CG Programming Assignment 2 Report

Daris Dzakwan Hoesien

April 16, 2025

1 Introduction

This assignment explores key features of modern OpenGL (3.1+), such as geometric rendering, texture mapping, external object loading, and shader programming. The goal is to render a 3D cube, texture it, manage its rendering using shaders, and load an external model.

2 Scene Initialization

OpenGL context is established using GLUT. The ‘Win‘ class inherits from an abstract base class that sets up rendering callbacks and input processing. Depth test and face culling are enabled to render 3D geometry correctly.

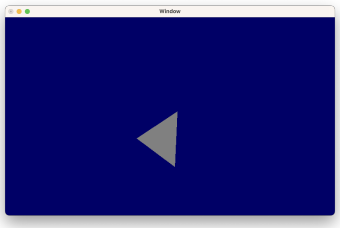

Figure 1: Default rendered scene before geometry or texture was added

3 Camera Control

The ‘MVPController‘ controls view and projection matrix computation using keyboard (WASD) and mouse input. The camera direction vector is calculated from pitch and yaw angles, and the matrices are recalculated with each interaction to update the view.

4 Drawing a Cube with Triangles

The cube is drawn with 12 triangles (two for each face), using hardcoded vertex coordinates. The coordinates are carefully laid out in counter-clockwise order for proper face orientation and lighting.

Vertex Layout:

```
vertex_buffer_data = [
    # Front face
    -1.0, -1.0, 1.0,
     1.0, -1.0, 1.0,
     1.0,  1.0, 1.0,
     1.0,  1.0, 1.0,
```

```
    -1.0,  1.0, 1.0,
    -1.0, -1.0, 1.0,
    ...
]
```

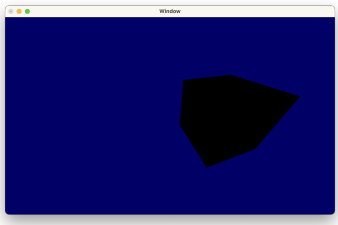

Figure 2: Manually defined 2x2x2 cube (12 triangles)

5 Mapping Texture to the Cube

Each triangle was mapped to a subregion of a texture image. The vertices' order is matched with the texture coordinate buffer (uv_buffer_data) to ensure correct alignment.

```
uv_buffer_data = [
    # Front face
    0.0, 0.0,  1.0, 0.0,  1.0, 1.0,
    1.0, 1.0,  0.0, 1.0,  0.0, 0.0,
    ...
]
```

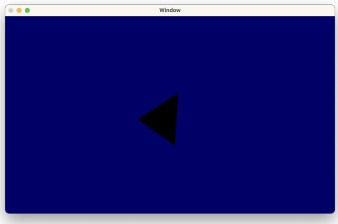

Figure 3: A single triangle with texture mapping applied correctly

6 Rendering a Textured Cube

After mapping UV coordinates for all sides, OpenGL renders the fully textured cube. The texture is reused for all six faces by splitting it into subregions.

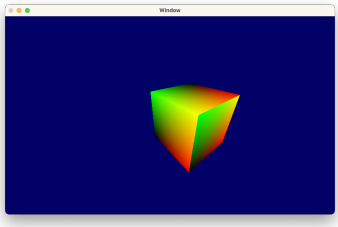

Figure 4: All 6 faces textured correctly with UV mapping

7 Loading External Object

Using the ‘ObjLoader‘, the cube is rendered from a Wavefront ‘.obj‘ file. The loader parses the file and provides vertex and UV buffers that are loaded into OpenGL.

```
obj = ObjLoader("resources/cube.obj")
obj.load_model()
glBufferData(GL_ARRAY_BUFFER, len(obj.vertices) * 4, ...)
glBufferData(GL_ARRAY_BUFFER, len(obj.uvs) * 4, ...)
```

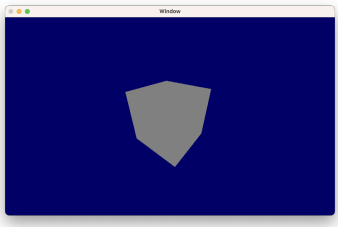

Figure 5: External cube model loaded from OBJ file

8 Shader Changes

Two fragment shaders were implemented:

1. A psychedelic shader that uses sine functions over time to modulate RGB values.
2. A color-inverting shader that flips the RGB values.

Psychedelic Shader:

```
vec3 psychedelicColor = vec3(
    sin(fragmentColor.r * 3.14 + time) * 0.5 + 0.5,
    sin(fragmentColor.g * 3.14 + time * 1.5) * 0.5 + 0.5,
    sin(fragmentColor.b * 3.14 + time * 2.0) * 0.5 + 0.5
);
gl_FragColor = vec4(psychedelicColor, 1.0);
```

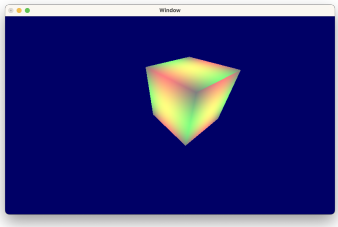

Figure 6: First shader effect with pulsating psychedelic tones

Color Inversion Shader:

```
gl_FragColor = vec4(1.0 - texture2D(texture_sampler, UV).rgb, 1.0);
```

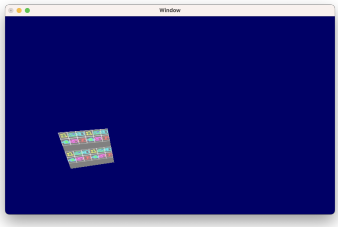

Figure 7: Second shader effect (color inversion)

9 Conclusion

Throughout the process of completing this assignment, I gained hands-on experience in understanding core OpenGL principles and pipeline operations. Starting with rendering a hand-crafted 3D cube through vertex buffers, I explored how geometric data is passed through the graphics pipeline. This exercise helped to solidify my understanding of Model-View-Projection (MVP) transformations and their influence on object position and perspective within a 3D space.

In addition, the difficulty in mapping a single texture to six faces of a cube highlighted the importance of assigning correct UV coordinates. Managing vertex and texture buffers with care was crucial in generating accurate visual output. Transitioning from fixed geometry to loading an external object via the OBJ format once again challenged my understanding of dynamic model handling and buffer generation. The most insightful aspect of the assignment was to employ and experiment with hand-coded custom GLSL fragment shaders. By experimenting with time-based psychedelic coloring and color inversion, I witnessed at first hand how manipulations on the fragment level can radically alter the appearance and visual mood of a scene. This opened up possibilities for more expressive and innovative rendering techniques.

In general, this project gave me a broad overview of necessary OpenGL functionality and reinforced my knowledge of both the theoretical and the practical underpinnings of modern graphics programming.