2024 Fall CIS515/451 – Lab5
Shading Language

Instructor: Dr. Jie Shen
Release date: Nov. 1, 2024
Due date: Nov. 8, 2024

Student Name:

Undergraduate students may work on each lab in teams of two members, while graduate students are required to work on the lab individually.

For each of the following tasks, you need to present your running results in the format of screenshots. Source code should be provided through copy and paste.

In PyOpenGL, you can use OpenGL Shading Language (GLSL) to create and manage shaders for a rendering task in a graphics pipeline. Overall, two popular shaders are (a) vertex shader and (b) fragment shader. The vertex shader is used to compute the position of each vertex, while the fragment shader aims to determine the color information of each pixel. Each shader can be represented by a Python triple-quote string as follows:

```
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;
}
"""
```

Alternatively, each shader can be saved as a .txt file. In this case, you need to open the text file, read in all the lines, and store the information as a Python string as follows:

```
with open(vertexFilepath,'r') as f:
     vertex_src = f.readlines()
```

Secondly, we need to compile the shader source string in the following way:

```
from OpenGL.GL import *
import OpenGL.GL.shaders

# Compile the Vertex Shader
```

```
vertex_shader = glCreateShader(GL_VERTEX_SHADER)
glShaderSource(vertex_shader, vertex_src)
glCompileShader(vertex_shader)

# Check for compilation errors
if glGetShaderiv(vertex_shader, GL_COMPILE_STATUS) != GL_TRUE:
    raise RuntimeError(glGetShaderInfoLog(vertex_shader))

# Compile the Fragment Shader
fragment_shader = glCreateShader(GL_FRAGMENT_SHADER)
glShaderSource(fragment_shader, fragment_src)
glCompileShader(fragment_shader)

# Check for compilation errors
if glGetShaderiv(fragment_shader, GL_COMPILE_STATUS) != GL_TRUE:
    raise RuntimeError(glGetShaderInfoLog(fragment_shader))
```

Thirdly, we need to link shaders into a program and can then use the program before rendering any object in our system:

```
 # Link Shaders into a Program
shader_program = glCreateProgram()
glAttachShader(shader_program, vertex_shader)
glAttachShader(shader_program, fragment_shader)
glLinkProgram(shader_program)

# Check for linking errors
if glGetProgramiv(shader_program, GL_LINK_STATUS) != GL_TRUE:
    raise RuntimeError(glGetProgramInfoLog(shader_program))

glUseProgram(shader_program)
# Render your objects here
# ...
```

Question 1 Shaders in a Python Triple-quote String (10 points)

Use the following two strings:

```
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;
}
"""

#with open(fragmentFilepath,'r') as f:
    #fragment_src = f.readlines()
fragment_src = """
```

```
        #version 330 core

        in vec3 fragmentColor;

        out vec4 color;

        void main()
        {
            color = vec4(fragmentColor, 1.0);
        }
        """
```

The geometry is a triangle defined as follows:

```
class Triangle:
    def __init__(self):


        # x, y, z, r, g, b
        self.vertices = (
            -0.5, -0.5, 0.0, 1.0, 0.5, 0.0,
             0.5, -0.5, 0.0, 0.0, 1.0, 0.5,
             0.0,  0.5, 0.0, 0.5, 0.0, 1.0
        )
        self.vertices = np.array(self.vertices, dtype=np.float32)

        self.vertex_count = 3

        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)

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

        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,))
```

A while loop can be used to draw the above triangle:

```
        running = True
        while (running):
            #check events
            for event in pg.event.get():
                if (event.type == pg.QUIT):
                    running = False
            #refresh screen
            glClear(GL_COLOR_BUFFER_BIT)
```

```
            glBindVertexArray(self.triangle.vao)
            glUseProgram(self.shader)
            glDrawArrays(GL_TRIANGLES, 0, self.triangle.vertex_count)

            pg.display.flip()

            #timing
            self.clock.tick(60)
```

Create a screenshot of the rendered triangle.

Question 2 Shaders in a text file (10 points)

Use two shader files: vertex.txt and fragment.txt, which are available at Canvas.

Perform a similar rendering as in Question 1 and show a screenshot.

Submission of Lab 5

You should create screenshots for all the tasks in this lab.
You should submit your work through Canvas site as an MS word document or a pdf file for cis 515/451 lab. The filename of the word document should follow the convention: FirstName_LastName_Cis515-451-Lab5.doc.