Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Lab Report V

[COMP 342]

Submitted by:

Abhijeet Poudel

CS(III/I)

Roll No: 44

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 18th December 2022

Table of Contents

Chapter 1: Introduction

For our lab work and project, the following programming language and tools are used:

Graphics Library: PyOpenGL 3.1.6

Programming Language: Python 3.9

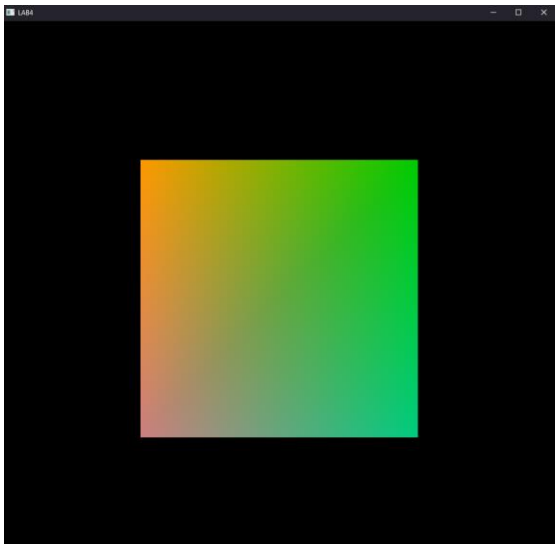
Window Context: GLFW

Helpers: Numpy, ctypes

Chapter 2: The Two-Dimensional Object and Preliminaries

The two-dimensional object that we will be using for this demonstration is a simple cube that is drawn with the help of the primitive GL_TRIANGLES. The cube had to be manually drawn as there is no default three-dimensional object that comes with the Modern OpenGL in contrary to the traditional OpenGL where cube came with the library.

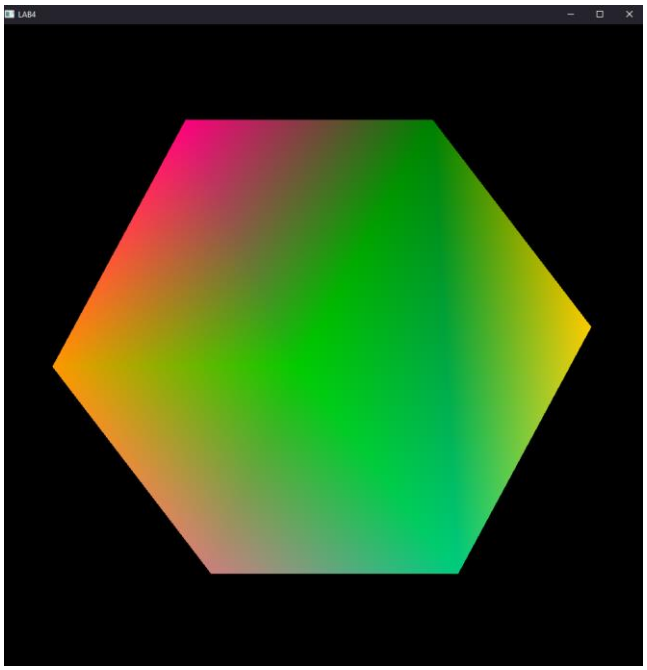
Hence, a custom cube was made using vertices, colors and indices. From the front view the cube lookedlike a square as the overlapped vertices couldn’t be seen.



Hence, for the purpose of this Lab Report the cube will be rotated along the x by 40 degrees and

y axis by 50 degrees to show its depth and will be considered the base cube for the remainder of this project.

```
transformY = np.matrix(f"""
{np.cos((np.pi / 180) * 40)},0.0,{np.sin(-(np.pi / 180) * 40)},0.0;
0.0,1.0,0.0,0.0;
{np.sin((np.pi / 180) * 40)},0.0,{np.cos((np.pi / 180) * 40)},0.0;
0.0,0.0,0.0,1.0""")

transformX = np.matrix(f""" 1.0,  0.0, 0.0, 0.0;
          0.0,{np.cos((np.pi / 180) * -50)},{np.sin((np.pi / 180) * -50)}, 0.0;
          0.0,{np.sin(-(np.pi / 180) * -50)},{np.cos((np.pi / 180) * -50)}, 0.0;
          0.0, 0.0,0.0,1.0""")
```



We will use a 4 by 4 homogenous matrix to translate the points of the three-dimensional object and hence multiply the matrix with the vertices of the object, a cube in this case. We will multiply the generated transformation matrix with the object vertices inside the vertex shader and fragment as follows:

```
  vertex_src = """
#version 330
layout(location=0) in vec3 aPos;
layout(location=1)  in  vec3 aColor;

out vec3 vColor;

uniform mat4 transformation;
void main(){
    gl_Position = transformation * vec4(aPos,1.0);
    vColor=aColor;

    }
"""

  fragment_src = """
#version 330
in vec3 vColor;
out vec4 FragColor;
void main(){
    FragColor = vec4(vColor,0.0f);

}
"""
```

Chapter 3: 3D Translation

3.1. Translation Matrix

In order to translate any 3D object the matrix to be used is:

1	0	0	Tx
0	1	0	Ty
0	0	1	Tz
0	0	0	1

Furthermore, the vertex shader performs matrix multiplications in column major basis but the normal translation matrix is row major matrix so firstly the above matrix is to be transposed as:

1	0	0	0
0	1	0	0

0	0	1	0
Tx	Ty	Tz	1

Now, using this matrix any 3D object can be easily translated by the given factor.

3.2. Source Code

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

from helpers import altList, toNVC


def window_resize(window, width, height):
    glViewport(0, 0, width, height)


RESOLUTION = 1000
transformY = np.matrix(f"""
{np.cos((np.pi / 180) * 40)},
0.0,
{np.sin(-(np.pi / 180) * 40)},
0.0;
0.0,1.0,0.0,0.0;
{np.sin((np.pi / 180) * 40)},
0.0,
{np.cos((np.pi / 180) * 40)},
0.0;
0.0,0.0,0.0,1.0""")

transformX = np.matrix(f""" 1.0,0.0,0.0, 0.0;
          0.0,
          {np.cos((np.pi / 180) * -50)},
          {np.sin((np.pi / 180) * -50)},
          0.0;
          0.0,
          {np.sin(-(np.pi / 180) * -50)},
          {np.cos((np.pi / 180) * -50)},
```

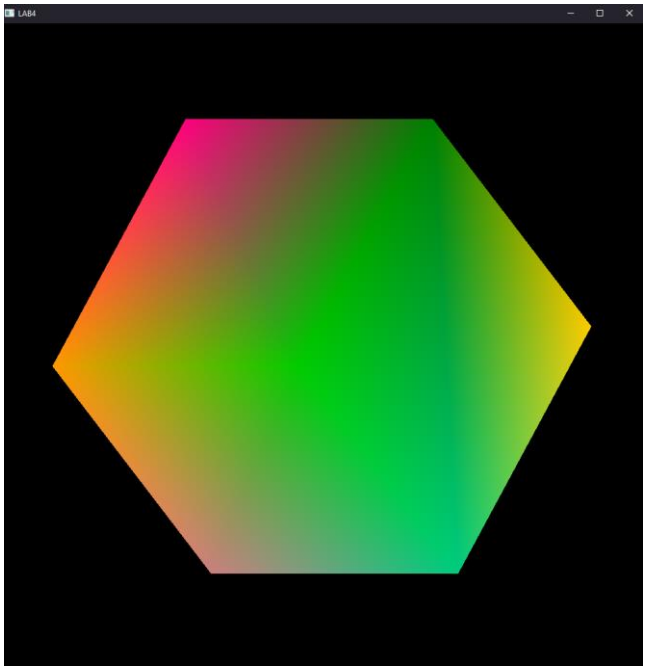
```
          0.0;
          0.0, 0.0,0.0,1.0""")
newTransform = np.matrix(
    f""" 1.0, 0.0, 0.0,0.0;
        0.0,1.0, 0.0,0.0;
        0.0,0.0,1.0,0.0;
        0.2,0.2,0.2,1.0"""
)

transformation = np.dot(newTransform, np.dot(transformX, transformY))


def main(transformation):

    vertex_src = """
  #version 330
  layout(location=0) in vec3 aPos;
  layout(location=1)  in  vec3 aColor;

  out vec3 vColor;

  uniform mat4 transformation;
  void main(){
      gl_Position = transformation * 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, "LAB4", 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 = [
        -0.5,-0.5, 0.5,0.5,
        -0.5,0.5,0.5, 0.5,
        0.5,-0.5,0.5,0.5,
        -0.5,-0.5,-0.5,0.5,
        -0.5,-0.5,0.5,0.5,
        -0.5,-0.5,0.5, -0.5,
    ]

    colors = [
        0.0,0.0, 0.0,
        1.0, 0.8, 0.0,
        0.0, 0.5,0.0,
        1.0, 0.0, 0.5,
        0.8, 0.5, 0.5,
        0.0, 0.8,0.5,
        0.0, 0.8,0.0,
        1.0,0.6, 0.0,
    ]

    indices = [
        0,1,2,2,3,0,4,5,6,6,7,4,4,5,1,1,0,4, 6,7, 3,3, 2, 6,  5,6,2,2, 1,5,7,4,
        0, 0, 3, 7,
    ]

    bufferData = vertices + colors

    vertices = np.array(vertices, dtype=np.float32)
    bufferData = np.array(bufferData, dtype=np.float32)
    indices = np.array(indices, dtype=np.uint32)

    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)
```

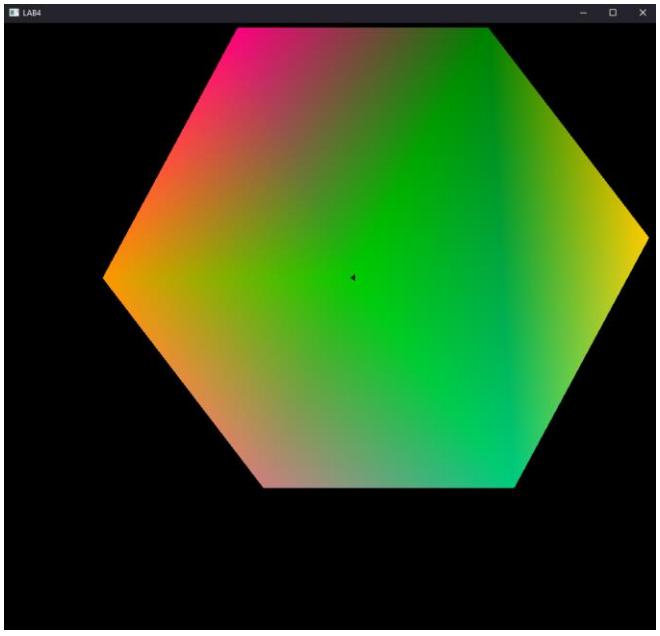
```
    glBufferData(GL_ARRAY_BUFFER, bufferData.nbytes, bufferData,
                 GL_STREAM_DRAW)

    element_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
                 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)
    transformation_location = glGetUniformLocation(shader, "transformation")

    glEnable(GL_DEPTH_TEST)

    while not glfw.window_should_close(window):

        glfw.poll_events()
        glClear(GL_DEPTH_BUFFER_BIT)
        glUniformMatrix4fv(transformation_location, 1, GL_FALSE,
                           transformation)
        # glUniformMatrix4fv(transformation_location,1,GL_FALSE,transformation)
        glDrawElements(GL_TRIANGLES, len(indices), GL_UNSIGNED_INT, None)

        glfw.swap_buffers(window)

    glfw.terminate()


main(transformation)
```

3.3 Output

3.3.1. Original Cube



3.3.2. Triangle after translation of 0.5



Chapter 4: 3D Rotation

4.1. Rotation Matrix

The rotation matrix for 3D is:

Rotation on x-axis:

1	0	0	0
0	cos(θ)	-sin(θ)	0
0	sin(θ)	cos(θ)	0
0	0	0	1

Rotation on y-axis:

cos(θ)	0	sin(c)	0
0	1	0	0
sin(-c)	0	cos(θ)	0
0	0	0	1

Rotation on z-axis:

cos(θ)	sin(-c)	0	0
sin(c)	cos(θ)	0	0
0	0	1	0
0	0	0	1

However, as this matrix is row major and vertex shader calculates matrix multiplication in column major basis, we have to transform the above matrix as:

Rotation on x-axis:

1	0	0	0
0	cos(θ)	sin(θ)	0
0	-sin(θ)	cos(θ)	0
0	0	0	1

Rotation on y-axis:

cos(θ)	0	sin(-c)	0
0	1	0	0
sin(c)	0	cos(θ)	0
0	0	0	1

Rotation on z-axis:

cos(θ)	sin(θ)	0	0
-sin(θ)	cos(θ)	0	0
0	0	1	0
0	0	0	1

Now using this matrix, we can rotate any 3D object.

4.2. Source Code

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

from helpers import altList, toNVC


def window_resize(window, width, height):
    glViewport(0, 0, width, height)


RESOLUTION = 1000
transformY = np.matrix(f"""
{np.cos((np.pi / 180) * 40)},
0.0,
{np.sin(-(np.pi / 180) * 40)},
0.0;
0.0,1.0,0.0,0.0;
{np.sin((np.pi / 180) * 40)},
0.0,
{np.cos((np.pi / 180) * 40)},
0.0;
0.0,0.0,0.0,1.0""")

transformX = np.matrix(f""" 1.0,0.0,0.0, 0.0;
          0.0,
          {np.cos((np.pi / 180) * -50)},
          {np.sin((np.pi / 180) * -50)},
          0.0;
          0.0,
          {np.sin(-(np.pi / 180) * -50)},
          {np.cos((np.pi / 180) * -50)},
          0.0;
          0.0, 0.0,0.0,1.0""")
newTransform = np.matrix(
    f"""{np.cos((np.pi / 180) * 45)},0.0,{np.sin(-(np.pi / 180) * 45)},0.0;
        0,1,0,0.0;
        {np.sin((np.pi / 180) * 45)},0,{np.cos((np.pi / 180) * 45)},0.0;
        0.0,0.0,0.0,1.0""")


transformation = np.dot(newTransform, np.dot(transformX, transformY))
```

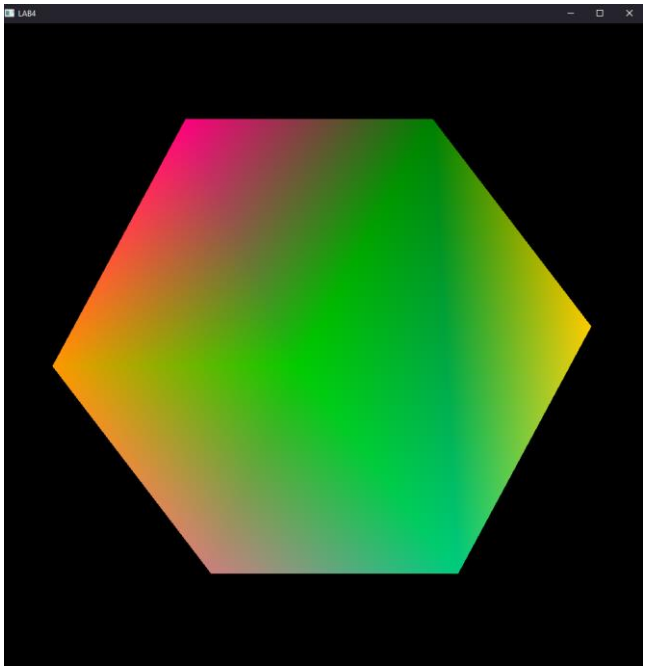
```
def main(transformation):

    vertex_src = """
  #version 330
  layout(location=0) in vec3 aPos;
  layout(location=1)  in  vec3 aColor;

  out vec3 vColor;

  uniform mat4 transformation;
  void main(){
      gl_Position = transformation * 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, "LAB4", 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 = [
        -0.5,-0.5, 0.5,0.5,
        -0.5,0.5,0.5, 0.5,
```

```
        0.5,-0.5,0.5,0.5,
        -0.5,-0.5,-0.5,0.5,
        -0.5,-0.5,0.5,0.5,
        -0.5,-0.5,0.5, -0.5,
    ]

    colors = [
        0.0,0.0, 0.0,
        1.0, 0.8, 0.0,
        0.0, 0.5,0.0,
        1.0, 0.0, 0.5,
        0.8, 0.5, 0.5,
        0.0, 0.8,0.5,
        0.0, 0.8,0.0,
        1.0,0.6, 0.0,
    ]

    indices = [
        0,1,2,2,3,0,4,5,6,6,7,4,4,5,1,1,0,4, 6,7, 3,3, 2, 6,  5,6,2,2, 1,5,7,4,
        0, 0, 3, 7,
    ]

    bufferData = vertices + colors

    vertices = np.array(vertices, dtype=np.float32)
    bufferData = np.array(bufferData, dtype=np.float32)
    indices = np.array(indices, dtype=np.uint32)

    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_DRAW)

    element_buffer_object = glGenBuffers(1)
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
                 GL_STREAM_DRAW)

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

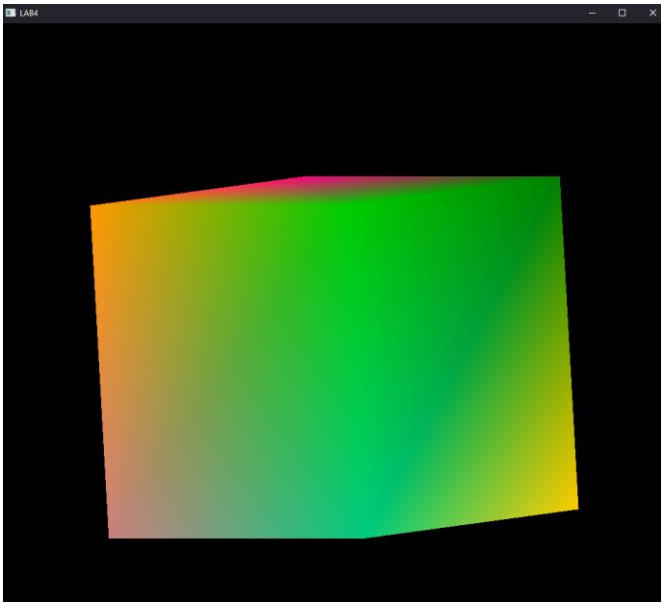
```
    glEnableVertexAttribArray(1)
    glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0,
                          ctypes.c_void_p(vertices.nbytes))

    glUseProgram(shader)
    transformation_location = glGetUniformLocation(shader, "transformation")

    glEnable(GL_DEPTH_TEST)

    while not glfw.window_should_close(window):

        glfw.poll_events()
        glClear(GL_DEPTH_BUFFER_BIT)
        glUniformMatrix4fv(transformation_location, 1, GL_FALSE,
                           transformation)
        # glUniformMatrix4fv(transformation_location,1,GL_FALSE,transformation)
        glDrawElements(GL_TRIANGLES, len(indices), GL_UNSIGNED_INT, None)

        glfw.swap_buffers(window)

    glfw.terminate()


main(transformation)
```

4.3. Output

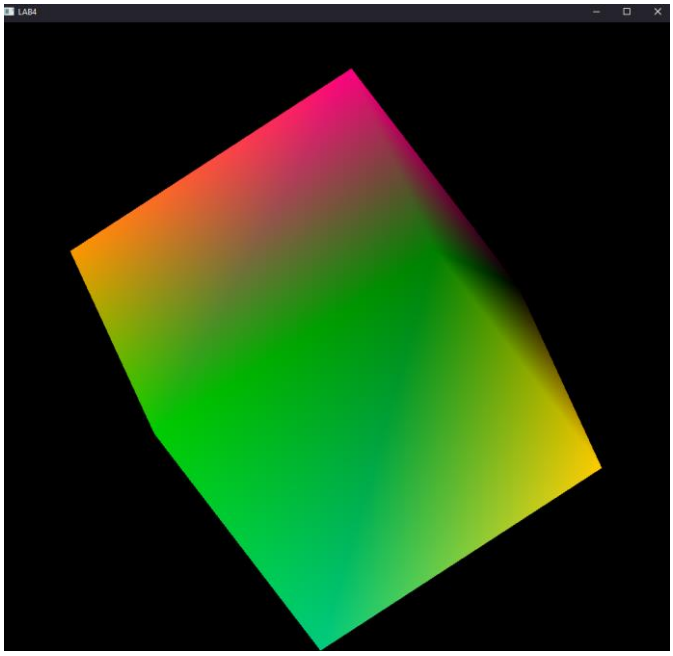
4.3.1. Original Cube



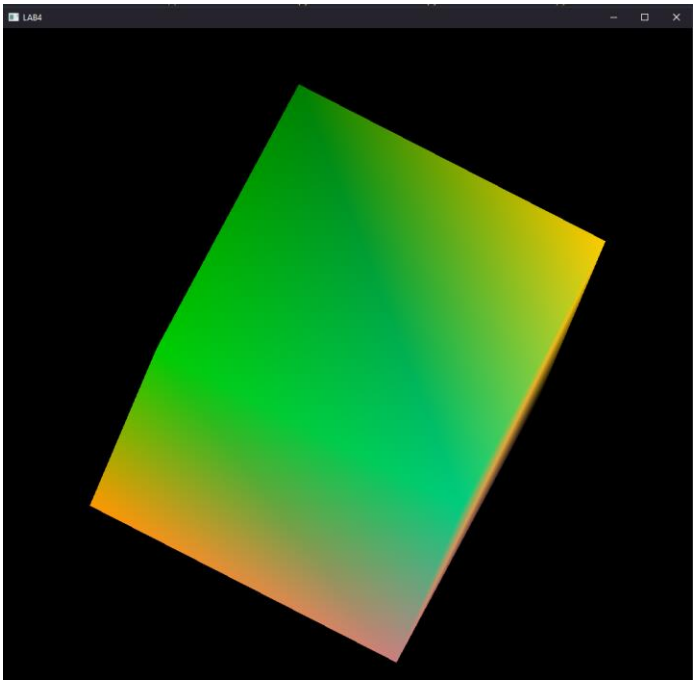
4.3.2. Rotation along x-axis of 45 degree



4.3.3. Rotation along y-axis of 45 degree



4.3.4. Rotation along x-axis of 45 degree



Chapter 5: 3D Scaling

5.1. Scaling Matrix

The scaling matrix is given as:

Sx	0	0	0
0	Sy	0	0
0	0	Sz	0
0	0	0	1

As discussed earlier, the above row major matrix should be converted in column major matrix by transposing, but as the original matrix and the transposed matrix are identical, above rotation matrix can be used to get required scale.

5.2. Source Code

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

from helpers import altList, toNVC


def window_resize(window, width, height):
    glViewport(0, 0, width, height)


RESOLUTION = 1000
transformY = np.matrix(f"""
{np.cos((np.pi / 180) * 40)},
0.0,
{np.sin(-(np.pi / 180) * 40)},
0.0;
0.0,1.0,0.0,0.0;
{np.sin((np.pi / 180) * 40)},
```

```
0.0,
{np.cos((np.pi / 180) * 40)},
0.0;
0.0,0.0,0.0,1.0""")

transformX = np.matrix(f""" 1.0,0.0,0.0, 0.0;
          0.0,
          {np.cos((np.pi / 180) * -50)},
          {np.sin((np.pi / 180) * -50)},
          0.0;
          0.0,
          {np.sin(-(np.pi / 180) * -50)},
          {np.cos((np.pi / 180) * -50)},
          0.0;
          0.0, 0.0,0.0,1.0""")
newTransform = np.matrix(f""" 2.0, 0.0, 0.0,0.0;
        0.0,1.0, 0.0,0.0;
        0.0,0.0,1.0,0.0;
        0.2,0.2,0.2,1.0""")

transformation = np.dot(newTransform, np.dot(transformX, transformY))


def main(transformation):

    vertex_src = """
  #version 330
  layout(location=0) in vec3 aPos;
  layout(location=1)  in  vec3 aColor;

  out vec3 vColor;

  uniform mat4 transformation;
  void main(){
      gl_Position = transformation * vec4(aPos,1.0);
      vColor=aColor;

      }
  """

    fragment_src = """
  #version 330
  in vec3 vColor;
  out vec4 FragColor;
  void main(){
```

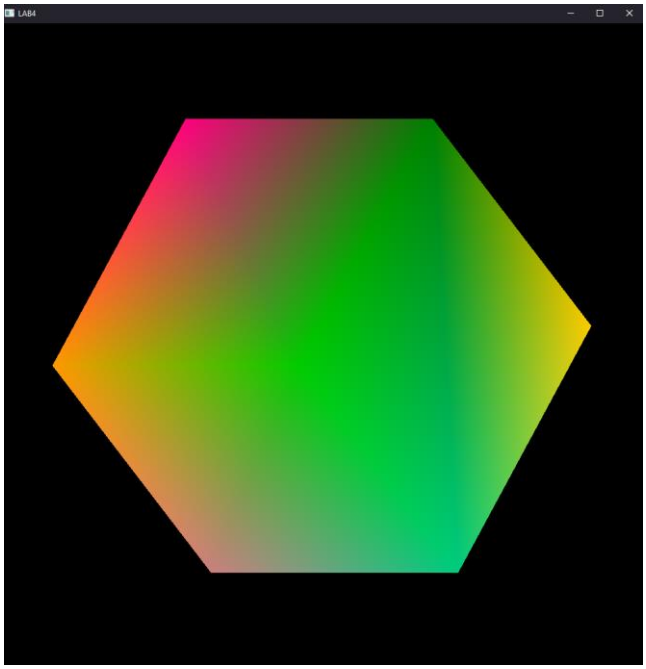
```
    FragColor = vec4(vColor,0.0f);

}
"""

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

  window = glfw.create_window(RESOLUTION, RESOLUTION, "LAB4", 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 = [
      -0.5,-0.5, 0.5,0.5,
      -0.5,0.5,0.5, 0.5,
      0.5,-0.5,0.5,0.5,
      -0.5,-0.5,-0.5,0.5,
      -0.5,-0.5,0.5,0.5,
      -0.5,-0.5,0.5, -0.5,
  ]

  colors = [
      0.0,0.0, 0.0,
      1.0, 0.8, 0.0,
      0.0, 0.5,0.0,
      1.0, 0.0, 0.5,
      0.8, 0.5, 0.5,
      0.0, 0.8,0.5,
      0.0, 0.8,0.0,
      1.0,0.6, 0.0,
  ]

  indices = [
      0,1,2,2,3,0,4,5,6,6,7,4,4,5,1,1,0,4, 6,7, 3,3, 2, 6,  5,6,2,2, 1,5,7,4,
      0, 0, 3, 7,
  ]

  bufferData = vertices + colors
```

```
vertices = np.array(vertices, dtype=np.float32)
bufferData = np.array(bufferData, dtype=np.float32)
indices = np.array(indices, dtype=np.uint32)

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_DRAW)

element_buffer_object = glGenBuffers(1)
glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, element_buffer_object)
glBufferData(GL_ELEMENT_ARRAY_BUFFER, indices.nbytes, indices,
             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)
transformation_location = glGetUniformLocation(shader, "transformation")

glEnable(GL_DEPTH_TEST)

while not glfw.window_should_close(window):

    glfw.poll_events()
    glClear(GL_DEPTH_BUFFER_BIT)
    glUniformMatrix4fv(transformation_location, 1, GL_FALSE,
                       transformation)
    # glUniformMatrix4fv(transformation_location,1,GL_FALSE,transformation)
    glDrawElements(GL_TRIANGLES, len(indices), GL_UNSIGNED_INT, None)

    glfw.swap_buffers(window)

glfw.terminate()
```

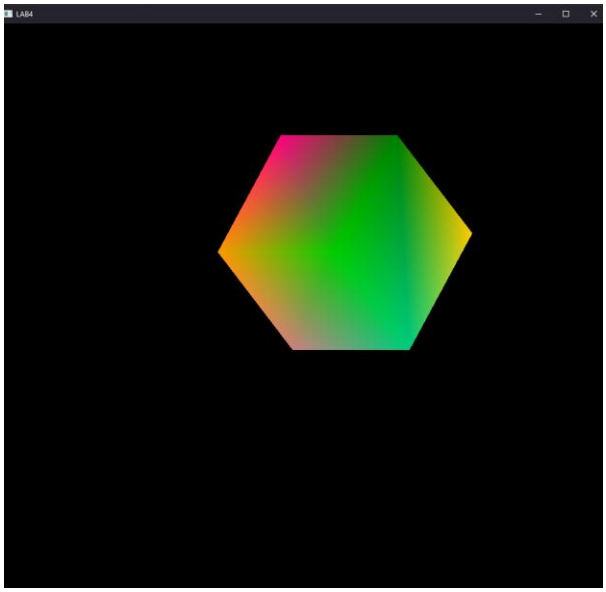
```
main(transformation)
```

5.3. Output

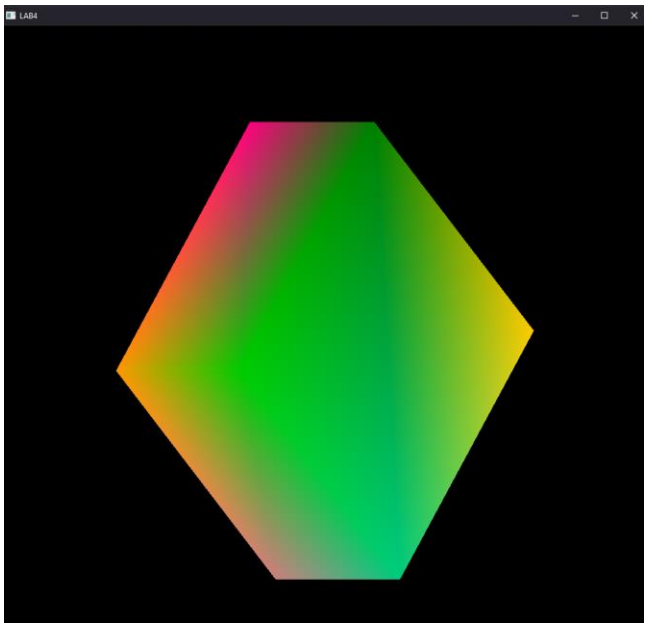
5.3.1. Original Cube



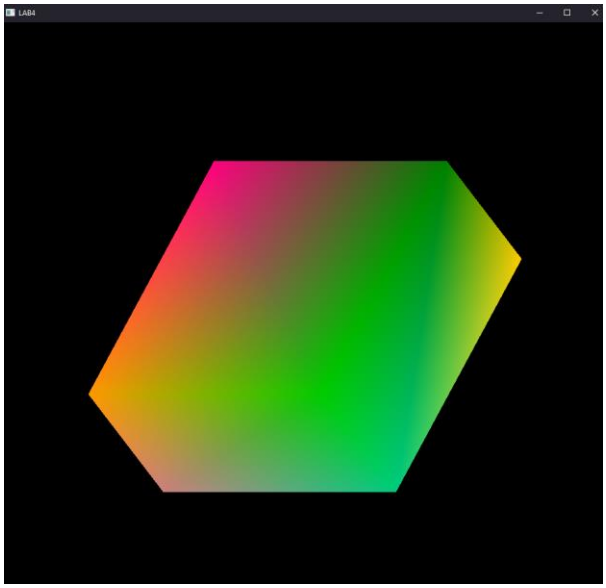
5.3.2. Scaled Cube with scaling factor 0.5



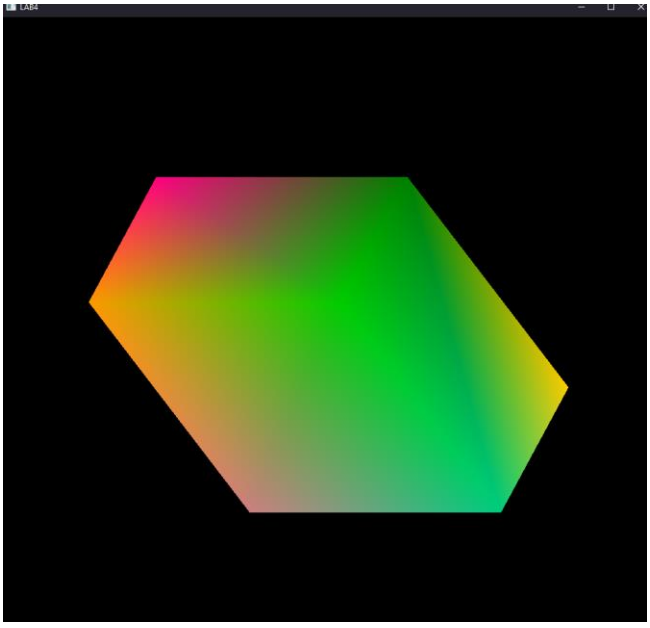
5.3.3. Scaled Cube with Scaling factor 0.5 along x-axis



5.3.4. Scaled Cube with Scaling factor 0.5 along y-axis



5.3.5 Scaled Cube with Scaling factor 0.5 along z-axis



Chapter 6: Conclusion

In this Lab work, various transformation matrix was used to achieve various transformations on 3D objects. The transformations that we implemented are: translations, rotation,and scaling The transformation matrix that we normally use cannot be directly sent to the program as GPU uses column major format of matrices, so the transposed matrix has been sent to the program.

Initially our vertices was defined as:

```
vertices = [
      -0.5,
      -0.5,
      0.5,
      0.5,
      -0.5,
      0.5,
      0.5,
      0.5,
      0.5,
      -0.5,
      0.5,
      0.5,
      -0.5,
      -0.5,
      -0.5,
      0.5,
      -0.5,
      -0.5,
      0.5,
      0.5,
      -0.5,
      -0.5,
      0.5,
      -0.5,
  ]
```

Then these vertices were transformed using our transformation matrix for the respective transformation operations.

Then, a uniform matrix transformation shader had been made, and the location of which was stored under transformation_location as:

```
transformation_location = glGetUniformLocation(shader, "transformation")
```

Finally, using the glUniformMatrix4fv matrix was transformed and GL_TRIANGLES was made by using glDrawElements.

```
glUniformMatrix4fv(transformation_location, 1, GL_FALSE, transformation)
```