Richard Paredes
UHID: 1492535
COSC4370

Assignment 4

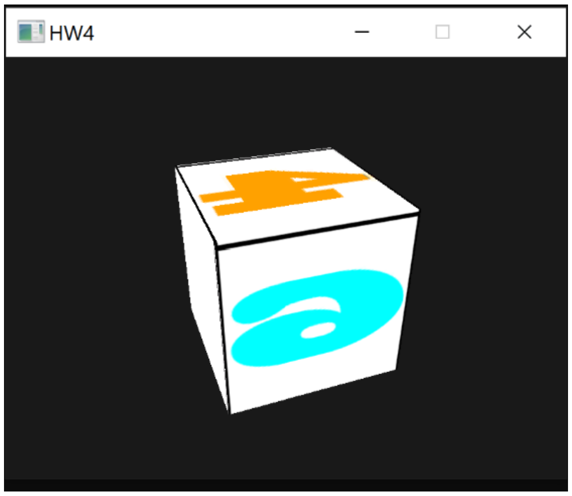
The purpose of this assignment was to become more acquainted with the process of texture mapping in OpenGL and shaders. The goal was to create a rotating cube with a numbered texture overlayed on top, similar to a dice. To perform this task, the fragment and vertex shaders needed to be changed, and the main program also needed to be updated to properly bind the imported texture.

After setting up the library dependencies and running the compiled program, the initial output was a black screen with a blank canvas. Therefore, using the knowledge from the previous assignment, the shaders needed to be updated to account for the cube in the view space so that it could properly be rendered. After updating the vertex shader with the position of the vertices of the cube based on the model, view, and projection matrices, the cube was finally rendering albeit without any color or texture. Adding a simple color vector, such as vec4(0.1, 1.0, 0.1, 1.0) would show a green cube, verifying that the issue is a missing texture and UV data for mapping.

The next step to adding the texture was to allow OpenGL to access the texture information by binding it. This was easily done using the glBindTexture function, which took in the type of texture, which was a 2D image, and an identifier pointing to the texture that was being used. Once this was set up, the output still did not render a textured cube. This is because the UV was missing.

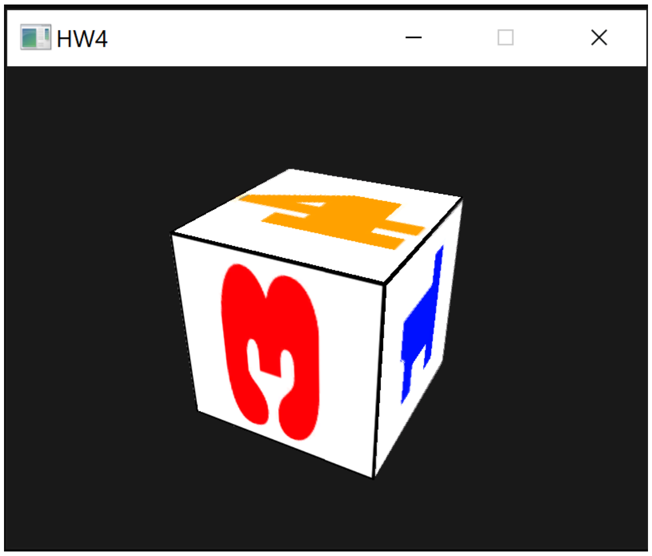
To allow OpenGL access to the UV data, the UV buffer needed to be configured and passed to the vertex shader. This was done by a series of functions: glGenBuffers, glBindBuffer, glBufferData, glVertexAttribPointer, and glEnableVertexAttribArray in sequence. Firstly, glGenBuffers tells OpenGL to establish a new buffer of data. glBindBuffer then binds the an array buffer to the pointer that was generated with the previous function. glBufferData then transfers the data from the uv static array containing uv coordinates into the generated buffer. Lastly, the glVertexAttribPointer and glEnableVertexAttribArray pass this UV buffer data to the vertex shader and enable it to be drawn during the main loop.

Once the UV positions are available in the vertex shader, they still needed to be passed to the fragment shader. This was easily done by setting the inputted UV data to the output UV data. Finally, once the data is available in the fragment shader, it could be used to map the textures onto the cube. This was done using the texture function, which took in the texture sampling and the UV data in order to map the corresponding colors at the appropriate positions on the cube.



The resulting cube was displaying the two sides 4 and 6 as expected in the guidelines. However, the remaining sides were completely blank, meaning the UV may have mapped incorrectly or overlayed the white portion of the texture on top of the other sides, masking it. To account for this, the UV buffer data needed to be directly updated.

Since the texture was compressed using inverted V texture coordinates, the previous UV buffer data was accounting for this inversion by subtracting the v-coordinate from 1.0 in the UV buffer for each position. Therefore, by reversing the inversion, the resulting texture mapping was fixed, as shown below.



However, by reversing the inversion of the coordinates, the numbers were also inverted on the sides, which caused the output to look slightly different from the guidelines. Nonetheless, the result is still more or less the same – a rotating cube with a texture mapped that shows a number on each side.