Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report - I

[COMP 342]

Submitted by:

Abhijeet Poudel (44)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 18/12/2022

Table of Contents

Chapter 1: Introduction

1.1 Language Primitives and Graphics Library

Graphics Library: PyOpenGL 3.1.6

Programming Language: Python 3.10.5

Window Context: GLFW

Helpers: Numpy

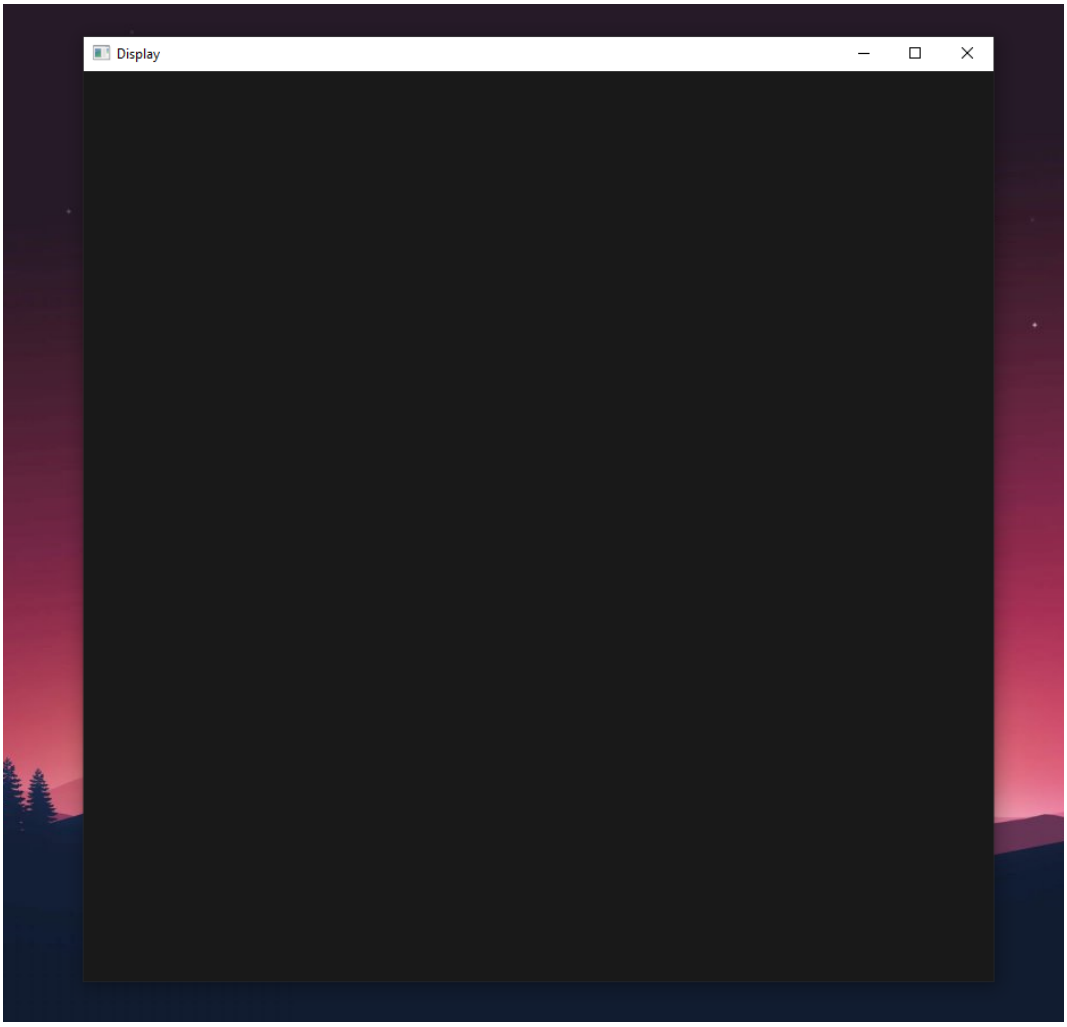
Chapter 2: Environment Setup and Resolution

2.1. Source Code for Environment Setup and Resolution

```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader


RESOLUTION = 800

def window_resize(window, width, height):
    glViewport(0, 0, width, height)
    print(f"The Resolution of the Output Screen is:{width} x {height}")


def main():
    if not glfw.init():
        raise Exception("glfw can not be initialized!")

    window = glfw.create_window(RESOLUTION, RESOLUTION, "Display", None,
None)

    if not window:
        glfw.terminate()
        raise Exception("glfw window can not be created!")
```

```
    # glfw.set_window_pos(window,100,100)
    glfw.set_window_size_callback(window, window_resize)
    glfw.make_context_current(window)

    glClearColor(0.1, 0.1, 0.1, 0.1)

    while not glfw.window_should_close(window):
        glfw.poll_events()
        glClear(GL_COLOR_BUFFER_BIT)
        glfw.swap_buffers(window)

    glfw.terminate()


main()
```

2.2 Output



```
D:\Computer-Graphics-III.I\src>python resolution.py
The Resolution of the Output Screen is:0 x 0
The Resolution of the Output Screen is:800 x 800
```

Chapter 3: Flag of Nepal

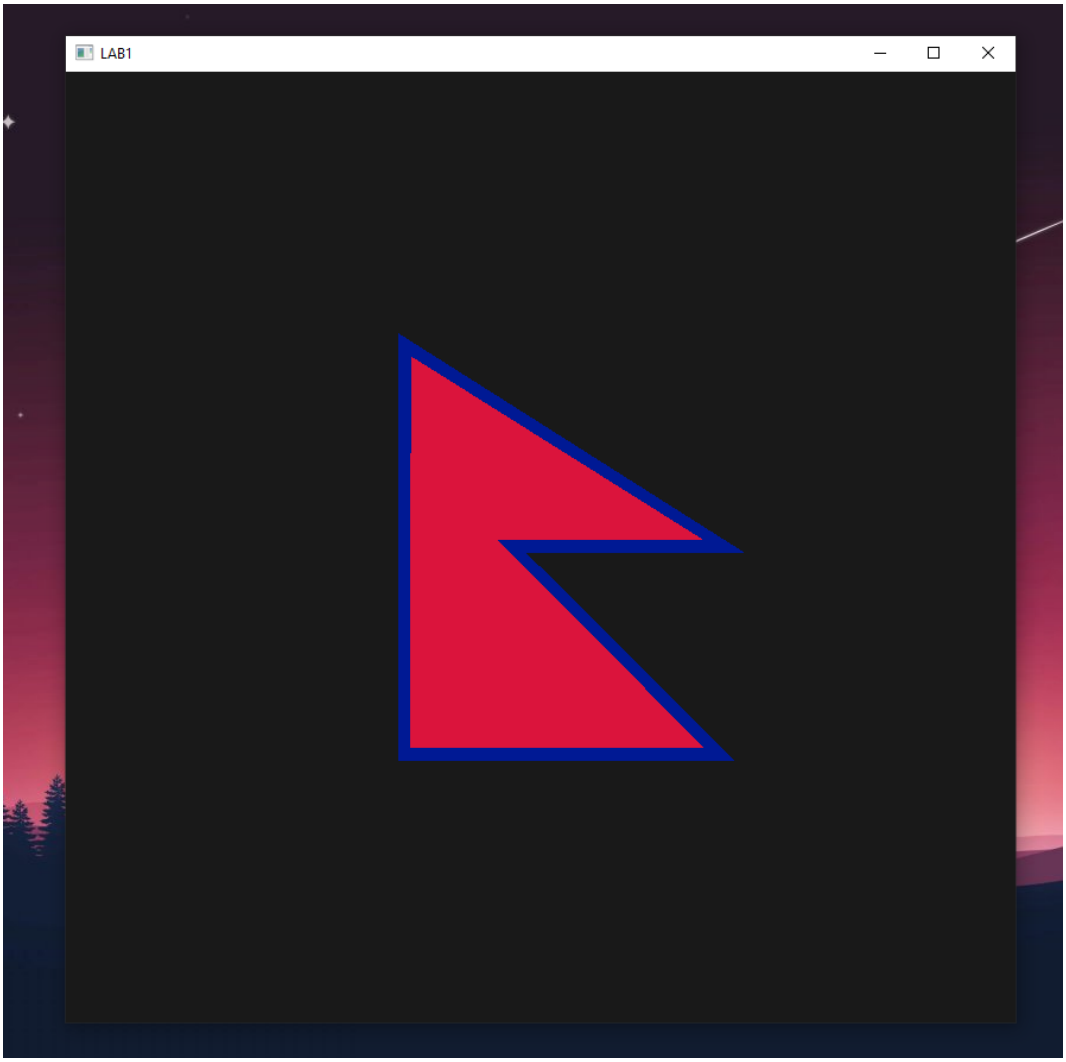
3.1. Source Code

```
import glfw
import numpy as np
from OpenGL.GL import *
from OpenGL.GL.shaders import compileProgram, compileShader


RESOLUTION = 800
def window_resize(window, width, height):
    glViewport(0,0,width,height)

def main():

  vertex_src = """
  #version 330

  layout(location=0) in vec3 aPos;
  layout(location=1) in vec3 aColor;

  out vec3 vColor;

  void main(){
      gl_Position = vec4(aPos,1.0);
      vColor= aColor;
      }
  """

  fragment_src = """
  #version 330

  in vec3 vColor;
  out vec4 FragColor;

  void main(){
      FragColor = vec4(vColor, 0.0f);
```

```
  }
  """

  if not glfw.init():
      raise Exception("glfw can not be initialized!")

  window = glfw.create_window(RESOLUTION, RESOLUTION, "LAB1", None,
None)

  if not window:
      glfw.terminate()
      raise Exception("glfw window can not be created!")

  # glfw.set_window_pos(window,100,100)
  glfw.set_window_size_callback(window,window_resize)
  glfw.make_context_current(window)

  vertices = [
        #1st
        -0.30000000000000004, 0.44999999999999996,0,
        -0.30000000000000004, -0.012968750000000029,0,
        0.43124999999999997, -0.012968750000000029,0,
        #2nd
        -0.30000000000000004, 0.25625,0,
        -0.30000000000000004, -0.4501562500000001,0,
        0.40937500000000004, -0.4501562500000001,0,
        #3rd
        -0.27326877, 0.4015625,0,
        -0.27326877, 0.014062500000000033,0,
        0.34437500000000004, 0.014062500000000033,0,
        #4th
        -0.27390625, 0.19890625000000003,0,
        -0.27390625, -0.4223437499999999,0,
        0.34421874999999985, -0.4223437499999999,0,
    ]
  colors = [#1st
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            #2nd
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            #3rd
            0.859, 0.078125, 0.234,
            0.859, 0.078125, 0.234,
            0.859, 0.078125, 0.234,
            #4th
            0.859, 0.078125, 0.234,
```

```
            0.859, 0.078125, 0.234,
            0.859, 0.078125, 0.234,
           ]

  bufferData= vertices+colors
  indicesData = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11],
dtype=np.uint32)
  vertices = np.array(vertices,dtype=np.float32)
  bufferData = np.array(bufferData,dtype=np.float32)

  shader =
compileProgram(compileShader(vertex_src,GL_VERTEX_SHADER),compileShader(
fragment_src,GL_FRAGMENT_SHADER))

  vertex_buffer_object = glGenBuffers(1)
  glBindBuffer(GL_ARRAY_BUFFER,vertex_buffer_object)
  glBufferData(GL_ARRAY_BUFFER,bufferData.nbytes,bufferData,GL_STREAM_DR
AW)

  element_buffer_object = glGenBuffers(1)
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER,element_buffer_object)
  glBufferData(GL_ELEMENT_ARRAY_BUFFER,indicesData.nbytes,indicesData,GL
_STREAM_DRAW)

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

  glEnableVertexAttribArray(1)
  glVertexAttribPointer(1,3,GL_FLOAT,GL_FALSE,0,ctypes.c_void_p(vertices
.nbytes))

  glUseProgram(shader)
  glClearColor(0.1,0.1,0.1,0.1)

  while not glfw.window_should_close(window):
    glfw.poll_events()
    glClear(GL_COLOR_BUFFER_BIT)
    glDrawElements(GL_TRIANGLES,len(indicesData),GL_UNSIGNED_INT,None)
    glfw.swap_buffers(window)

  glfw.terminate()

main()
```

3.2. Output



3.3. Conclusion

Programmable pipelines in modern OpenGL allow developers to create custom shaders that are written in a high-level language such as GLSL (OpenGL Shading Language). This allows developers to create complex effects and customizations that are not possible in the fixed-function pipeline. In Python, the PyOpenGL library provides bindings for the OpenGL API which can be used to access these shaders and create custom effects. Additionally, the PyGLFW library can be used to create the necessary windowing context for using OpenGL within a Python application.

Procedure

1. This code is used to draw a the Flag of Nepal using OpenGL in Python. It sets up a window for the shape to be drawn in and then creates the necessary buffers and vertex arrays to draw the shape.

2. The first step is to import the necessary modules. glfw is used to create the window and set the window size, while numpy and OpenGl are used to create the buffers and vertex arrays needed to draw the shape.

3. The window_resize() function is then defined to set the viewport size when the window is resized.

4. In the main() function, the shaders are defined first, which are the vertex_src and fragment_src. The vertex shader is responsible for transforming the vertices and the fragment shader is responsible for coloring the shape.

5. glfw is then initialized and the window is created with a width and height of RESOLUTION (800). The window_resize function is then set to be called when the window is resized.

6. The vertices and colors of the shape are then defined in a list and the indicesData array is also defined. The vertices and colors are then converted to a numpy array for use in the vertex and fragment shaders.

7. Next, the vertex and fragment shaders are compiled and a shader program is created. The buffers and vertex arrays are then created, with the vertex and color data being stored in the bufferData array.

```
vertex_src = """
  #version 330

  layout(location=0) in vec3 aPos;
  layout(location=1) in vec3 aColor;
```

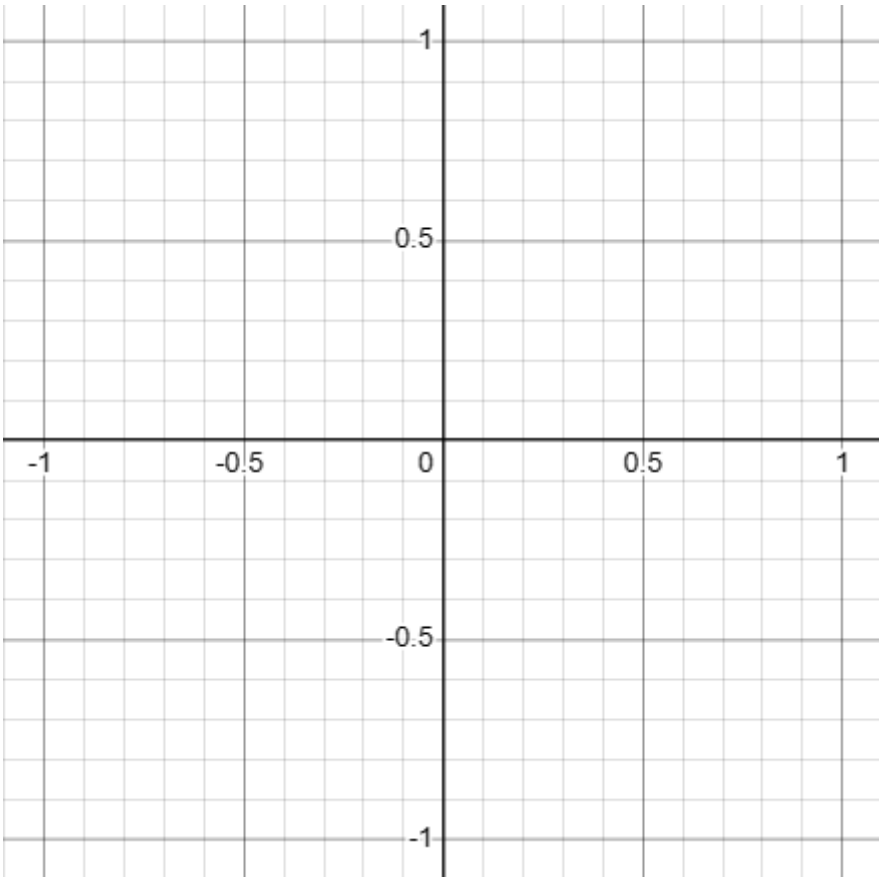
```
  out vec3 vColor;

  void main(){
      gl_Position = vec4(aPos,1.0);
      vColor= aColor;
      }
  """

  fragment_src = """
  #version 330

  in vec3 vColor;
  out vec4 FragColor;

  void main(){
      FragColor = vec4(vColor, 0.0f);

  }
  """
```

8. The vertex and color attributes are then enabled and the shader program is set to be used. The glClearColor() function is then called to set the background color of the window.

```
vertices = [
        #1st
        -0.30000000000000004, 0.44999999999999996,0,
        -0.30000000000000004, -0.012968750000000029,0,
        0.43124999999999997, -0.012968750000000029,0,
        #2nd
        -0.30000000000000004, 0.25625,0,
        -0.30000000000000004, -0.4501562500000001,0,
        0.40937500000000004, -0.4501562500000001,0,
        #3rd
        -0.27326877, 0.4015625,0,
        -0.27326877, 0.014062500000000033,0,
        0.34437500000000004, 0.014062500000000033,0,
        #4th
        -0.27390625, 0.19890625000000003,0,
        -0.27390625, -0.4223437499999999,0,
        0.34421874999999985, -0.4223437499999999,0,
    ]
  colors = [#1st
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            0, 0.10, 0.58,
            #2nd
```

```
        0, 0.10, 0.58,
        0, 0.10, 0.58,
        0, 0.10, 0.58,
        #3rd
        0.859, 0.078125, 0.234,
        0.859, 0.078125, 0.234,
        0.859, 0.078125, 0.234,
        #4th
        0.859, 0.078125, 0.234,
        0.859, 0.078125, 0.234,
        0.859, 0.078125, 0.234,
    ]
```

Here the vertices were obtained after mapping the actual flag of Nepal into photoshop the obtaining the **normalized viewing coordinates** by dividing the coordinates by the resolution. The color codes were obtained similarly. This was done so to abide by the Modern OpenGL's configuration where the points are mapped on window as seen in the figure below.



9. Finally, a while loop is used to draw the shape on each frame until the window is closed. The glfw.poll_events() function is used to poll for keyboard and mouse events, the glClear() function is used to clear the frame buffer, the glDrawElements() function is used to draw the shape, and the glfw.swap_buffers() function is used to swap the front and back buffers.