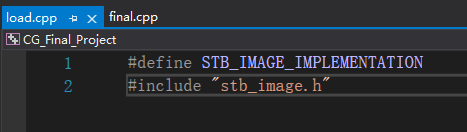
1. **Goals of the project**

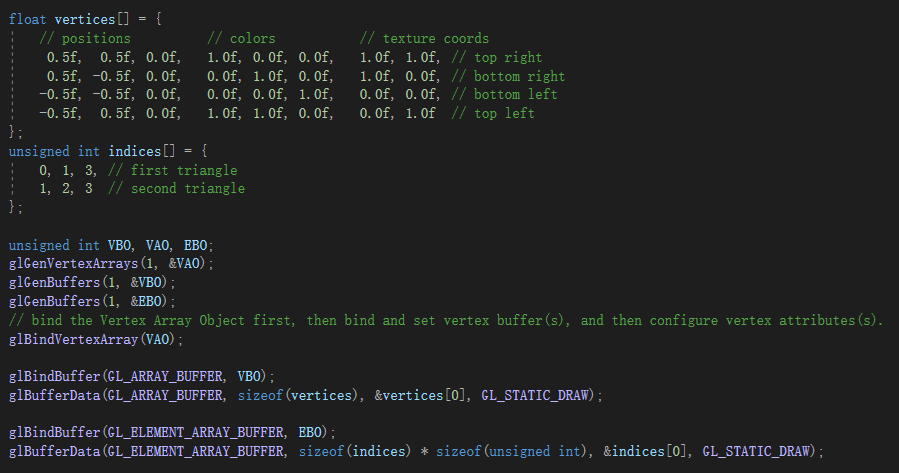
My goal of this project is to implement some techniques that were mentioned in class, so that I can improve my skills as well as extend my expertise in computer graphics. I implemented texture mapping to 2D and 3D objects, cube mapping，environment mapping and shadow mapping in the project.

1. **What have been done:**
2. Texture mapping:

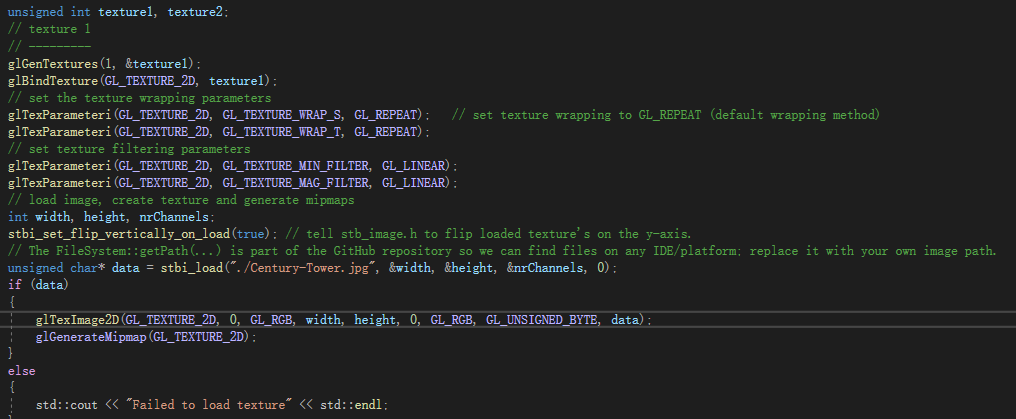
First, to load the texture pictures, I used the library “stb\_image.h”.



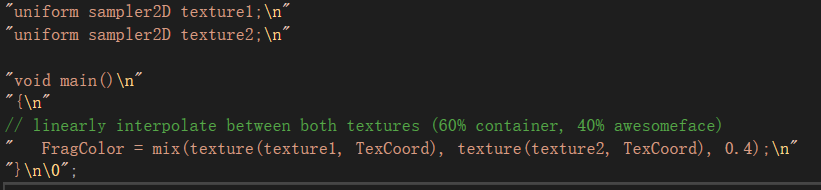
Then create a rectangle to render. And then create buffers and vertex arrays and bind them, this time I used an Element buffer object(EBO) to draw the object.

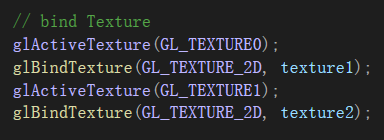


I used 2 pictures as my textures this time. First, I need to set the texture wrapping and texture filtering methods the textures and the load them:



Then set up the fragment shader and bind texture:



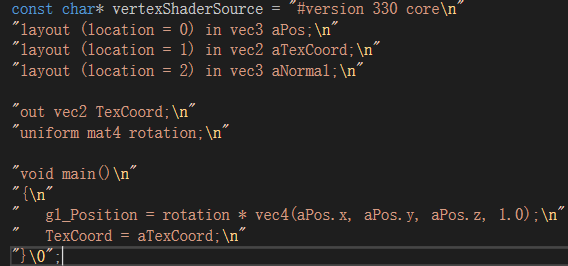


The texture showed will be 60% of texture1(The tower) and 40% of texture2 (UF logo):

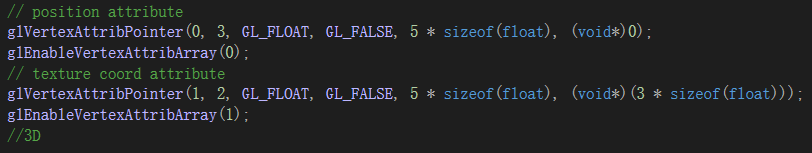


Then I used the loadOBJ() function that I build in project 2 to load the “head.obj” and map those 2 textures above to this 3D object.

First, I need to modify the vertex shader to use the vertices read from the obj file and to do some rotation:

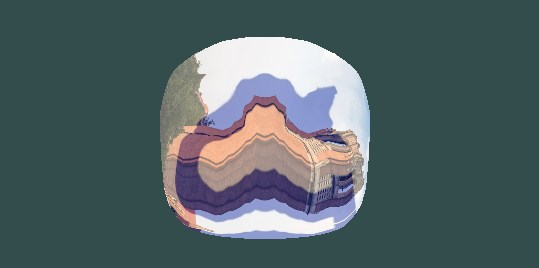


And also change the vertex attribute pointer and number of triangles that needed to be displayed:

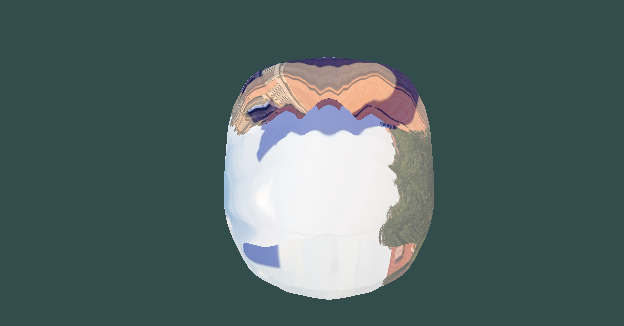




And here is how it looked like:



Looked a little bit weird, so I added a function to rotate the object around the x-axis base on the mouse input, and this is its effect:

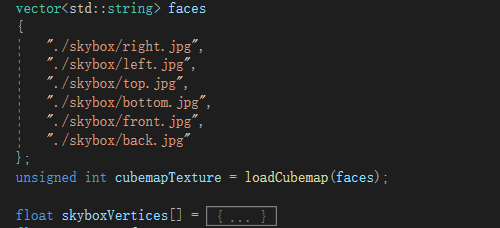




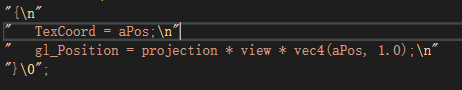
1. Cube mapping:

First, I built a “skybox”, which used 6 pictures as the background and could make the users to believe they are in a very large environment.

To begin with, I used a vector “faces” to store the path of those 6 pictures, then wrote a “loadCubemap” function to load them, and then set up the skybox’s vertices.



Then I modified the vertex shader, using the vertices’ position as texture coordinates (Because the skybox’s center is located in the coordinate’s origin, so its vertex coordinate is the texture’s coordinate). Then pass the “TexCoord” to the fragment shader and use the skybox texture.

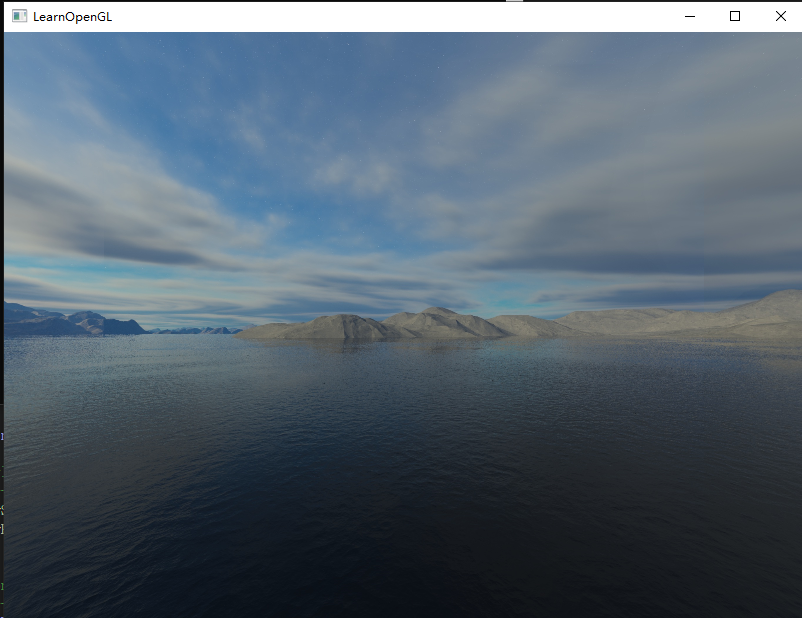




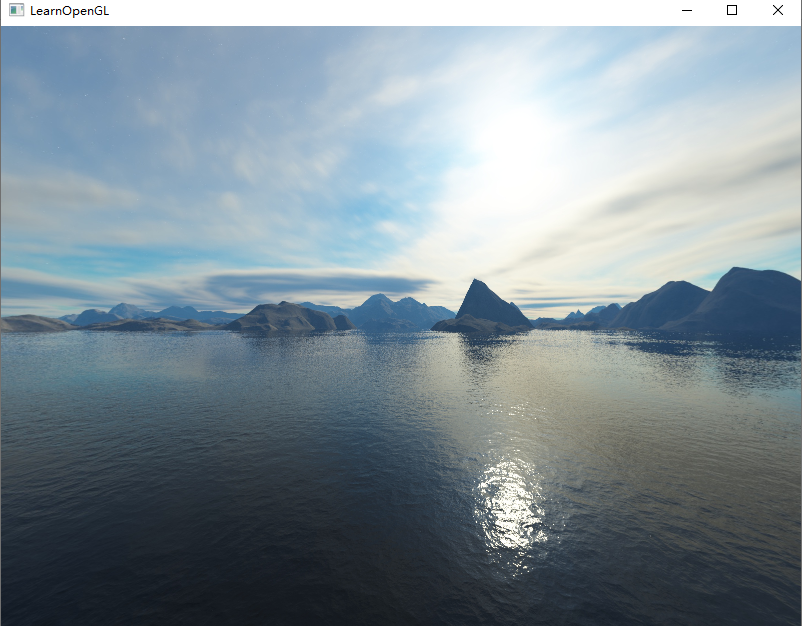
To view the whole skybox, I also add the function to change the camera’s look at direction based on the keyboard input (W, S, A, D):



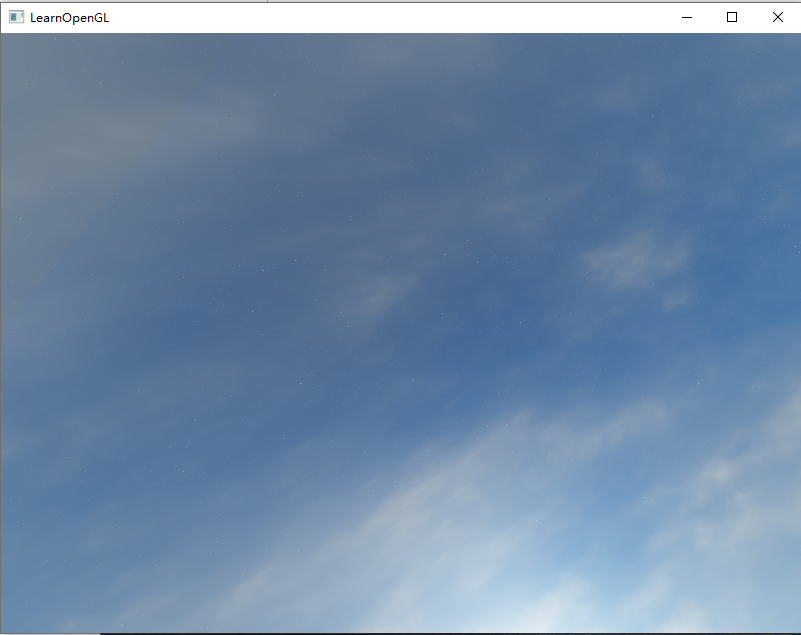
And here is how the skybox look like:



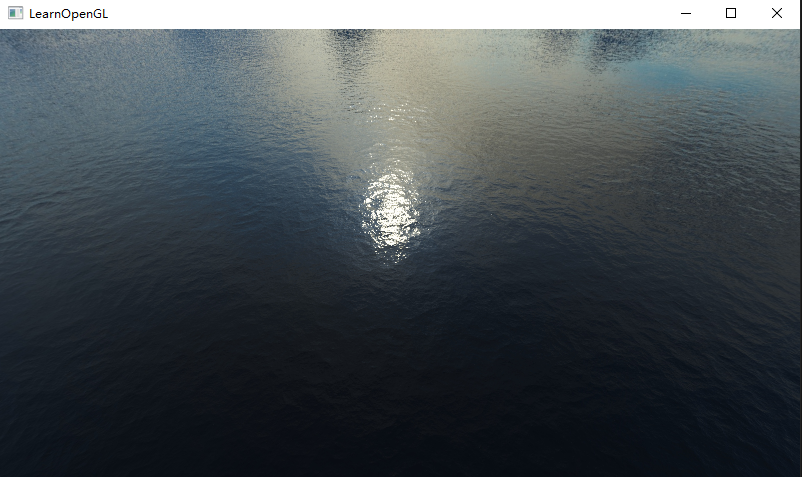
Press “D” for a while to turn around:



Press “W” for a while to look up:

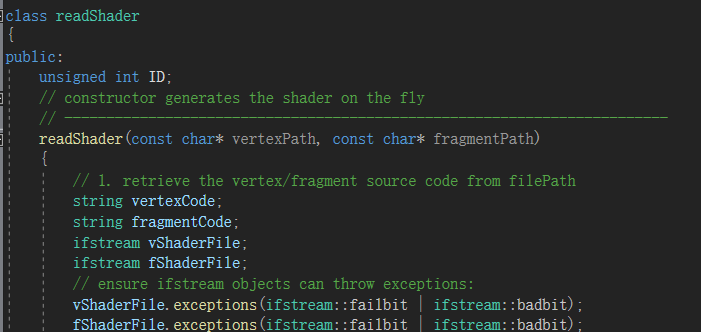


And press “S” for a while to look down:



Now we can combine texture mapping and cube mapping together, because just a simple skybox or a model is too tedious.

To do that, I changed the way to load shaders first. I used a class “readShader” to load vertex & fragment shader files instead of hardcoding those shaders every time:



Then use it:

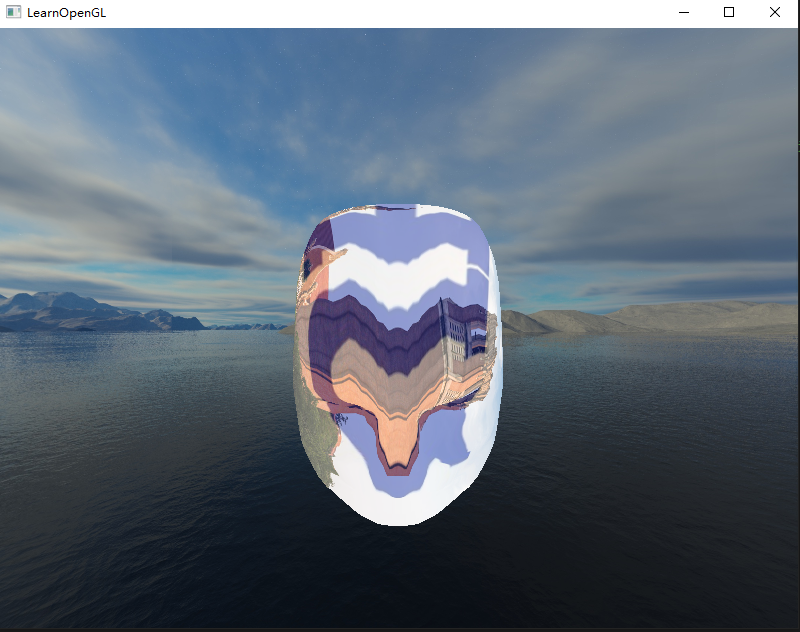


Then trick the depth buffer to make it believe the skybox has the maximum depth value of 1.0:





And this is how it looks like:

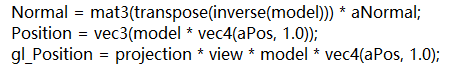




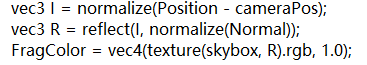
1. environment mapping:

First, I need 2 new shaders.

For vertex shader, interpolate the normal and position of the 3D model and pass the interpolate position coordinate to the fragment shader:



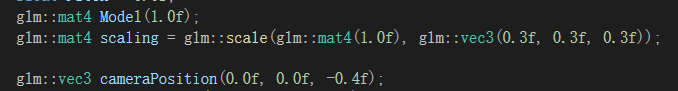
In fragment shader, compute the reflected vector according to the normal and camera’s viewing direction per fragment, and use that vector to decide the color of the object:



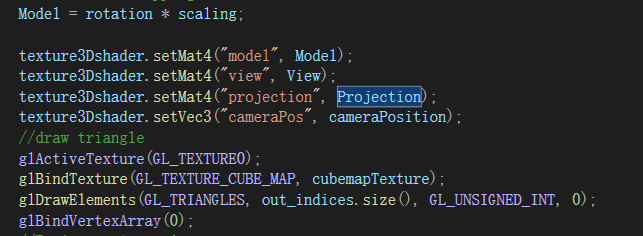
Use those 2 shaders:



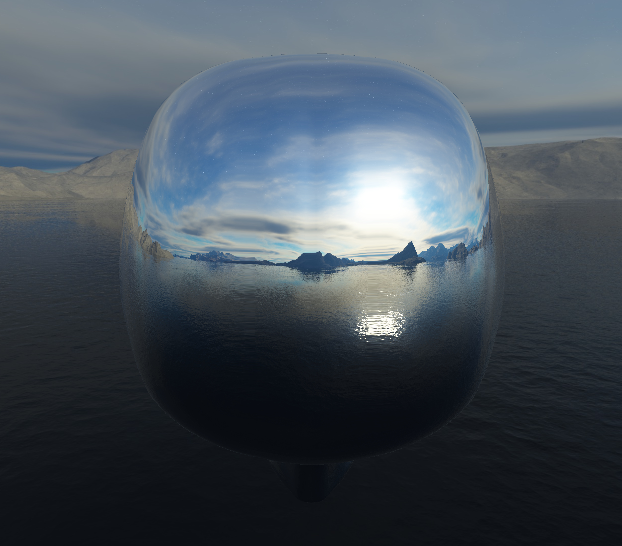
To better show the effect, I scaled the model and also changed the camera position:

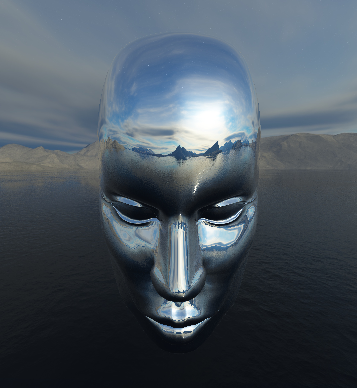


And then transmit values that those shaders need, bind textures, and then render the object:



And we can see that the skybox has successfully been reflected to the object:

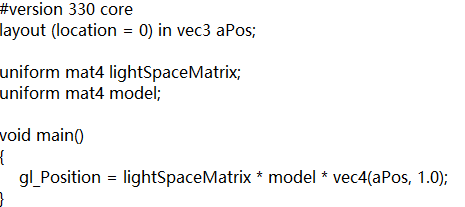




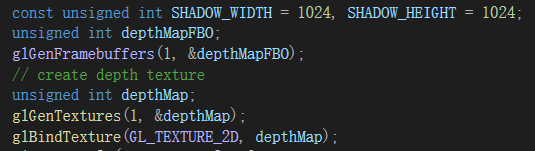


1. Shadow mapping:

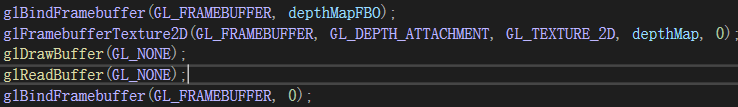
To implement shadow mapping, we first need a vertex shader to convert the vertices into the light source’s position to update the Z-buffer, and we do not need to do anything with the fragments because we don’t need colors, so the fragment shader will be empty:



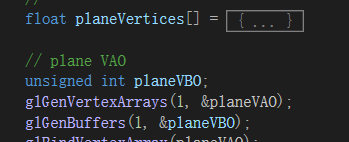
Then we’ll need a framebuffer object, and create a 2D texture as a depth map.



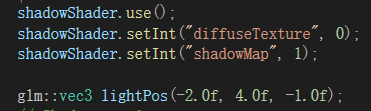
And use the generated texture as the framebuffer’s depth buffer. To complete the framebuffer object, we need to tell OpenGL we don’t need color data to render by setting the glDrawBuffer and glReadBuffer to be” GL\_NONE”:



We also need a plane to show the projected shadow:



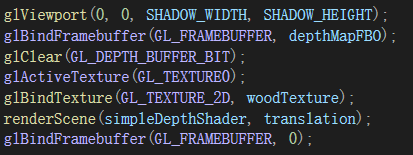
Configure shader and the light’s position:



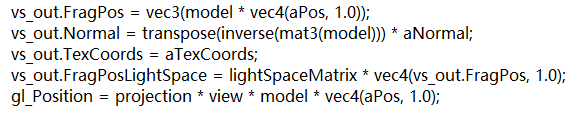
This time I used a directional light to generate the shadow:



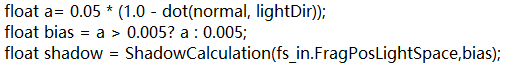
In the while loop, generate the shadow map:



And finally, we can use the generated shadow map to generate shadows. First, we need to configure the shaders. In the vertex shader, we just need to transform the vertex position into the light source’s space, and interpolate normal, and pass those values to the fragment shader.



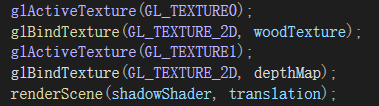
In fragment shader, use the Blin-Phong illumination model and calculate the shadow based on the shadow map. I also used the shadow bias technique to avoid shadow acne, to compute “bias” according to the angel between the light direction and the surface normal and use it:



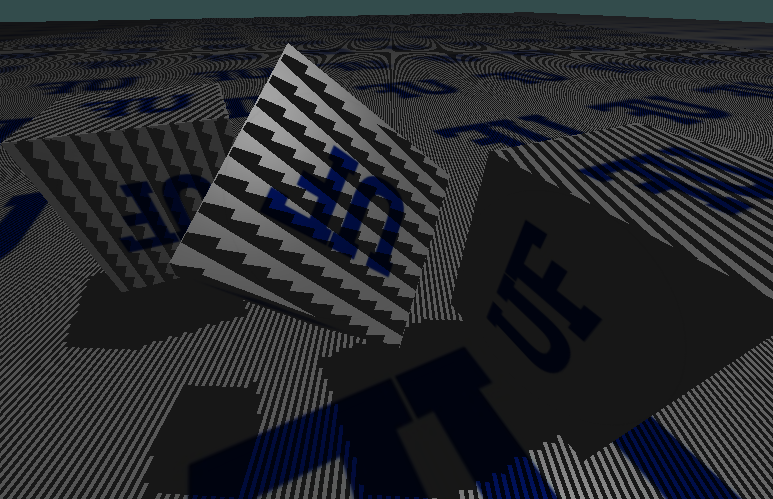


Bind the shadow map and render the scene in the while loop:



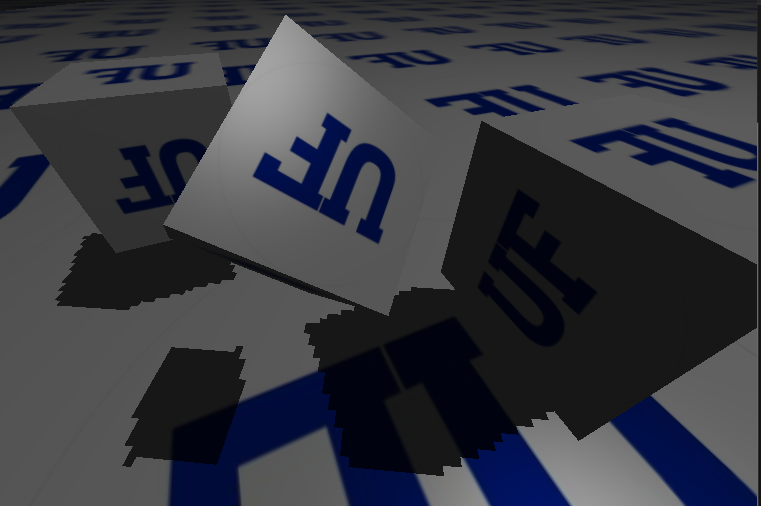


I render 3 boxes and a plane, in the scene, and this is what it looked like without using shadow bias:



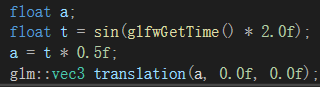
We can see that there are lots of shadow strips in the scene, which is very unrealistic.

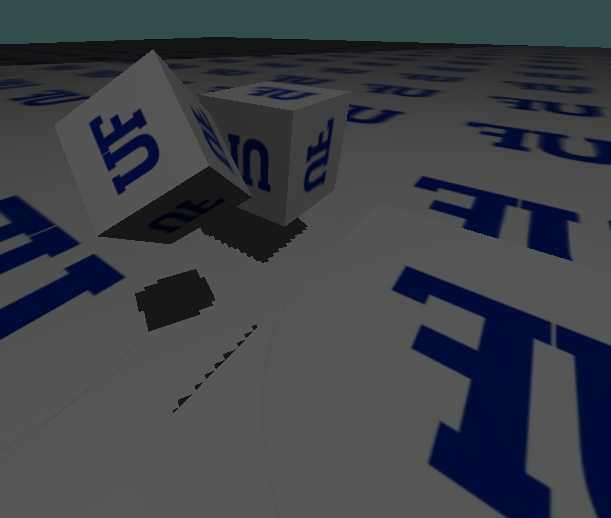
And this is how it looked like after using the bias technique:

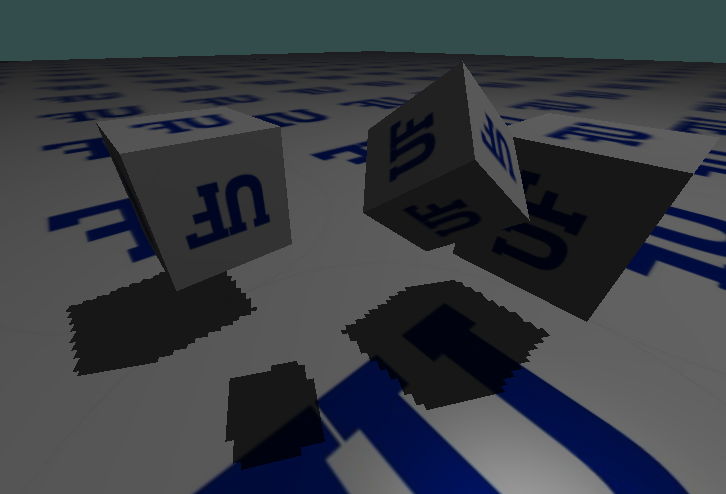


Now the scene looked perfect, it doesn’t have any black strips anymore, except for our low quality shadow, which suggested some aliasing issues.

I also added a translation vector, it will shift the objects’ position horizontally back and forth, so that we can have a dynamic scene and it will be funny to watch.

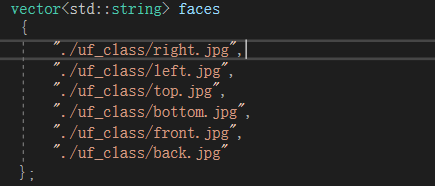


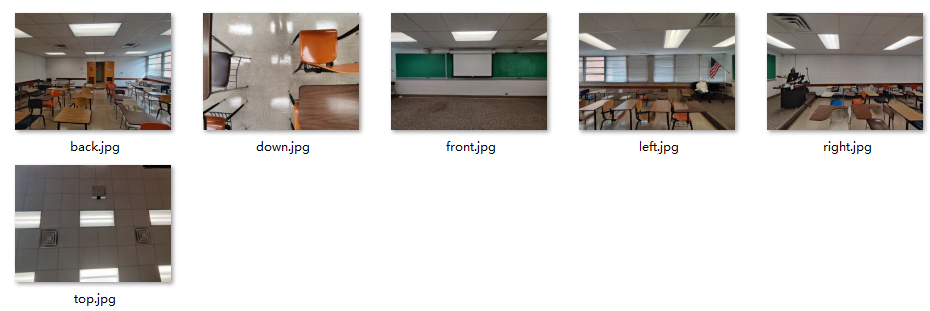


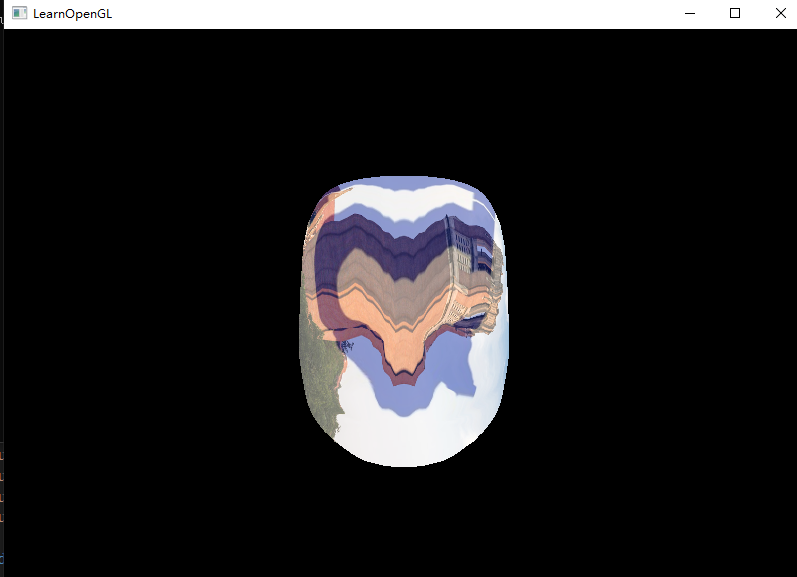


**What didn’t do:**

I actually took six photos in our classroom and tried to implement cube mapping using those photos. But unfortunately, when I changed the path from the skybox to my own photos, OpenGL didn’t give me the ideal result, and instead the background is completely black:







But if I load those photos separately as 2D/3D textures, it worked pretty well. I did some research and found that skybox cannot use some random pictures, it has some rigid requirements of them, they’d better be generated by some 3D modeling software. So, I downloaded Autodesk 3D max 2022 and tried to make my own skybox. But very unfortunately, somehow it didn’t provide the “Standard material” and “diffuse mapping”, and thus I uninstalled it and delete some files manually. And then when I tried to install the 2020 version of it, some errors happened and it cannot be installed and I couldn’t afford the time to reset my laptop, thus I gave up making the skybox of our classroom. It was such a pity.

1. **Learning outcomes**

Through this project, I learnt a lot of graphics ideas and techniques, including texture mapping, cube mapping, environment mapping and shadow mapping. I learnt how to map image textures to 2D and 3D objects, how to create a skybox & how to render it correctly, how to map the environment into objects, and how to create a shadow map and use it to generate shadows in the scene.

1. **Thoughts on future work**

In the cube mapping & environment mapping sections, I can use some 3D processing software to generate 6 pictures of my own skybox so that I can customize the scene to achieve practical goals. I’ll be able to take pictures from the scenes that is required by the future projects and map those scenes into different objects to achieve amazing effects and satisfy practical industrial needs.

In the shadow mapping section, there are aliasing issues with my generated shadow, so I can explore ways of shadow’s anti-aliasing in the future, so that the sawtooth of the shadow could be eliminated or at least be smoothed out a little bit. Perhaps I can used what we’ve learned in signal processing, using a filter to resample the shadow and smooth it out.

1. **Presentation**

https://youtu.be/eywgGJr5HA4