COSC 4370 - Homework 3

Rizwan Lokhandwala PSID: 1887820

October 2022

1 Problem

The main problem was to implement texture mapping in OpenGL. Given a set of texture and vertex coordinates, we needed to produce a rotating cube with the numbers 1-6 on each of the faces.

main.cpp, texture.vs, and texture.frag were to be modified.

2 Method: Displaying the cube

The first goal was to display the cube on screen. The vertices for the cube were already provided so all that was needed to display the cube was to write the projection matrix in main.cpp and the position function in texture.vs This was done in a similar fashion to HW3, so i will not go into specifics as they were discussed in the last report. But to put it simply, 108 data points were given, signifying 36 coordinates. These coordinates were groups of 3, each representing a triangle. OpenGL then uses these vertices to draw 12 triangles (2 for each face) with the functions

projection = glm::perspective(glm::radians(82.0f), (float)WIDTH / (float)HEIGHT, 0.1f, 100.0f);

and,

glPosition = projection * view * model * vec4(position, 1.0);

where view and model are given in main.cpp.

3 Method: Displaying the Numbers

With the cube now on the screen, we now had to display the numbers on the cube. In order to do this, we first had to set up the UV Buffer.

To reserve the buffer: glGenBuffers(1, &UVBO);

To create and bind the buffer object to the target(UVBO): glBindBuffer(GL ARRAY BUFFER, UVBO);

and to populate the buffer with data (texture coordinates): glBufferData(GL ARRAY BUFFER, sizeof(uv), uv, GL STATIC DRAW);

here, we are allocating enough space for uv (sizeof(uv)), we are then telling OpenGL which data points we need (uv), and then telling it that it will only be modified once, and used for the purpose of being drawn (GL STATIC DRAW).

We then specify the location and dataformat of the vertex attributes to be used in rendering: glVertexAttribPointer(1,2,GL FLOAT, GLFALSE, 2 * sizeof(float), (GLvoid*)(0));

the 1 represents which index of the pointer we will be modifying (in this case 1 since 0 is the index of the vertices, hence 1 will be the texture coordinates)

the 2 represents the number of points for each vertex, in this case 2 means it is a 2d texture.

GL FLOAT tells OpenGL that the values given are of type float.

GL FALSE then says that the points should not be normalized before use.

the 2*sizeof(float) signifies the stride aka the byte offset between each attribute.

and finally (GLVoid*)(0) tells OpenGL that the first component has no offset.

We then enable it: glEnableVertexAttribArray(1).

4 Method: Creating the Texture

Now that the UV buffer is set up, we need to actually create our texture.

This is done in texture.vs and texture.frag.

texture.vs receives information on the position and uv from main.cpp.

with uv cooridnates, we need to change it up a little bit before it can be used. As is, the numbers are inverted. In order to fix, we use the line:

UV = vec2(vertexUV.s, 1.0f - vertexUV.t);

UV then gets passed on to texture.frag so that it can create the texture.

In texture.frag, we need to define the color at each pixel. For this we use the function:

color = texture(myTextureSampler, UV).rgba;

With this the texture is complete and now just needs to be bound to the vertices (occurs in main.cpp)

5 Implementation: Binding texture to vertices

Now that the texture is created, we need to actually need to bind it to our vertices.

glEnable(GL TEXTURE 2D) glBindTexture(GL TEXTURE 2D, texture);

where texture is grabbed from texture.dds

glTexParameteri(GL TEXTURE 2D, GL TEXTURE MAG FILTER, GL NEAREST); glTexParameteri(GL TEXTURE 2D, GL TEXTURE MIN FILTER, GL NEAREST);

this tells OpenGL that we want to color the pixels when magnifying and minimizing images using the nearest pixel approach. This means we will see which color a pixel is closest to and light it respectively. This may leave an 8bit appearance when magnifying.

With that we have successfully bound the texture to the vertices and are ready to run the program.

6 Results

My results ended up looking very similar to the desired image. The color is the same, the faces have the proper numbers, however, in the pdf image, the numbers are inverted, but with the use of texture.vs, I fixed that.