Ciaran O'Hagan

CS4052 Computer Graphics

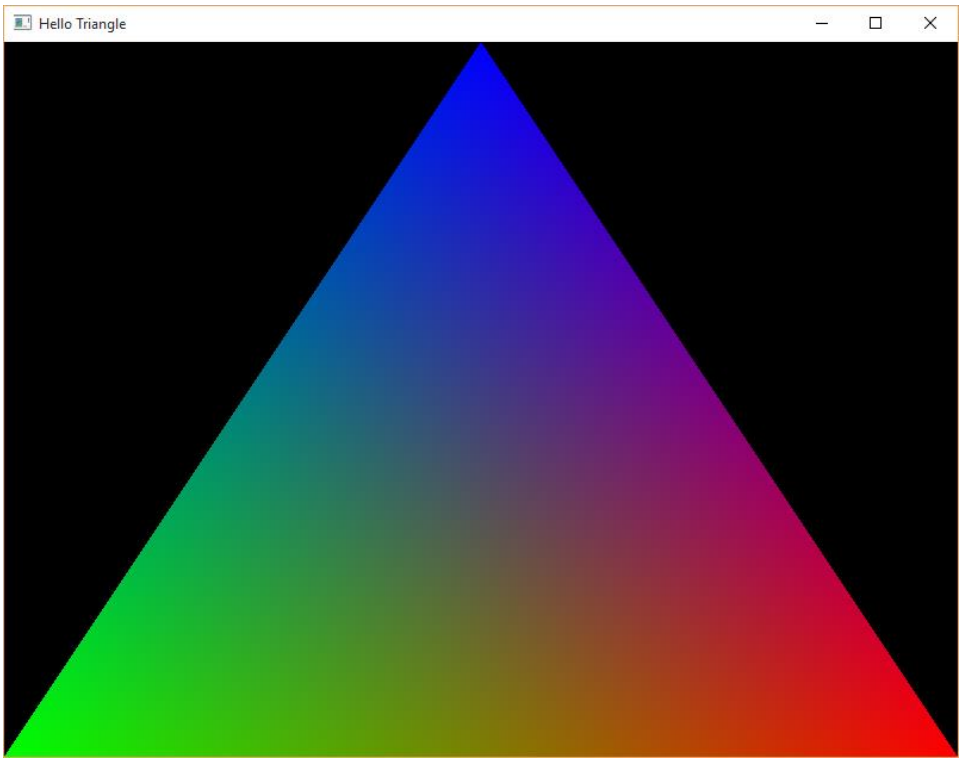
Lab I Report

1. Using colours defined in the Vertex Buffer

To accomplish this, I added a new variable to the fragment shader to match the output sent by the vertex shader:

```
//Vertex Shader
in vec3 vPosition;

in vec4 vColor;

out vec4 color;


void main()
{
    gl_Position = vec4(vPosition.x, vPosition.y, vPosition.z, 2.0);
      color = vColor;
}

//Fragment Shader
in vec4 color;

out vec4 FragColor;

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

Output

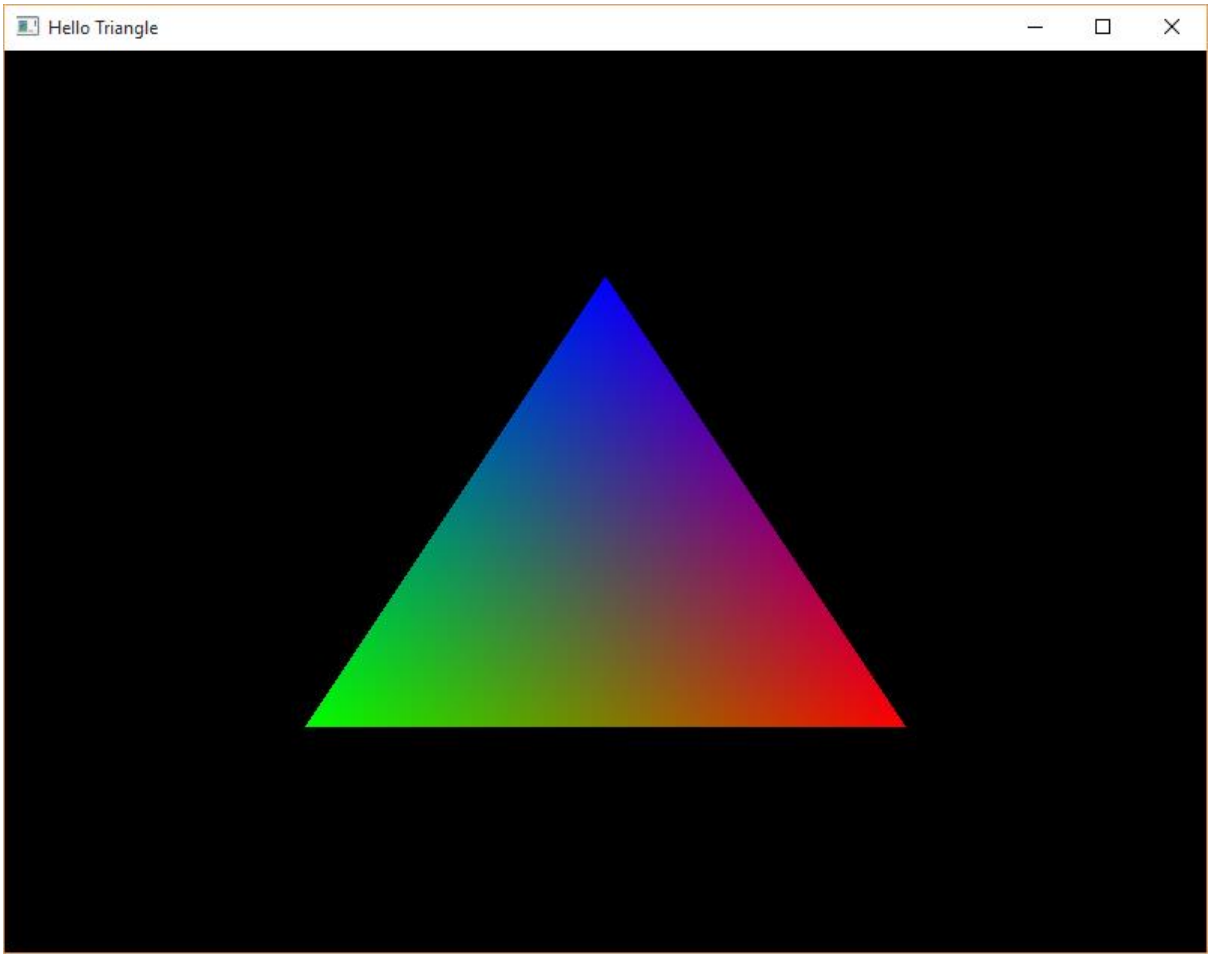


2. Halving the size of the triangle

To halve the size of the on-screen triangle, I simply edited the final w parameter of the `gl_Position` variable within the vertex shader, going from 1.0 to 2.0. What this does (in this context anyway) is divide the x, y and z coordinates provided by the value provided in the w parameter (in this case, 2.0)

```
gl_Position = vec4(vPosition.x, vPosition.y, vPosition.z, 2.0);
```

Output



3. Creating a square and changing the colour of its vertices

To accomplish this, I edited the arrays in the `init()` function containing both the vertex coordinates and their corresponding colours (`vertices[] & colors[]` respectively).

Firstly, to create a square, you have to create a second triangle, so three vertices were added to the array. Two of them matched up with the existing triangle, while the final one created the other shape. To colour the vertices, I assigned the RGB values `1.0 0.0 0.0` (red) and `1.0 1.0 0.0` (yellow) to the corresponding vertices

```
GLfloat vertices[] = {
             -1.0f, -1.0f, 0.0f,
              1.0f, -1.0f, 0.0f,
              1.0f, 1.0f, 0.0f,

              1.0f, 1.0f, 0.0f,
             -1.0f, -1.0f, 0.0f,
             -1.0f, 1.0f, 0.0f
};


GLfloat colors[] = {
             1.0f, 0.0f, 0.0f, 1.0f,
             1.0f, 1.0f, 0.0f, 1.0f,
             1.0f, 0.0f, 0.0f, 1.0f,

             1.0f, 0.0f, 0.0f, 1.0f,
             1.0f, 0.0f, 0.0f, 1.0f,
             1.0f, 1.0f, 0.0f, 1.0f
};
```

Output