Computer Graphics Assignment 1 Report

Daris Dzakwan Hoesien

1 Introduction

The main objective of this assignment is the usage of affine transformations, geometry rendering in Basic OpenGL rendering project with Python and the PyOpenGL library. All the Computer Graphic Task will be uploaded into these Github repository (https://github.com/darisdzakwanhoesien2/ComputerG

2 Overview

The project is split into two main files:

- `CG_assignment1.py`: Contains the rendering logic, data buffering, and object drawing (Task 1).
- `mvp_controller.py`: Manages the model-view-projection transformations, including camera controls (Tasks 2, 3, and 4).

Task 1: Rendering and Geometry Setup

In `CG_assignment1.py`, the geometry of a tetrahedron is defined using triangle vertices. Each triangle has randomized vertex colors stored in a separate buffer.

```
vertex_buffer_data = [ ... ]  # Triangle vertex coordinates
glBufferData(..., vertex_buffer_data, ...)
```

A shader pipeline is created, vertex and color buffers are created, and the object is rendered using `glDrawArrays(GL_TRIANGLES, ...)`. Shaders are implemented in terms of GLSL 120 to support macOS and attribute locations are bound manually due to lack of layout qualifiers.

After running the program, a snapshot of the rendered object is captured, showing a well-rendered, multi-colored tetrahedron.

Task 2: Affine Transformation (Model Rotation)

In the `calc_model` method of the

textttWin class, the model matrix is updated to constantly rotate the object around its X-axis.

```
elapsed = time.time() - start_time
angle = glm.radians(elapsed * 50)  # degrees/sec
self.model_matrix = glm.rotate(glm.mat4(1.0), angle, glm.vec3(1.0, 0.0,
    0.0))
```

This demonstrates the use of affine transformations (rotation matrix) and the

textttglm library to effect continuous object transformation in terms of elapsed time.

Task 3: Camera View Matrix and Euler Angles

In `mvp_controller.py`, the `calc_view_projection` ethod constructs the view matrix from Euler angles (yaw, pitch) in order to calculate the direction, right, and up vectors:

```
direction = glm.vec3(
    glm.cos(pitch) * glm.sin(yaw),
    glm.sin(pitch),
    glm.cos(pitch) * glm.cos(yaw)
)
self.front = glm.normalize(direction)
self.right = glm.normalize(glm.cross(self.front, world_up))
self.up = glm.normalize(glm.cross(self.right, self.front))
```

These vectors are then used to create the view matrix via `glm.lookAt(position, position + direction, up)`. Mouse movement is handled by the `on_mousemove` function to update `yaw` and `pitch`, allowing for free camera rotation when the left button is held down.

Task 4: Keyboard Input and Camera Movement

Camera movement is handled via the `on_keyboard` function in `mvp_controller.py`, with WASD-style controls:

```
if key == b'w': self.position += self.direction * self.speed
elif key == b's': self.position -= self.direction * self.speed
elif key == b'a': self.position -= self.right * self.speed
elif key == b'd': self.position += self.right * self.speed
elif key == b'e': self.position += self.up * self.speed
elif key == b'r': self.position -= self.up * self.speed
```

This allows the user to move about the scene in 6 degrees of freedom freely.

Conclusion

This project demonstrates the pieces of a 3D graphics program:

- Drawing geometry with OpenGL
- Applying affine transformations
- Developing a fully controllable 3D camera with mouse and keyboard interaction

All of the tasks were successfully completed, and a dynamic and interactive 3D visualization using Python and modern OpenGL conventions was developed for use in macOS systems.