COSC 4370 - Homework 4

Name: Cyrus Shekari **PSID:** 2042446

April 2023

1 Objective

The program aims to produce a rotating numerical textured cube object using the OpenGL graphics library and 3D viewing techniques. To achieve this, the program would need to calculate the view matrix and projection matrix to properly view the object from the camera's perspective. The program would also need to transfer the uv data to the OpenGL buffer. Additionally, the program would need to bind the texture in the rendering loop and shader code to draw the texture. By combining these techniques, the program can create a realistic and visually appealing rotating numerical textured cube.

2 Method

To achieve the rotating textured cube, several key components had to be implemented, including the setup of both the UV buffer and the projection matrix in main.cpp, and both the vertex and fragment shaders in texture.vs and texture.frag, respectively. The projection matrix projects the 3D model onto a 2D screen, with *glm::perspective()* used to specify the field of view, aspect ratio, and near and far clipping planes. The vertex shader, texture.vs, calculates the position of the vertex in screen coordinates and passes the necessary variables to the fragment shader. It also sets the UV texture coordinate variable. In the fragment shader, texture.frag, the entry point of the shader is created.

3 Implementation

The implemented program encompasses several programs and functions that work together to generate the desired texture cube. These programs and functions include *glm::perspective()*, *texture.vs, and texture.frag.*

3.1 glm::perspective

The glm::perspective() function creates a projection matrix for the texture cube. The first argument is the field of view angle in radians. The second argument is the aspect ratio, which is the width of the viewport *(GLfloat(WIDTH))* divided by the height of the viewport *(GLfloat(HEIGHT)).* The third argument is the distance to the near clipping plane. Any objects closer than this distance will be clipped and not drawn. Finally, the last argument is the distance to the far clipping plane. Any objects farther away than this distance will also be clipped and not drawn. However, before glm::perspective is implemented, the UV buffer must be set up. A new buffer is generated using the object name (UVBO) and memory is reserved for it with *glGenBuffers(1, &UVBO).* The generated buffer object (UVBO) is bound to the GL_ARRAY_BUFFER target, so that it can be used to store vertex attribute data in the *glBindBuffer(GL_ARRAY_BUFFER, UVBO)* function. Additionally, memory data is allocated and UV data is copied by the *glBufferData(GL_ARRAY_BUFFER, sizeof(uv), uv, GL_STATIC_DRAW)*.

3.2 texture.frag

This program contains the main function of the fragment shader and applies a texture to the texture cube. The *color = texture(myTextureSampler, UV)* retrieves the color of the texture at the specified UV coordinates (*UV*) and assigns it to the output variable color, which will set the color of the fragment being rendered. The texture function samples the texture bound to *myTextureSampler* at the specified texture coordinates and returns the resulting color.

3.3 texture.vs

This program contains the main function of the vertex shader and is used to transform and render the texture cube. The *gl_Position = projection * view * model * vec4(position, 0.99)* transforms the 3D vertex position by the projection, view, and model matrices to obtain the final 2D screen position of the vertex. The resulting position is assigned to the built-in output variable *gl_Position*. The *UV = -1 * vertexUV* flips the Y-axis of the 2D texture coordinate (*vertexUV*) and assigns it to the output variable *UV*, which will be used to interpolate the texture coordinates for the fragment shader.

4 Results

The result of the program is a rotating numerical textured cube. Each face of the cube contains a number ranging from 1 to 6, however, all of the numbers are used only once. The numbers are colored differently and the cube rotates counterclockwise to display them.