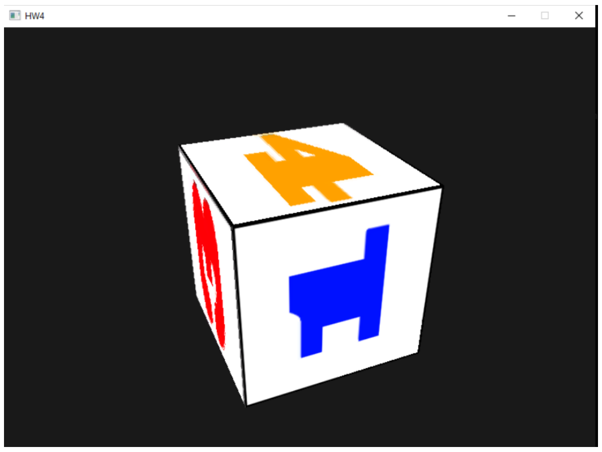
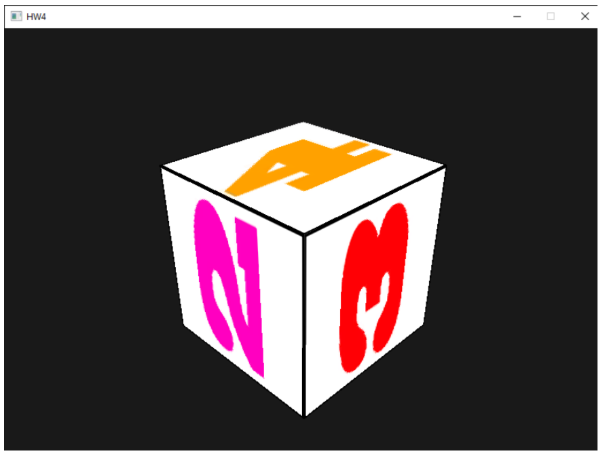
Report 4

Created	@November 10, 2022 1:13 PM
Class	Graphics
Semester	

Problem

The problem given to us was to recreate a cube that was textured on all sides (we can't really see the bottom, but assuming here) that looks like so:





As can be seen, the cube as numbers on all the faces. In the actual implementation, the cube rotates in order to view top and sides. Our goal is to produce a cube by mapping the texture onto the faces of the cube.

Method

Since we are dealing with textures, along with a cube create from vertices, we are exposed to two shaders: `texture.vs` and `texture.frag` . The role of `texture.vs` is to set up the transformation matrix in order to actually see the cube along with allowing the UV coordinates to be transferred to the fragment shader. The UV coordinates map the face the cube vertices in order to let interpolation take care of where the rest of the texture will lie.

The `texture.frag` takes these coordinates from the vertex shader and applies the texture as a ‘color’ to the defined shape.

Implementation

Texture.vs

```
void main(){
gl_Position = projection * view * model * vec4(position, 1.0);
UV = vec2(vertexUV.s, 1.0-vertexUV.t);
}
```

The inital step was to set up `gl_Position` to display anything. This was also done in `main.cpp` by completing the projection matrix like so

```
projection = glm::perspective(glm::radians(90.0f), (float)WIDTH/(float)HEIGHT, 0.1f, 100.0f);
```

Lastly, here UV, the vector data, will be passed to the fragment shader. Here `vertexUV.s` and `vertextUV.t` are each a vector of the UV coordinates.

Texture.frag

Here, that UV is data collected an input into the texture function and is set as a ‘color’ for the object/vertices. `myTextureSampler` is a uniform and is defined in `main.cpp` as the name of the texture to use in the fragment shader.

```
void main(){
color = texture(myTextureSampler,UV);
}
```

Main.cpp

In the `main.cpp` there are two main sections of note. Around lines 160 and on we have the following code:

```
glGenBuffers(1,&UVBO);
glBindBuffer(GL_ARRAY_BUFFER, UVBO);
```

```
glBufferData(GL_ARRAY_BUFFER, sizeof(uv), uv, GL_STATIC_DRAW);
glBindVertexArray(UVBO);
```

The purpose here is to set up the UV Buffer. That is, to tell the the `GL_ARRAY_BUFFER` the coordinates of the UVBO, that is the U and V axis (akin to the X Y axis) of the texture map.

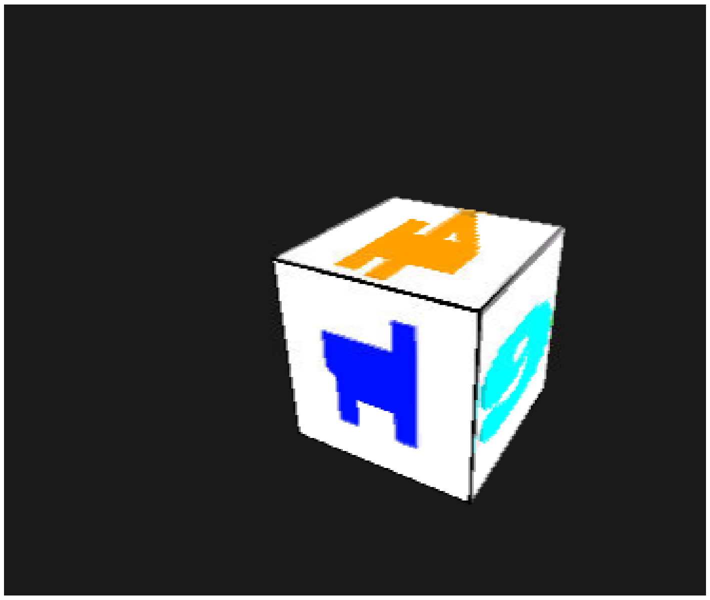
The second part is around line 200 where we have the this code:

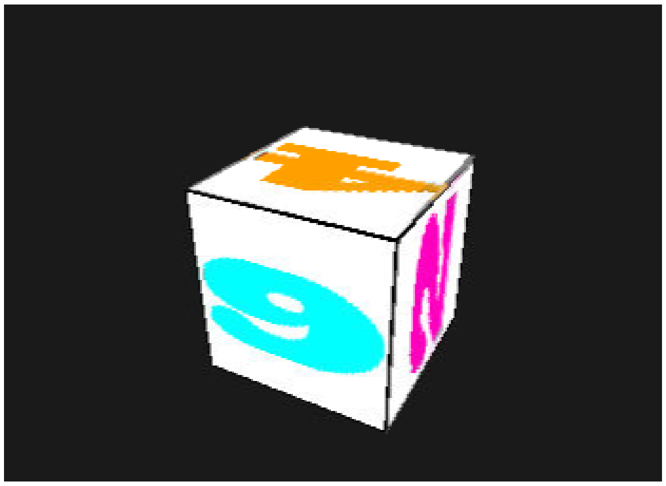
```
glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 2 * sizeof(float), (void*)(0* sizeof(float)));
glEnableVertexAttribArray(1);
```

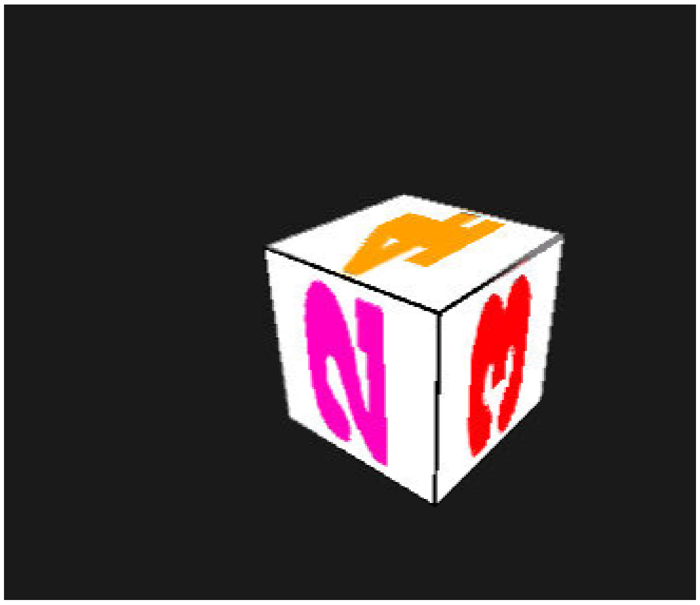
`glVertexAttribPointer()` takes in arguments that are defined by the texture map. Specifically the UV array of vertices. The arguments also dictate the offset where the points lie. Finally `glEnableVertexAttribArray()` allows the use of that array and those vertices.

Conclusion

The final result gets something like this. Where the cube is spinning and the numbers are plastered on the sides.




4
6
2


4
2
3