CIS 451 Lab 5

Andrew Schwartz

11.2.2024

*note: I had to do this lab on windows because the OpenGL shaders did not work on my m1 macbook.

Task One

Code:

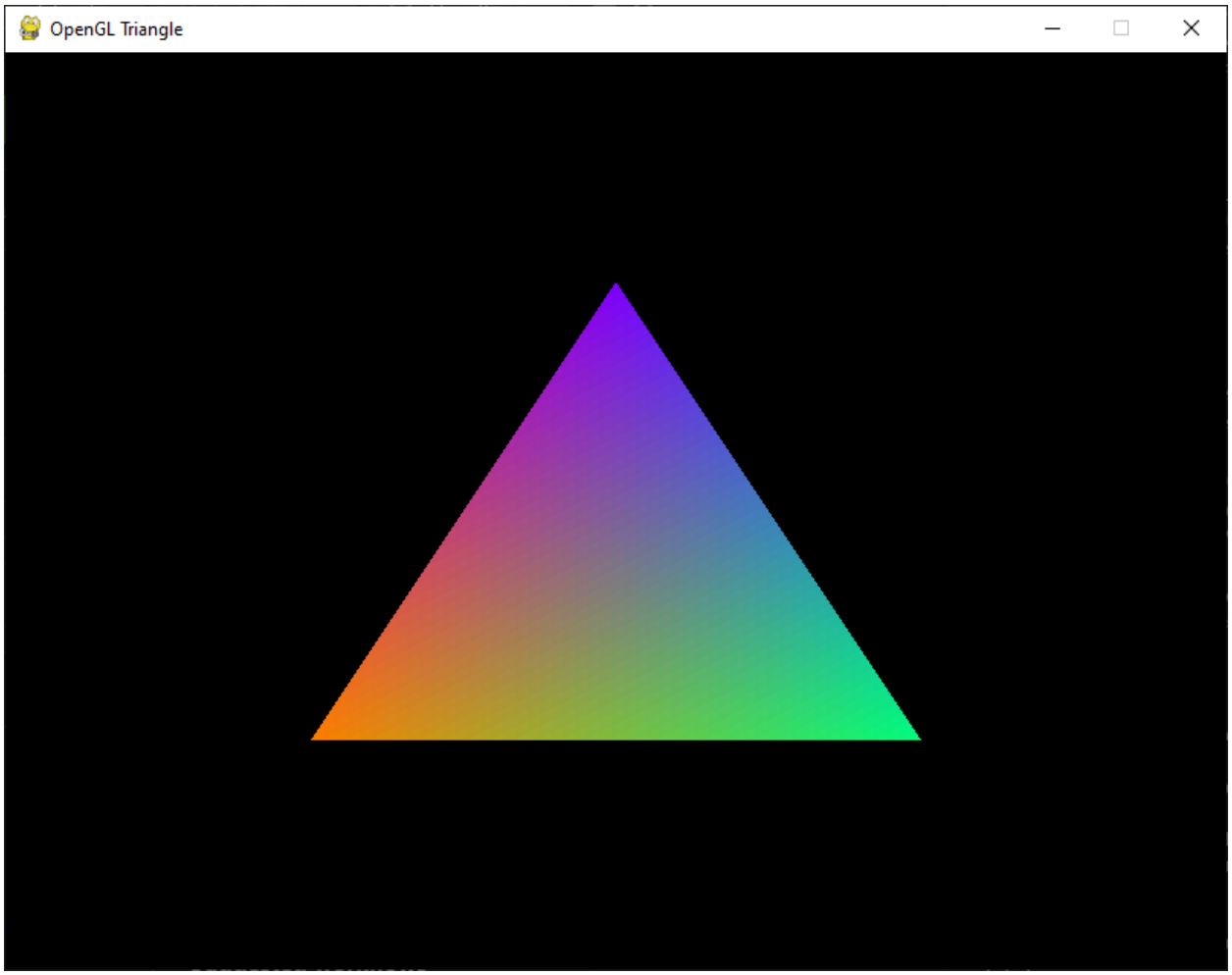
```
import numpy as np
import pygame as pg
from OpenGL.GL import *
import OpenGL.GL.shaders
import ctypes

# Vertex Shader
vertex_src = """
#version 330 core
layout (location=0) in vec3 vertexPos;
layout (location=1) in vec3 vertexColor;
out vec3 fragmentColor;
void main()
{
   gl_Position = vec4(vertexPos, 1.0);
   fragmentColor = vertexColor;
}
"""

# Fragment Shader
fragment_src = """
#version 330 core
in vec3 fragmentColor;
out vec4 color;
void main()
{
   color = vec4(fragmentColor, 1.0);
}
"""

class Triangle:
   def __init__(self):
       # Define vertices: x, y, z, r, g, b
       self.vertices = (
```

```
            -0.5, -0.5, 0.0, 1.0, 0.5, 0.0,  # Vertex 1
             0.5, -0.5, 0.0, 0.0, 1.0, 0.5,  # Vertex 2
             0.0,  0.5, 0.0, 0.5, 0.0, 1.0   # Vertex 3
        )
        self.vertices = np.array(self.vertices, dtype=np.float32)
        self.vertex_count = 3

        # Setup OpenGL buffers
        self.vao = glGenVertexArrays(1)
        glBindVertexArray(self.vao)

        self.vbo = glGenBuffers(1)
        glBindBuffer(GL_ARRAY_BUFFER, self.vbo)
        glBufferData(GL_ARRAY_BUFFER, self.vertices.nbytes, self.vertices,
GL_STATIC_DRAW)

        # Position attribute
        glEnableVertexAttribArray(0)
        glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(0))

        # Color attribute
        glEnableVertexAttribArray(1)
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(12))

    def destroy(self):
        glDeleteVertexArrays(1, (self.vao,))
        glDeleteBuffers(1, (self.vbo,))

def compile_shaders():
    # Compile Vertex Shader
    vertex_shader = glCreateShader(GL_VERTEX_SHADER)
    glShaderSource(vertex_shader, vertex_src)
    glCompileShader(vertex_shader)
    if glGetShaderiv(vertex_shader, GL_COMPILE_STATUS) != GL_TRUE:
        raise RuntimeError(glGetShaderInfoLog(vertex_shader))

    # Compile Fragment Shader
    fragment_shader = glCreateShader(GL_FRAGMENT_SHADER)
    glShaderSource(fragment_shader, fragment_src)
    glCompileShader(fragment_shader)
    if glGetShaderiv(fragment_shader, GL_COMPILE_STATUS) != GL_TRUE:
        raise RuntimeError(glGetShaderInfoLog(fragment_shader))
```

```
    # Link Shaders into a Program
    shader_program = glCreateProgram()
    glAttachShader(shader_program, vertex_shader)
    glAttachShader(shader_program, fragment_shader)
    glLinkProgram(shader_program)
    if glGetProgramiv(shader_program, GL_LINK_STATUS) != GL_TRUE:
        raise RuntimeError(glGetProgramInfoLog(shader_program))

    glDeleteShader(vertex_shader)
    glDeleteShader(fragment_shader)

    return shader_program

def main():
    # Initialize Pygame
    pg.init()
    screen = pg.display.set_mode((800, 600), pg.DOUBLEBUF | pg.OPENGL)
    pg.display.set_caption("OpenGL Triangle")
    clock = pg.time.Clock()

    # Compile shaders and create shader program
    shader_program = compile_shaders()

    # Create Triangle
    triangle = Triangle()

    # Main loop
    running = True
    while running:
        for event in pg.event.get():
            if event.type == pg.QUIT:
                running = False

        # Clear the screen
        glClear(GL_COLOR_BUFFER_BIT)

        # Draw the triangle
        glUseProgram(shader_program)
        glBindVertexArray(triangle.vao)
        glDrawArrays(GL_TRIANGLES, 0, triangle.vertex_count)
```

```
        # Swap buffers
        pg.display.flip()
        clock.tick(60)

    # Cleanup
    triangle.destroy()
    glDeleteProgram(shader_program)
    pg.quit()

if __name__ == "__main__":
    main()
```

Output:



Task Two

Code:

```
import numpy as np
import pygame as pg
from OpenGL.GL import *
import OpenGL.GL.shaders
import ctypes

# Function to read shader source from a file
def read_shader_file(filepath):
   with open(filepath, 'r') as f:
       return f.read()

class Triangle:
   def __init__(self):
       # Define vertices: x, y, z, r, g, b
       self.vertices = (
           -0.5, -0.5, 0.0, 1.0, 0.5, 0.0,  # Vertex 1
            0.5, -0.5, 0.0, 0.0, 1.0, 0.5,  # Vertex 2
            0.0,  0.5, 0.0, 0.5, 0.0, 1.0   # Vertex 3
       )
       self.vertices = np.array(self.vertices, dtype=np.float32)
       self.vertex_count = 3

       # Setup OpenGL buffers
       self.vao = glGenVertexArrays(1)
       glBindVertexArray(self.vao)

       self.vbo = glGenBuffers(1)
       glBindBuffer(GL_ARRAY_BUFFER, self.vbo)
       glBufferData(GL_ARRAY_BUFFER, self.vertices.nbytes, self.vertices,
GL_STATIC_DRAW)

       # Position attribute
       glEnableVertexAttribArray(0)
       glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(0))

       # Color attribute
       glEnableVertexAttribArray(1)
```

```
        glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 24, ctypes.c_void_p(12))

    def destroy(self):
        glDeleteVertexArrays(1, (self.vao,))
        glDeleteBuffers(1, (self.vbo,))

def compile_shaders():
    # Read shader source from files
    vertex_src = read_shader_file('vertex.txt')
    fragment_src = read_shader_file('fragment.txt')

    # Compile Vertex Shader
    vertex_shader = glCreateShader(GL_VERTEX_SHADER)
    glShaderSource(vertex_shader, vertex_src)
    glCompileShader(vertex_shader)
    if glGetShaderiv(vertex_shader, GL_COMPILE_STATUS) != GL_TRUE:
        raise RuntimeError(glGetShaderInfoLog(vertex_shader))

    # Compile Fragment Shader
    fragment_shader = glCreateShader(GL_FRAGMENT_SHADER)
    glShaderSource(fragment_shader, fragment_src)
    glCompileShader(fragment_shader)
    if glGetShaderiv(fragment_shader, GL_COMPILE_STATUS) != GL_TRUE:
        raise RuntimeError(glGetShaderInfoLog(fragment_shader))

    # Link Shaders into a Program
    shader_program = glCreateProgram()
    glAttachShader(shader_program, vertex_shader)
    glAttachShader(shader_program, fragment_shader)
    glLinkProgram(shader_program)
    if glGetProgramiv(shader_program, GL_LINK_STATUS) != GL_TRUE:
        raise RuntimeError(glGetProgramInfoLog(shader_program))

    glDeleteShader(vertex_shader)
    glDeleteShader(fragment_shader)

    return shader_program

def main():
    # Initialize Pygame
    pg.init()
    screen = pg.display.set_mode((800, 600), pg.DOUBLEBUF | pg.OPENGL)
```

```
    pg.display.set_caption("OpenGL Triangle from Files")
    clock = pg.time.Clock()

    # Compile shaders and create shader program
    shader_program = compile_shaders()

    # Create Triangle
    triangle = Triangle()

    # Main loop
    running = True
    while running:
        for event in pg.event.get():
            if event.type == pg.QUIT:
                running = False

        # Clear the screen
        glClear(GL_COLOR_BUFFER_BIT)

        # Draw the triangle
        glUseProgram(shader_program)
        glBindVertexArray(triangle.vao)
        glDrawArrays(GL_TRIANGLES, 0, triangle.vertex_count)

        # Swap buffers
        pg.display.flip()
        clock.tick(60)

    # Cleanup
    triangle.destroy()
    glDeleteProgram(shader_program)
    pg.quit()

if __name__ == "__main__":
    main()
```

Output: