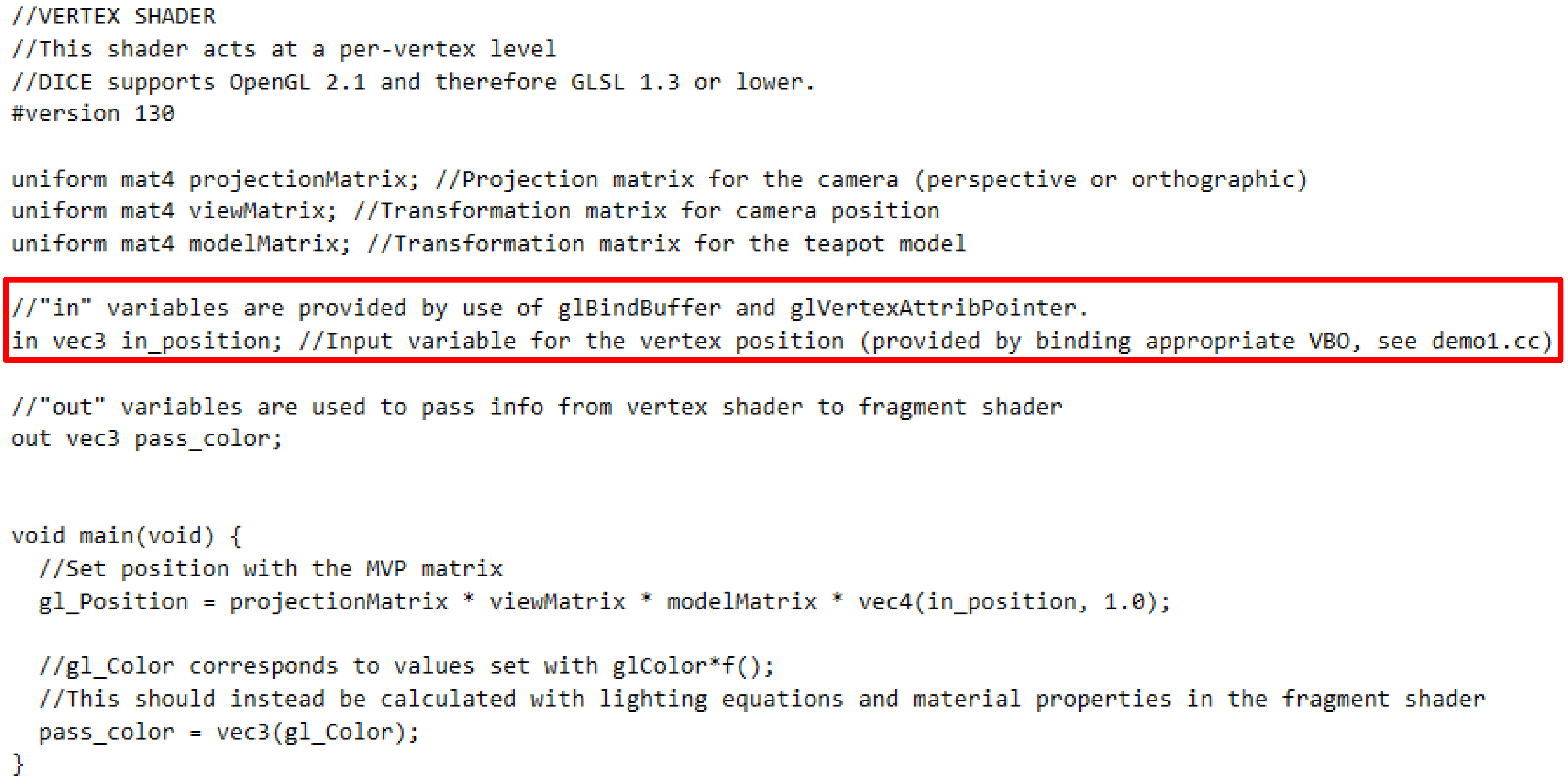
- Vertex Buffer Objects
  - Stores data that will be used in shader programs



```
void SetupVBO()
{
    //Create a new Vertex Buffer Object
    glGenBuffers(1,&vbo);
    //Bind and stream data to the VBO
    glBindBuffer(GL_ARRAY_BUFFER, vbo);
    glBufferData(GL_ARRAY_BUFFER, //the target buffer object, this is why we bind our VBO here
        sizeof(glm::vec3)*trig.VertexCount(), //Size in bytes for the data
        &trig.Vertices()[0],//pointer to the array of data
        GL_STATIC_DRAW);//How to use the data (incorrect flag here could significantly harm performance

    std::cout << "VBO generated!" << std::endl;
}
```

Source Code

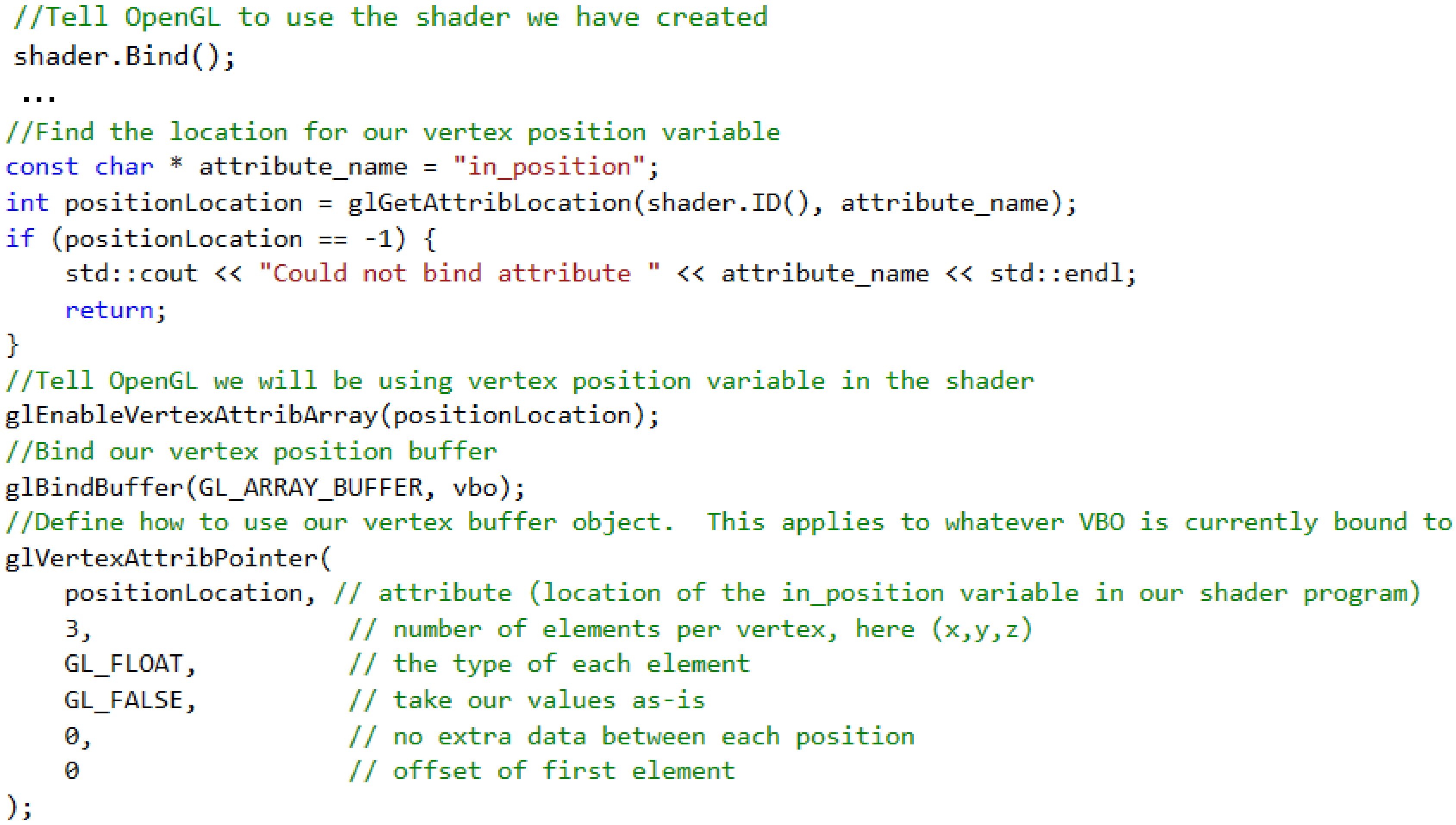
- “exampleShader.vert”



```
//VERTEX SHADER
//This shader acts at a per-vertex level
//DICE supports OpenGL 2.1 and therefore GLSL 1.3 or lower.
#version 130

uniform mat4 projectionMatrix; //Projection matrix for the camera (perspective or orthographic)
uniform mat4 viewMatrix; //Transformation matrix for camera position
uniform mat4 modelMatrix; //Transformation matrix for the teapot model

//"in" variables are provided by use of glBindBuffer and glVertexAttribPointer.
in vec3 in_position; //Input variable for the vertex position (provided by binding appropriate VBO, see demo1.cc)

//"out" variables are used to pass info from vertex shader to fragment shader
out vec3 pass_color;


void main(void) {
  //Set position with the MVP matrix
  gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(in_position, 1.0);

  //gl_Color corresponds to values set with glColor*f();
  //This should instead be calculated with lighting equations and material properties in the fragment shader
  pass_color = vec3(gl_Color);
}
```

Source Code

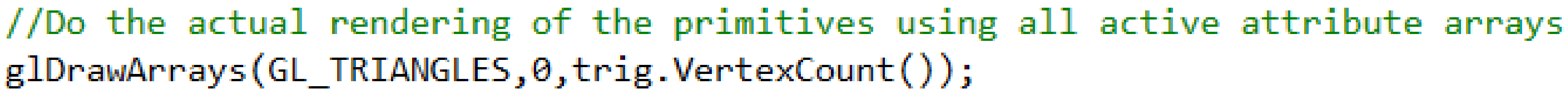
- “demo1.cpp” - DemoDisplay()
  - Setting “in” variables



```
//Tell OpenGL to use the shader we have created
shader.Bind();
...
//Find the location for our vertex position variable
const char * attribute_name = "in_position";
int positionLocation = glGetAttribLocation(shader.ID(), attribute_name);
if (positionLocation == -1) {
    std::cout << "Could not bind attribute " << attribute_name << std::endl;
    return;
}
//Tell OpenGL we will be using vertex position variable in the shader
glEnableVertexAttribArray(positionLocation);
//Bind our vertex position buffer
glBindBuffer(GL_ARRAY_BUFFER, vbo);
//Define how to use our vertex buffer object.  This applies to whatever VBO is currently bound to
glVertexAttribPointer(
    positionLocation, // attribute (location of the in_position variable in our shader program)
    3,                 // number of elements per vertex, here (x,y,z)
    GL_FLOAT,          // the type of each element
    GL_FALSE,          // take our values as-is
    0,                 // no extra data between each position
    0                  // offset of first element
);
```

Source Code

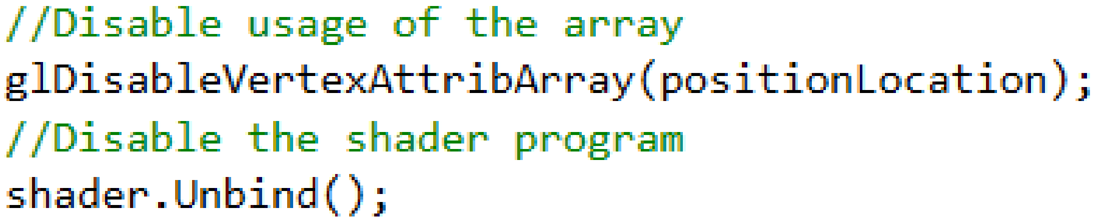
- “demo1.cpp” - DemoDisplay()
  - Rendering call



```
//Do the actual rendering of the primitives using all active attribute arrays
glDrawArrays(GL_TRIANGLES,0,trig.VertexCount());
```

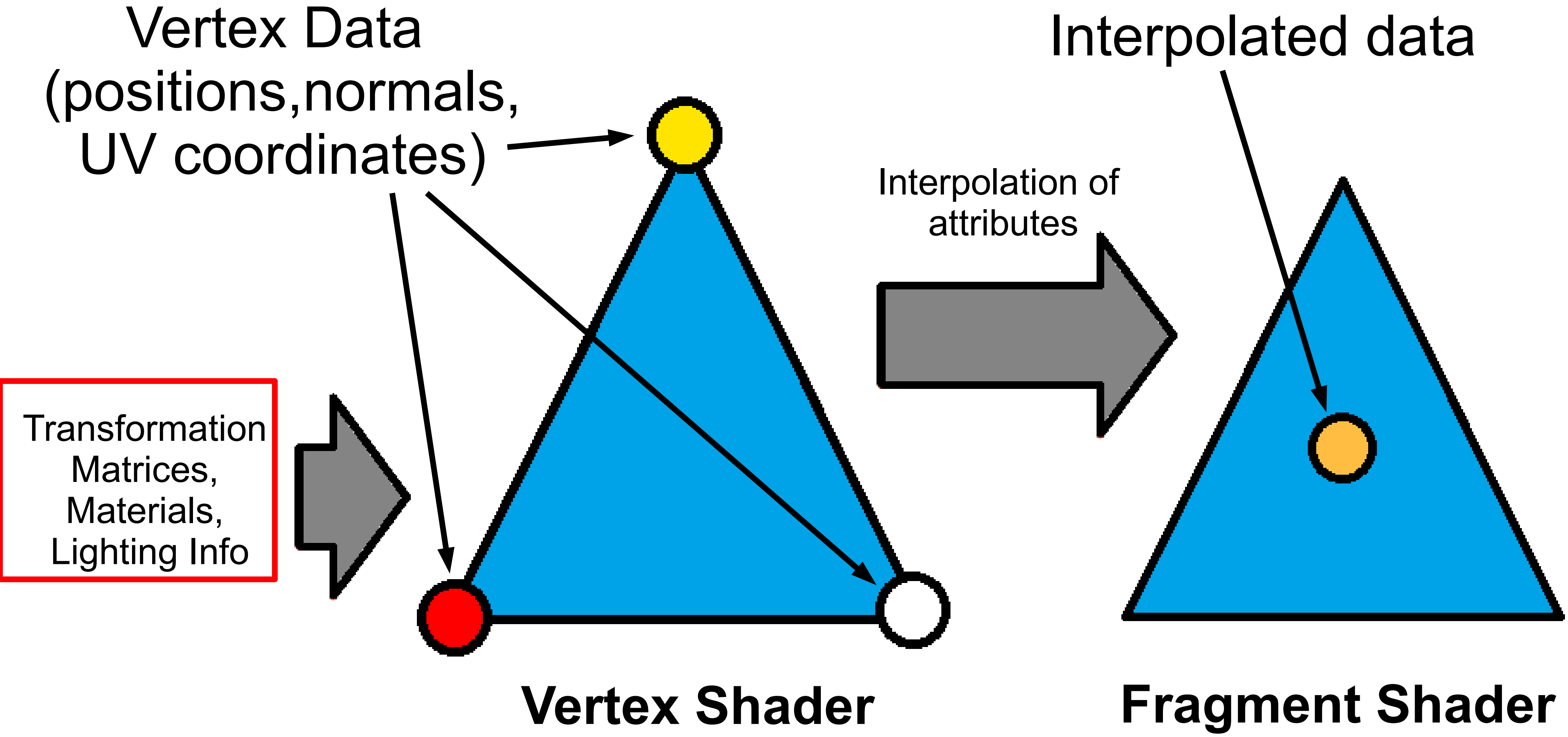
    - Alternative: glDrawElements (requires indexing)

  - Cleanup
    - Disable all arrays



```
//Disable usage of the array
glDisableVertexAttribArray(positionLocation);
//Disable the shader program
shader.Unbind();
```

GLSL Shaders



Vertex Data (positions,normals, UV coordinates)

Interpolated data

Interpolation of attributes

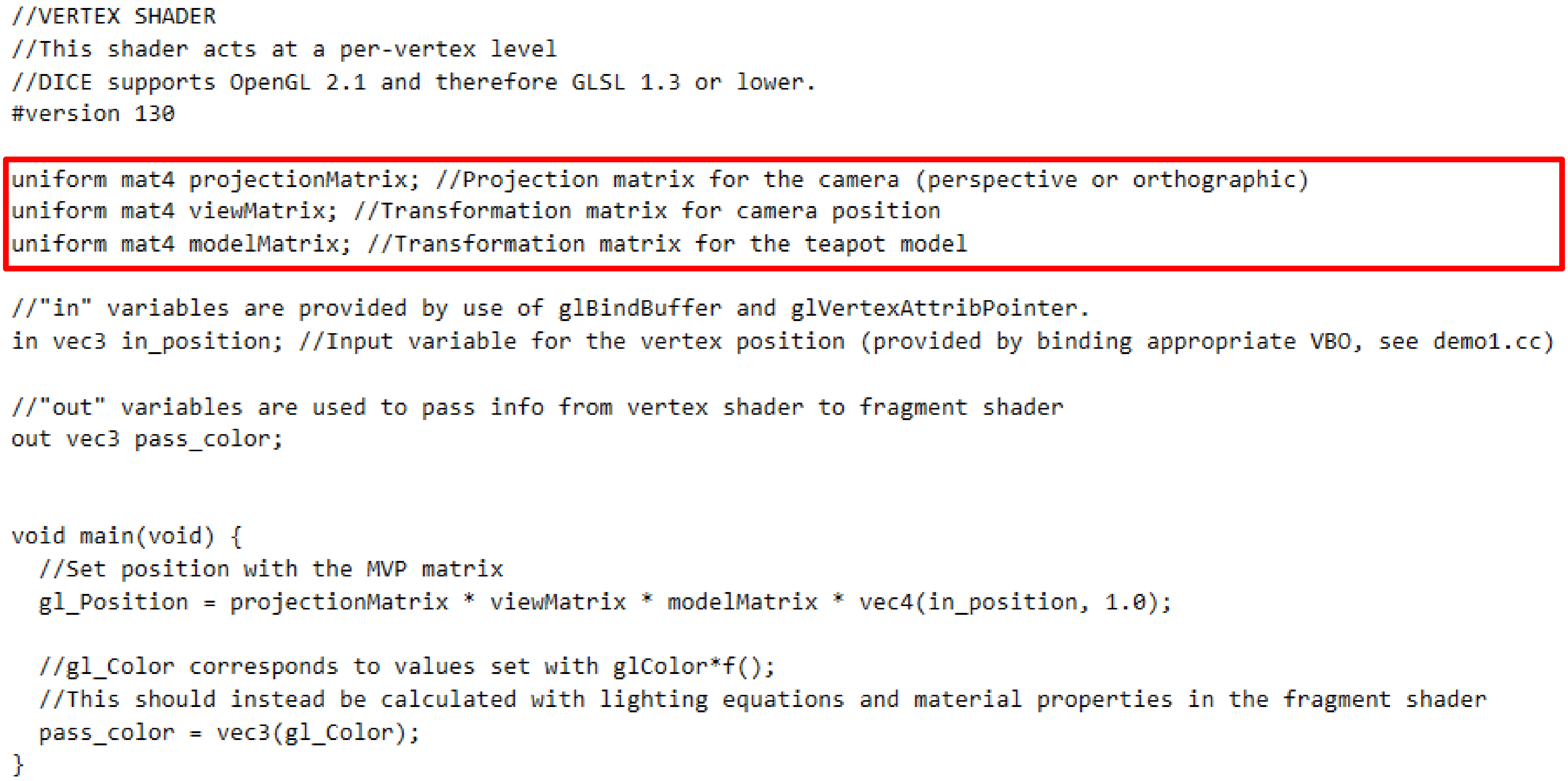
Transformation Matrices, Materials, Lighting Info

Vertex Shader

Fragment Shader

Source Code

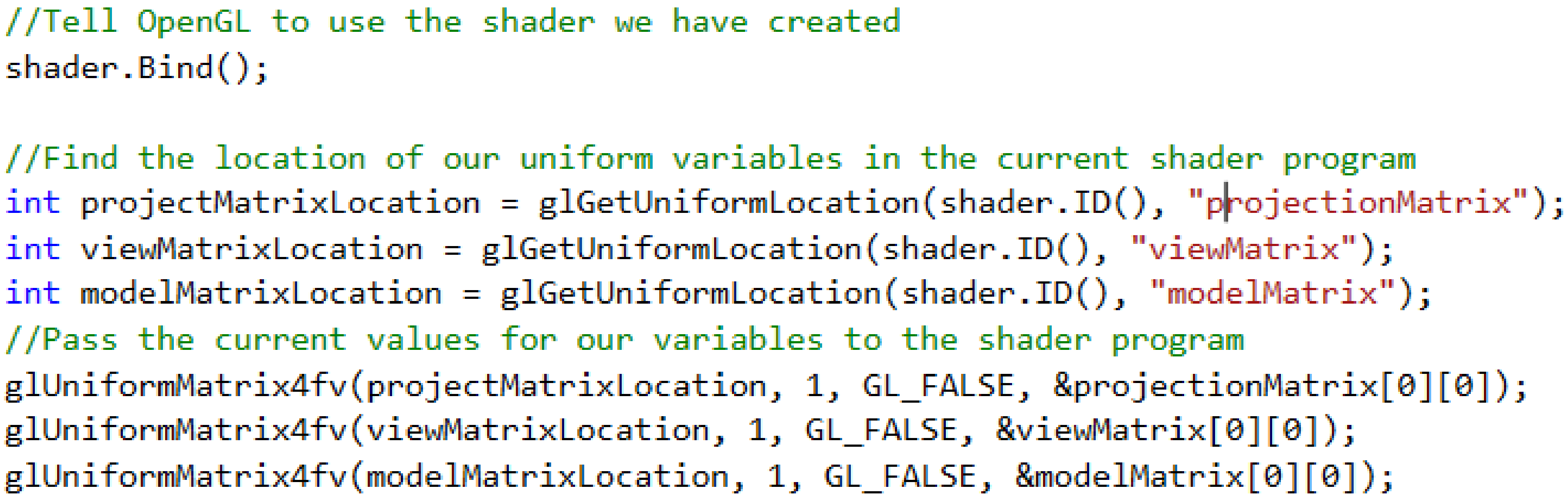
- “exampleShader.vert”



```
//VERTEX SHADER
//This shader acts at a per-vertex level
//DICE supports OpenGL 2.1 and therefore GLSL 1.3 or lower.
#version 130

uniform mat4 projectionMatrix; //Projection matrix for the camera (perspective or orthographic)
uniform mat4 viewMatrix; //Transformation matrix for camera position
uniform mat4 modelMatrix; //Transformation matrix for the teapot model

//"in" variables are provided by use of glBindBuffer and glVertexAttribPointer.
in vec3 in_position; //Input variable for the vertex position (provided by binding appropriate VBO, see demo1.cc)

//"out" variables are used to pass info from vertex shader to fragment shader
out vec3 pass_color;


void main(void) {
  //Set position with the MVP matrix
  gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(in_position, 1.0);

  //gl_Color corresponds to values set with glColor*f();
  //This should instead be calculated with lighting equations and material properties in the fragment shader
  pass_color = vec3(gl_Color);
}
```

Source Code

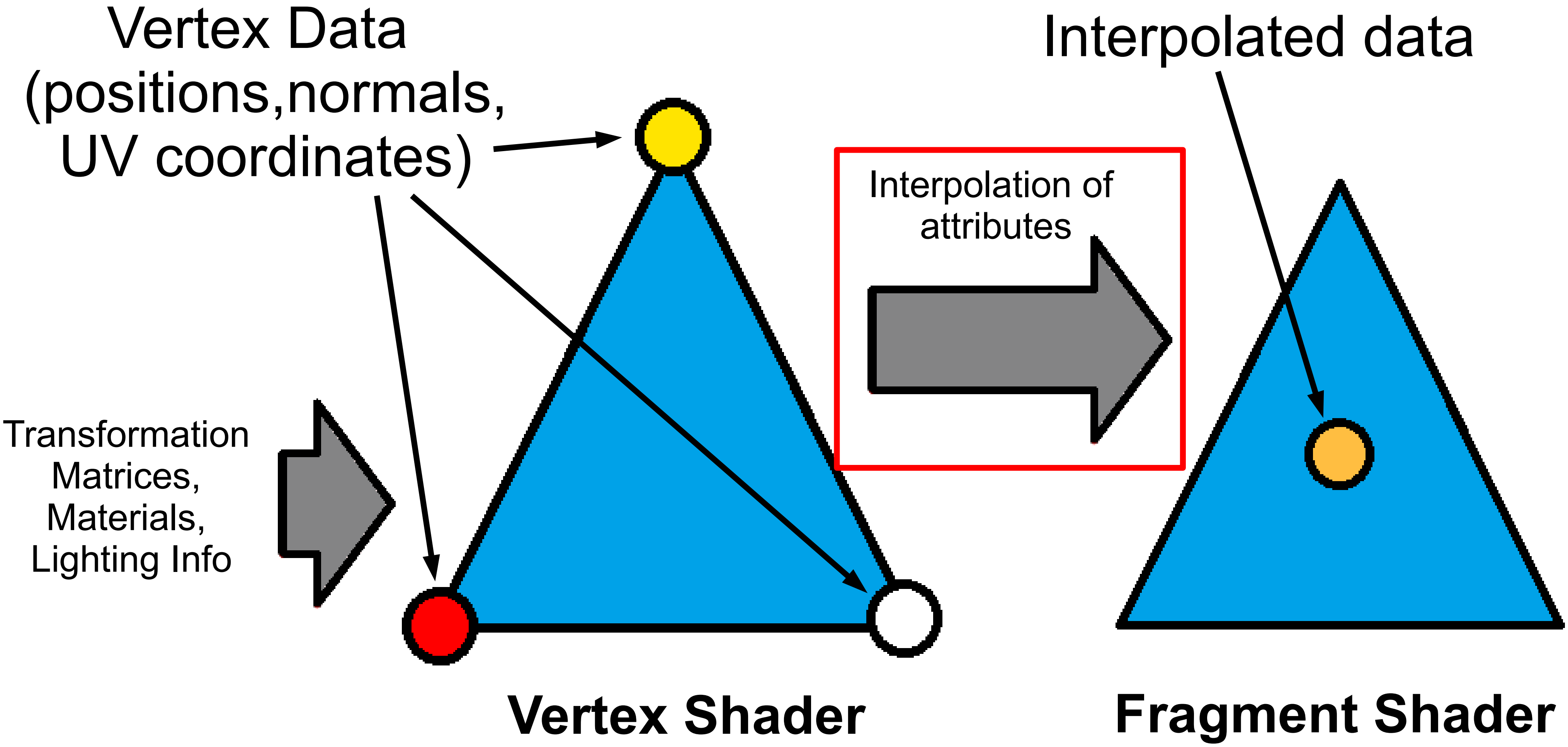
- “demo1.cpp” - DemoDisplay()
  - Setting Uniform Variables



```
//Tell OpenGL to use the shader we have created
shader.Bind();

//Find the location of our uniform variables in the current shader program
int projectMatrixLocation = glGetUniformLocation(shader.ID(), "projectionMatrix");
int viewMatrixLocation = glGetUniformLocation(shader.ID(), "viewMatrix");
int modelMatrixLocation = glGetUniformLocation(shader.ID(), "modelMatrix");
//Pass the current values for our variables to the shader program
glUniformMatrix4fv(projectMatrixLocation, 1, GL_FALSE, &projectionMatrix[0][0]);
glUniformMatrix4fv(viewMatrixLocation, 1, GL_FALSE, &viewMatrix[0][0]);
glUniformMatrix4fv(modelMatrixLocation, 1, GL_FALSE, &modelMatrix[0][0]);
```

  - Different “glUniform” methods for different variables

GLSL Shaders



Vertex Data
(positions,normals,
UV coordinates)

Interpolated data

Interpolation of
attributes

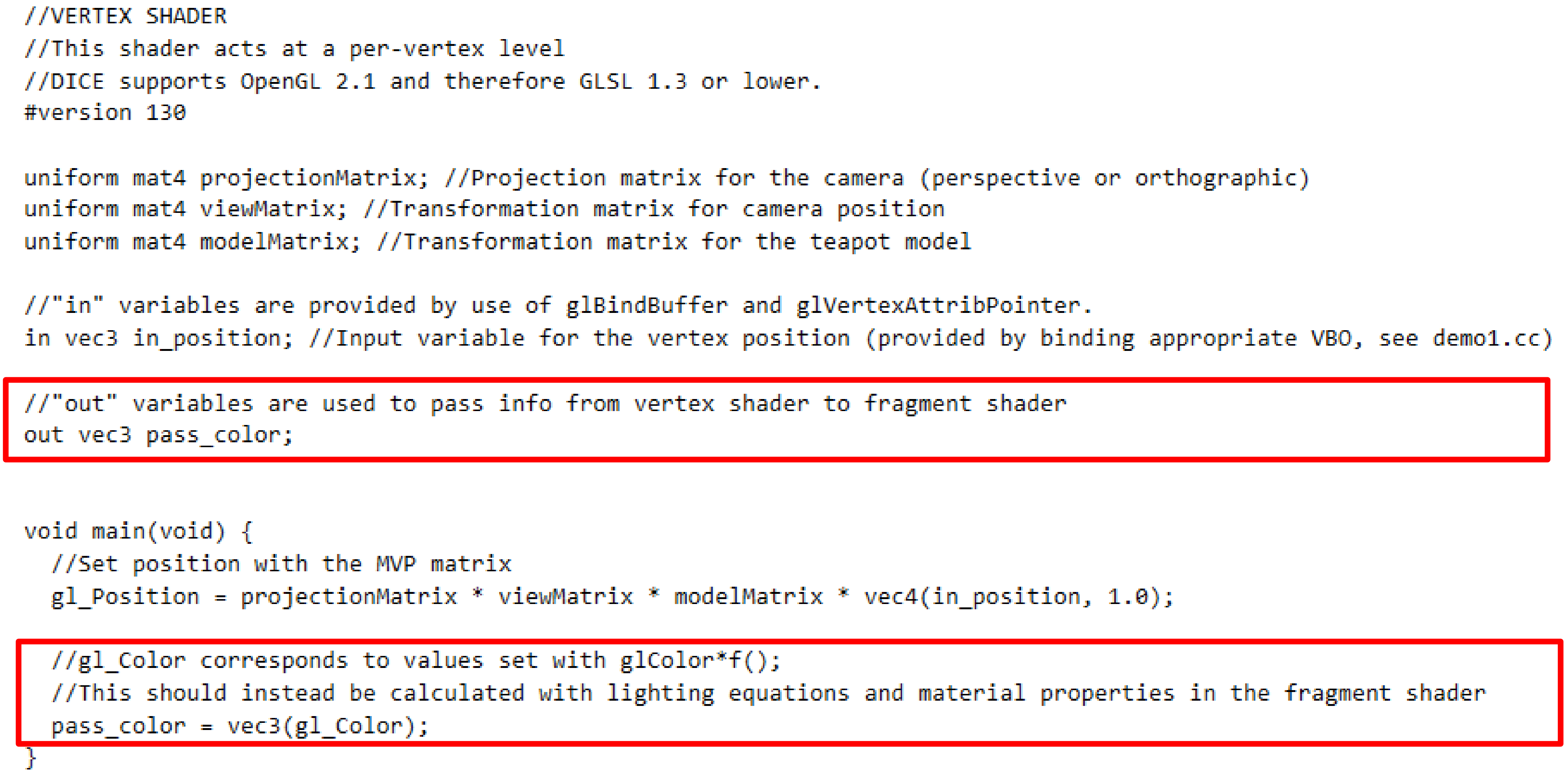
Transformation
Matrices,
Materials,
Lighting Info

Vertex Shader

Fragment Shader

Source Code

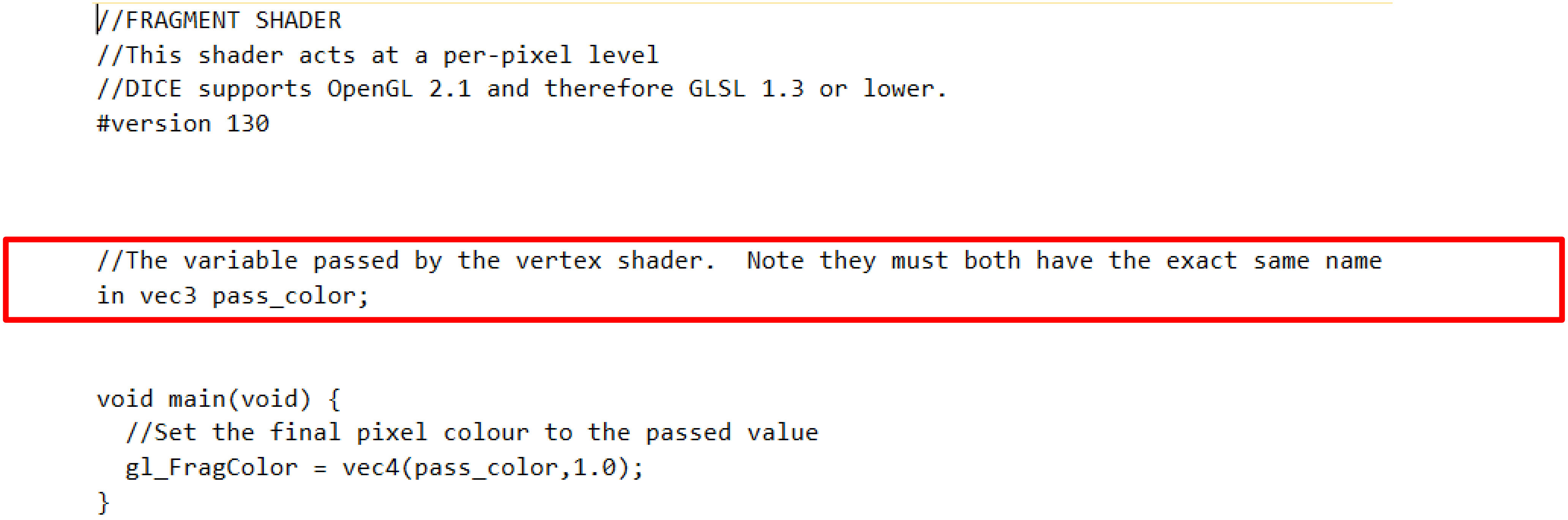
- “exampleShader.vert”



```
//VERTEX SHADER
//This shader acts at a per-vertex level
//DICE supports OpenGL 2.1 and therefore GLSL 1.3 or lower.
#version 130

uniform mat4 projectionMatrix; //Projection matrix for the camera (perspective or orthographic)
uniform mat4 viewMatrix; //Transformation matrix for camera position
uniform mat4 modelMatrix; //Transformation matrix for the teapot model

//"in" variables are provided by use of glBindBuffer and glVertexAttribPointer.
in vec3 in_position; //Input variable for the vertex position (provided by binding appropriate VBO, see demo1.cc)

//"out" variables are used to pass info from vertex shader to fragment shader
out vec3 pass_color;


void main(void) {
  //Set position with the MVP matrix
  gl_Position = projectionMatrix * viewMatrix * modelMatrix * vec4(in_position, 1.0);

  //gl_Color corresponds to values set with glColor*f();
  //This should instead be calculated with lighting equations and material properties in the fragment shader
  pass_color = vec3(gl_Color);
}
```

Source Code

- “ExampleShader.frag”
  - Values are interpolated between vertex and fragment shaders



```
//FRAGMENT SHADER
//This shader acts at a per-pixel level
//DICE supports OpenGL 2.1 and therefore GLSL 1.3 or lower.
#version 130



//The variable passed by the vertex shader.  Note they must both have the exact same name
in vec3 pass_color;


void main(void) {
  //Set the final pixel colour to the passed value
  gl_FragColor = vec4(pass_color,1.0);
}
```

Adding Lighting and Materials

- Either
  - Add with OpenGL commands
    - GlLightfv(...)
    - gl_LightSource structure in shader program
    - glMaterialfv(...)
    - gl_FrontMaterial structure in shader

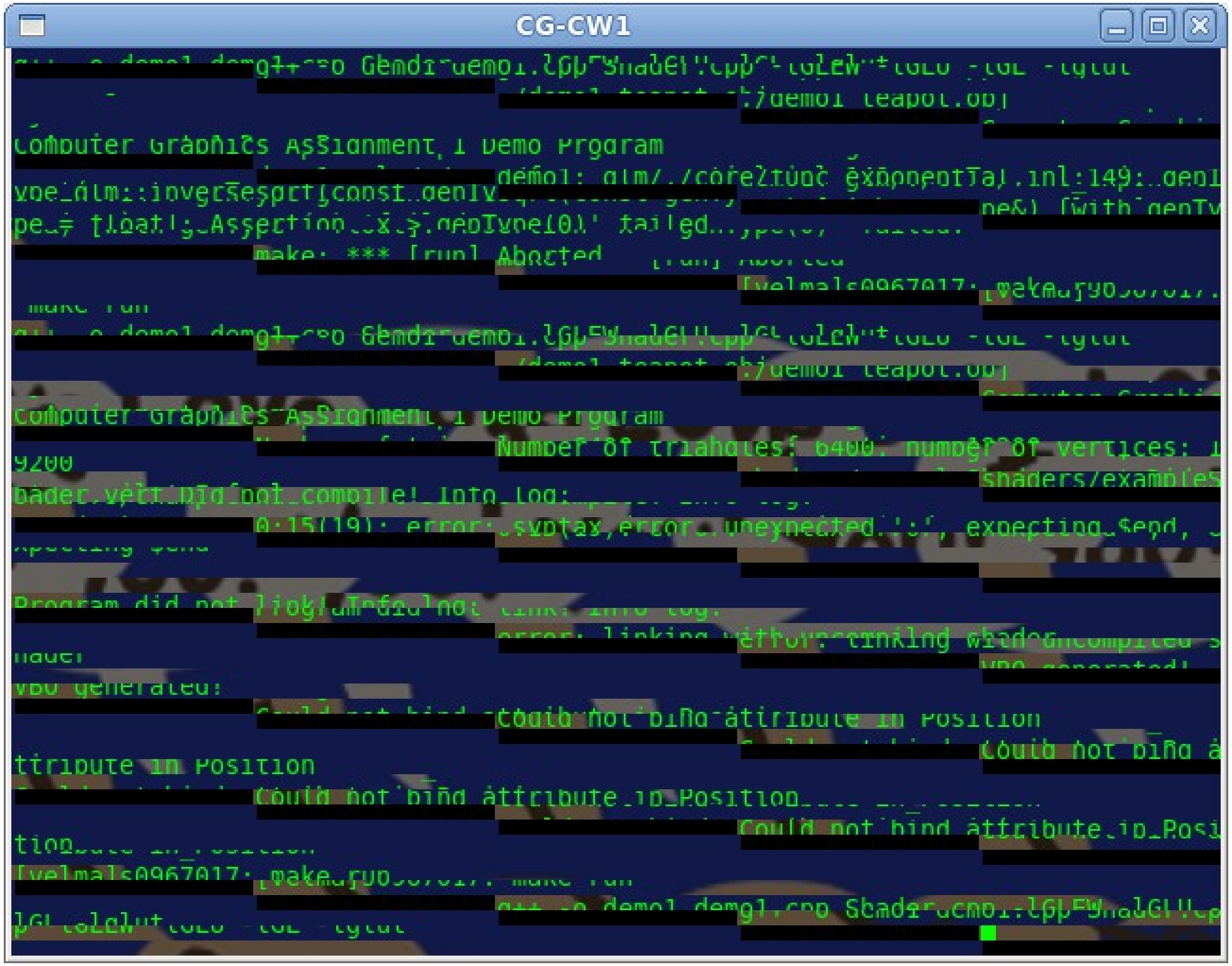
  - Pass as your own uniform variables
    - http://lwjgl.org/wiki/index.php?title=GLSL_Tutorial:_Communicating_with_Shaders

Additional Functions

- Viewpoint control
- Alternative shading e.g. toon shading
- Bump, light, displacement maps
- Texture Mapping
- Anti-Aliasing
- Shadows
- Environment Mapping
- Reflection and Refraction

Error

- Shader does not compile:



```
g++ -o demo1 demo1.cpp Shader.cpp -lGLEW -lGLU -lGL -lglut
./demo1 teapot.obj
Computer Graphics Assignment 1 Demo Program
demo1: glm/./core/func_exponential.inl:149: genType glm::inversesqrt(const genType&) [with genType = float]: Assertion `x > genType(0)' failed.
make: *** [run] Aborted
[velma]s0967017: make run
g++ -o demo1 demo1.cpp Shader.cpp -lGLEW -lGLU -lGL -lglut
./demo1 teapot.obj
Computer Graphics Assignment 1 Demo Program
Number of triangles: 6400; number of vertices: 19200
shaders/examples/shader.vert did not compile! Info log:
0:15(19): error: syntax error, unexpected ':', expecting $end, expecting $end
Program did not link! Info log:
error: linking with uncompiled shader
VBO generated!
Could not bind attribute in_Position
Could not bind attribute in_Position
Could not bind attribute in_Position
Could not bind attribute in_Position
[velma]s0967017: make run
g++ -o demo1 demo1.cpp Shader.cpp -lGLEW -lGLU -lGL -lglut
```