COSC 4370 – Homework 4

Name: Austin Thibodeaux PSID: 2037865

November 2022

1 Problem

The goal of this assignment is to render a simple textured cube using 3D viewing transforms. Boilerplate code is given to handle inputs and viewport display as well as setting up the cube for display, however the camera view transform, perspective projection, and shaders are left unfinished to be implemented.

2 Method

Four areas of the code needed to be updated to implement the requirements of this assignment:

- **GetViewMatrix function in Camera.h:** Creates a 4-dimension matrix using camera position data for the view of the camera to be used Phong.vs.
- **Projection matrix in main.cpp (line 198):** Calls glm::perspective to create a new 4-dimension matrix to be used in texture.vs.
- **Set up UV buffer in main.cpp (line 160-166):** Sets up the UV buffer for the vertex array.
- **Bind texture in main.cpp (line 210):** Binds texture for every frame
- **texture.vs:** Vertex shader. Sets up gl_Position with a perspective transformation chain including view and perspective matrix generated by the code implemented in the points above. Also sets up UV coordinates.
- **texture.frag:** Extremely simple fragment shader. Uses UV as input and a uniform texture sampler as an input and outputs the color of each pixel using UV data.

3 Implementation

3.1 GetViewMatrix

This implementation is fairly simple using the glm::lookAt function provided by the glm library. The first argument of this function is the current transform location of the camera, the second is the transform of the target that the camera will look at, and the third argument is the direction of up.

The camera should automatically spin around the cube. Thus, the view transform must always look at a point in space directly in front of the camera's front vector. Thankfully, the Camera class defined in Camera.h has two members: Position and Front. Position is the camera's current position in world space, and Front is the vector pointing in the direction of the front of the camera.

To implement the view matrix, the Position can be used as the first argument of lookAt. The second argument needs to be a place in *world space* that is directly in front of the camera which can be accomplished by adding Position and Front together and using the result as the second argument. The third argument uses the WorldUp member variable of the Camera which is the world direction of up.

This function then returns a 4-dimensional matrix representing the view of the camera.

3.2 Projection matrix in main.cpp (line 200)

This implementation is very simple as it makes use of the glm library's built-in perspective function. The function creates a new 4-dimension matrix for the perspective projection. The function has parameters for the field of view, aspect ratio, near plane and far plane. The resulting matrix is then stored into a variable called projection and then passed to the vertex shader.

3.3 Set up UV Buffer in main.cpp (line 160-166)

Code in this section is very similar to lines 147 to 157, except it deals with adding UV coordinates to the vertex array. The vertex array will contain the x,y,z coordinates (position) of each vertex as well as the u and v coordinates (UV map). This data will then be passed to the vertex shader to compute where the texture is drawn on the face of a triangle.

3.4 Bind texture in main.cpp (line 210)

This, I am not too confident in what this does exactly. If this line of code is not included the output is unchanged. I assume this is called every frame so it is possible to change the texture at runtime.

3.5 texture.vs

This is the vertex shader for the Phong implementation. This is where the model, view, and projection matrices are multiplied together along with the position of each vertex of the model. This part of the shader creates a flattened, perspective correct representation of the model to be viewed on a 2D screen. This matrix is then set to gl_Position.

This shader also outputs the UV coordinates of each vertex which is then used by vertex.frag to compute the color of the pixel. NOTE: The Y coordinate of the UV coordinates is inverted to fix an invalid result where the numbers display inverted (with numbers missing on some faces as well).

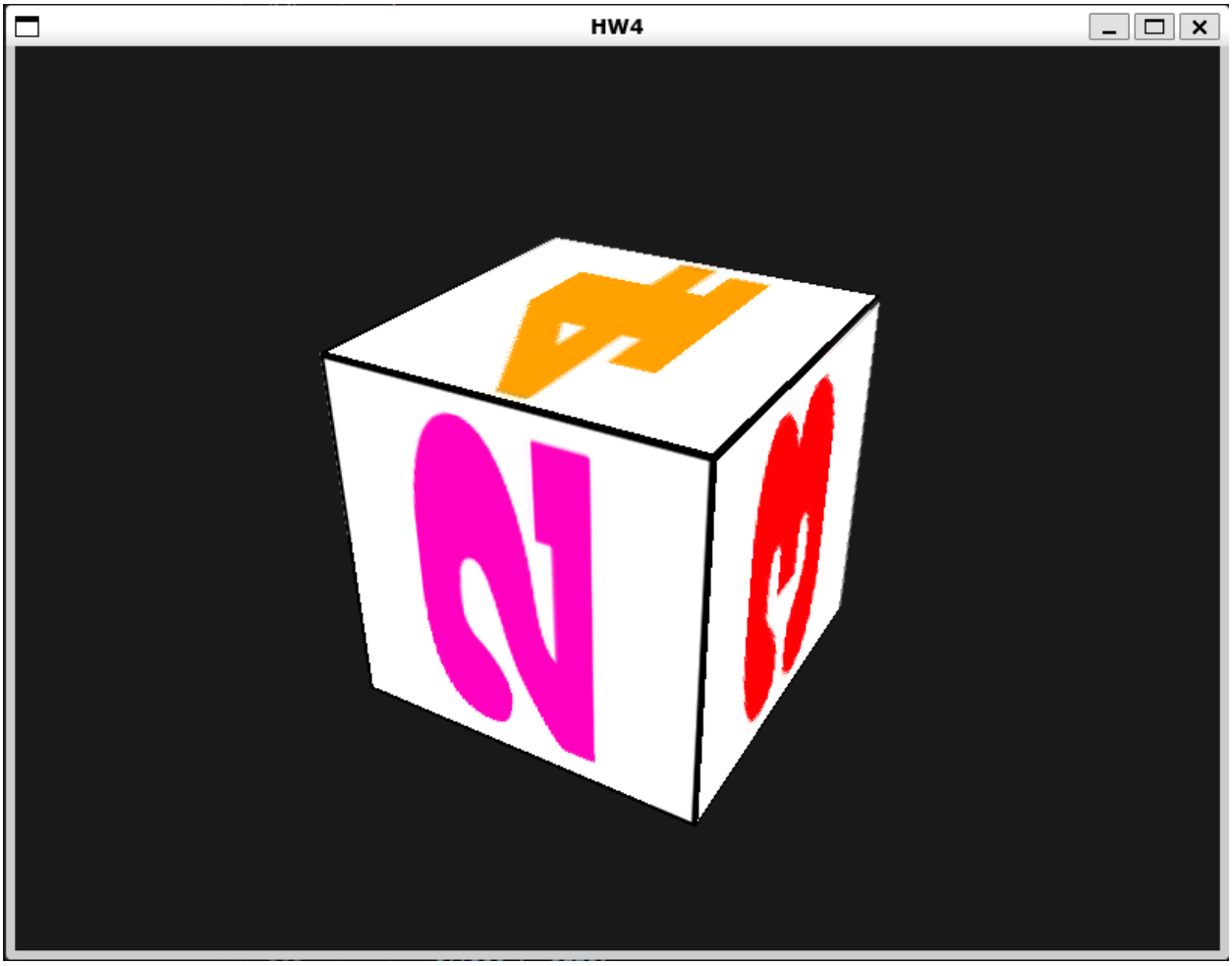
3.6 texture.frag

This is the fragment shader where the final color for each pixel is decided by the texture and UV maps. This shader is very simple with only one line in the main function. It simply uses the *texture* GLSL function where the texture sampler (passed in through a uniform variable) and the UV coordinates from texture.vs. The result of this function is set to the color output. The texture

GLSL function maps the UV coordinate to a pixel in the texture which is then sent as a color of the viewport pixel.

4 Results

When the program is run using script.sh (or run.sh that I created to run it outside of Replit), a window will appear showing a 3D cube that is textured with colored numbers on each side. The camera spins around the cube automatically.



6 References

Some of the source code in this assignment are derived from https://learnopengl.com. Page https://learnopengl.com/Getting-started/Coordinate-Systems was used as a guide for the view and perspective transformation implementation and https://learnopengl.com/Getting-started/Textures was used as a guide for the texturing implementation.