Homework 1.2 - OpenGL Basics:

Tristin Greenstein
9/15/2021

Problem:
Create a triangle that will interpolate its vertices between the colors of RGB.

Objective:
To give a hands on assignment to make sure the individual is able to use OpenGL API and apply the basics of Computer Graphics.

This assignment gave me a lot more difficulty than I expected. I would have to say I worked approximately 12 to 16 hours on this assignment trying to figure out the mechanics behind OpenGL. This is was I discovered during this time:
The vertex shader and the fragment shader are the main key part of this program. From here you are able to tap into the API and use built in functions to support the rest of the program.
In both the vertex and the fragment shader, you use “in” and “out” to determine which variables are being fed into or out of these programs. This is similar to how a normal method would have parameters and a return statement.
The vertex shader and the fragment shader are interlinked. What goes into the vertex shader ultimately ends up in the fragment shader.
Using “uniforms”is how we get callable values straight from the later code. By declaring a uniform of vec3, it will then be able to be fed three value vector by calling:

```
int locationOfTrackerUniform = glGetUniformLocation(shaderProgram, "tracker");
glUniform1i(locationOfTrackerUniform, tracker);
```

Here you can see “locationOfTrackerUniform” first ask the program where the uniform of “tracker” located. Then by using “glUniform1i” it can feed in the value to that uniform. glUniform1i means in layman's term to sent 1 (i)nt to the uniform. If you were sending a vec3 you would do glUniform3f.
The while (!glfwWindowShouldClose(window)) operates on the frame rate of your computer and continues looping until forcefully closed. Here any operation you do will differ in results based on what computer you are running the program on. For my particular program, the speed of which the colors interpolate is based on the size of the window. I was unable to figure out how to get around this issue.

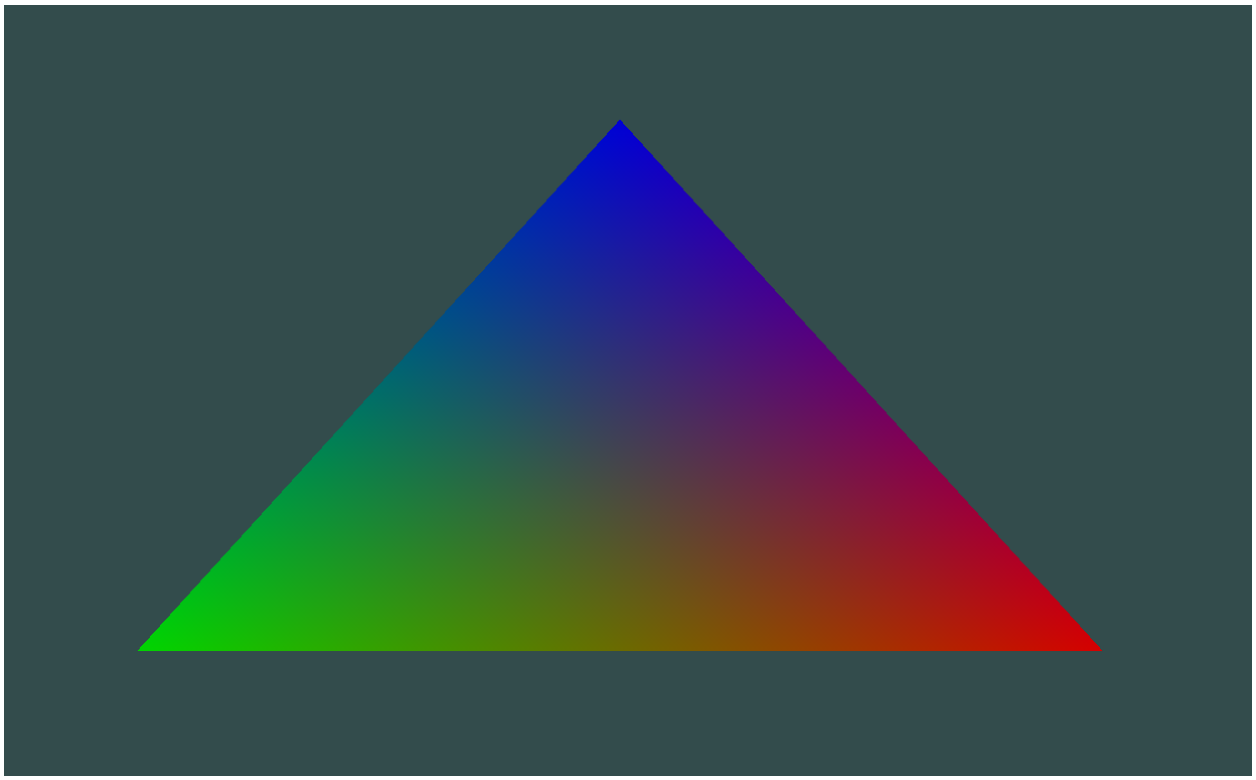
The section in this picture below is very interesting in figuring out how the colors and shape are rendered:

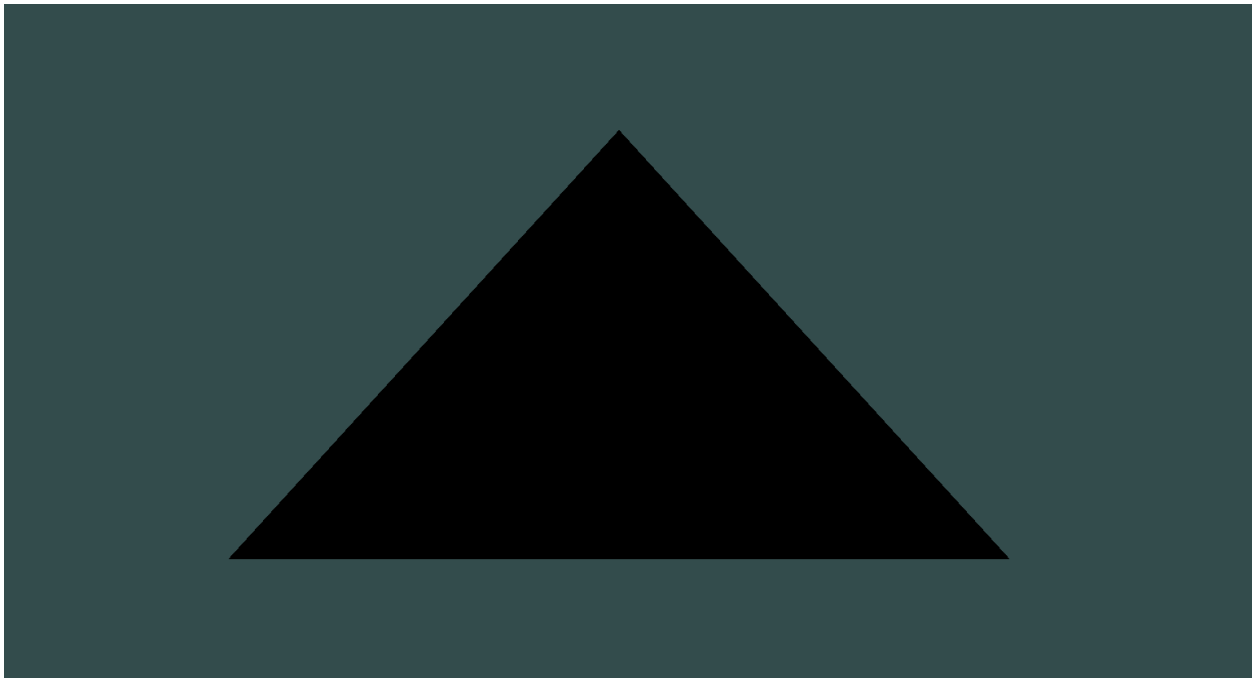
```
float vertices[] = {
    // positions         // colors
     0.5f, -0.5f, 0.0f,  1.0f, 0.0f, 0.0f,  // bottom right
    -0.5f, -0.5f, 0.0f,  0.0f, 1.0f, 0.0f,  // bottom left
     0.0f,  0.5f, 0.0f,  0.0f, 0.0f, 1.0f   // top
};

// Vertex Buffer Objects and Vertex Array Object
unsigned int VBO, VAO;
glGenVertexArrays(1, &VAO);
glGenBuffers(1, &VBO);
// bind the Vertex Array Object first, then bind and set vertex buffer(s), and th
glBindVertexArray(VAO);

glBindBuffer(GL_ARRAY_BUFFER, VBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);
```

The vao is where all the attribute points are stored. The specifics of how it does that is beyond me at this point.
The most difficult part of this assignment was my complete unfamiliarity with OpenGL and its mechanics.





IMPORTANT CODE SNIPPETS:

```
const char* vertexShaderSource =
"#version 330 core\n"
"layout (location = 0) in vec3 aPos;\n"
"layout (location = 1) in vec3 aColor;\n"
"out vec3 ourColor;\n"
"void main()\n"
"{\n"
"   gl_Position = vec4(aPos, 1.0);\n"
"   ourColor = aColor;\n"
"}\0";

// Fragment shader
// The fragment shader is all about calculating the color output of your pixels.
const char* fragmentShaderSource =
"#version 330 core\n"
"out vec4 FragColor;\n"
"uniform int tracker;\n"
"in vec3 ourColor;\n"
"void main()\n"
"{\n"
"   FragColor = vec4(ourColor*(sin(float(tracker)/256)), 1.0f);\n"//ourColor.r ourColor.g ourColor.b
"}\n\0";
```

```
int tracker = 0;
// render loop
// -----------
while (!glfwWindowShouldClose(window))
{
    // input
    // -----
    processInput(window);

    glClearColor(0.2f, 0.3f, 0.3f, 1.0f);
    glClear(GL_COLOR_BUFFER_BIT);

    int locationOfTrackerUniform = glGetUniformLocation(shaderProgram, "tracker");
    glUniform1i(locationOfTrackerUniform, tracker);

    // render the triangle
    glBindVertexArray(VAO);
    glDrawArrays(GL_TRIANGLES, 0, 3);

    // glfw: swap buffers and poll IO events (keys pressed/released, mouse moved etc.)
    // -------------------------------------------------------------------------------
    glfwSwapBuffers(window);
    glfwPollEvents();
    tracker = tracker + 1;
}
```