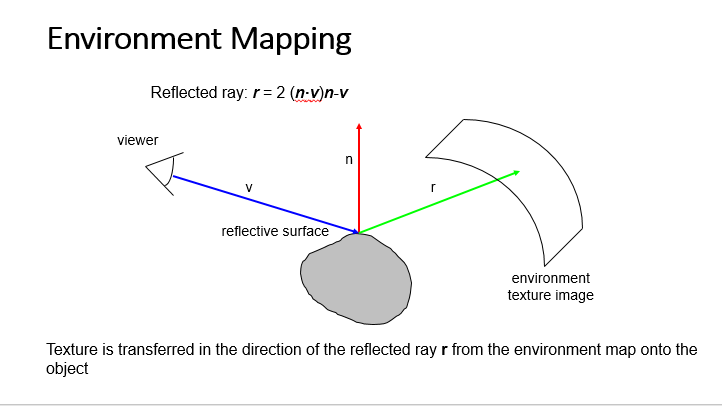
**Cube Mapping**

**Introduction**

This reports purpose is to demonstrate what cube mapping is, and how to implement it, along with a simple reflect and refraction Shaders.

**Theory**

Cube mapping is a method of environment mapping, that uses the six faces of a cube as the shape for the map, but firstly I will talk about environment mapping [1]. Environment mapping, is an efficient technique for approximating the appearance of a reflective surface by computing a texture onto the said object.



[2] figure A – explaining what environment mapping sets out to do

Environment mapping uses the reflection approach to mapping its surrounding, which is much more efficient than the classical ray tracing, due to its ability to simply work out the reflection formula, compared to its counter part of calculating the exact reflection by following a ray.

It however has some down sides, those being:

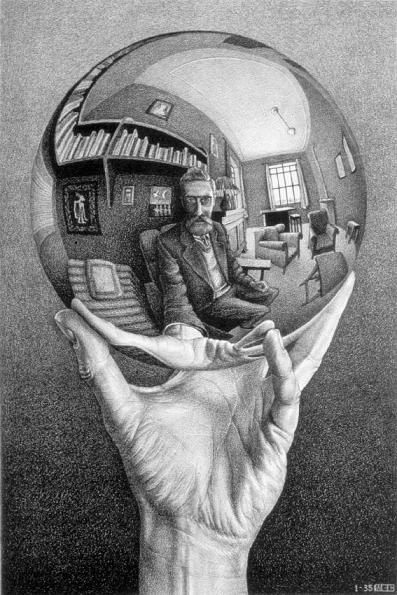
All radiance incident upon the object being shaded comes from an infinity distance [3]

The object being shading by this method is convex, which means it contains no self-interreflections, it only reflects the environment, which isn’t realistic.

Environment mapping has several ways of storing the information of the surrounding environment, the two I will talk about are sphere mapping, and cube mapping.

**Sphere Mapping**

Other than cube mapping there is sphere mapping, which is a simple and directly supported feature.

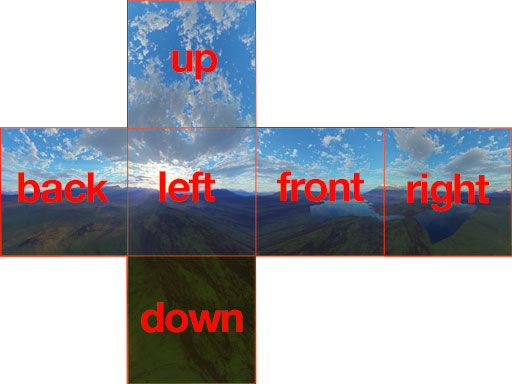


Which as you can see depicted by the picture, produces a spherical image of the surrounding environment.

it however has a few down sides, those being that the mapping technique is view-dependent, so each eye position will need a different sphere map texture. Furthermore, the mapping is subject to ‘speckle’ artefacts when glancing at the edges of sphere mapped objects.

**Cube Mapping**

Cube mapping is vastly superior and entirely surpasses sphere mapping. It projects the environment onto six faces of a cube, which are stored as six square texture. The cube map is generated by rendering a scene six times from the same viewpoint, with the views defined at a 90-degree view frustum [4]



By Creator:Arieee - Own work, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=15456566

This is an example of the cube mapping, as we can see the environment is split up into 6 pieces, which each representing a 90degree angle from a viewpoint, and once folded together will create a surrounding image.

The front being where we would be looking straight at, and then left and right being to either side, and folded behind us, as well as on top/below us will be the back/up/down parts of the picture.

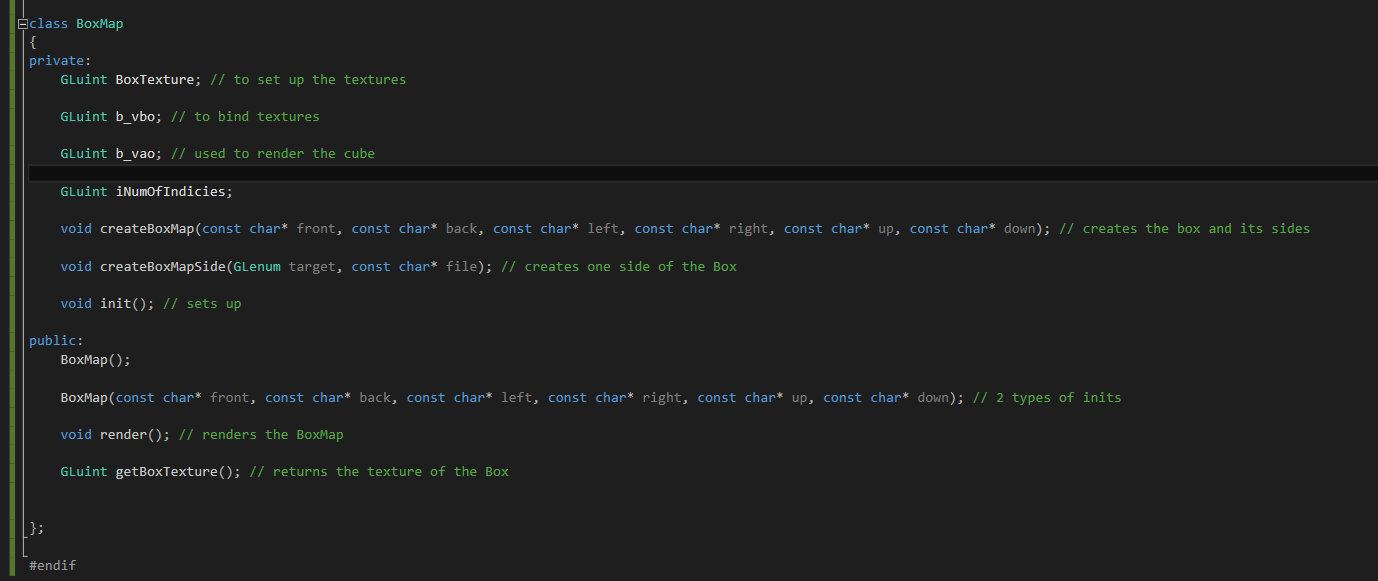
Pictures these days are much easier to obtain, you could in theory stand in one place and turn 90-degrees to take said pictures, however this will prove quite difficult as you will never precisely get the exact angle you would want/need. Furthermore, you can take a panorama photo, where modern phones can take a landscape type shop of the surrounding environment, once again however will prove it not to be to perfect, due to human error of turning/angling the camera.

Now we’ve talked about how a cube map could be created, lets show how to implement one in OpenGL.

**Implementation**

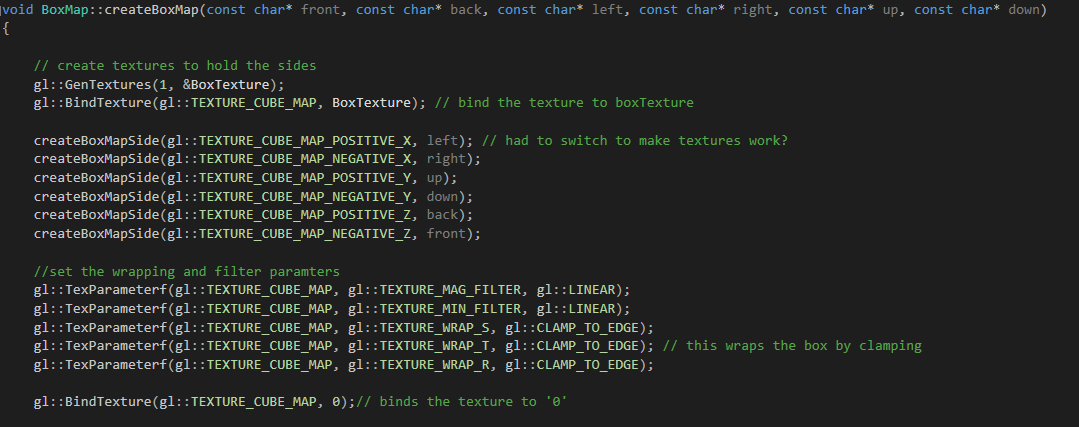
To create a SkyBox, we need to start from scratch and create a class for it.

Here is an example for one:



Here we are setting the basis for what we would need for the CubeMap. We need a vbo, and vao to control the data sent to the fragment and shader programs, as well as a function to take the textures in, to be implemented by openGL,

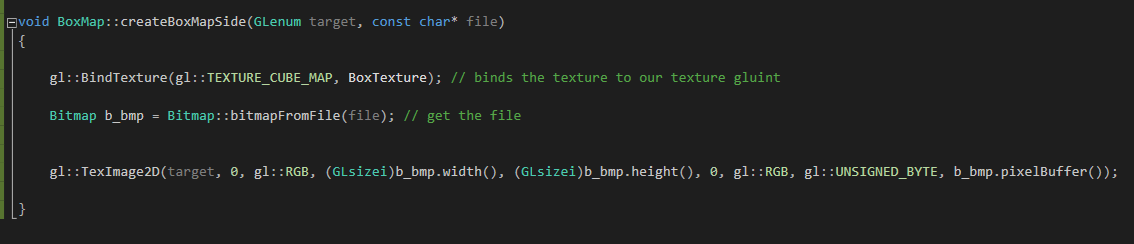
Here is further detail into the CreateBoxMap() function:



Here is the basis on how the texture work for a cube map in OpenGL.

Firstly, we bind the texture onto the one we have pre-instantiated, allowing the following code to be done on the BoxTexture shader variable.

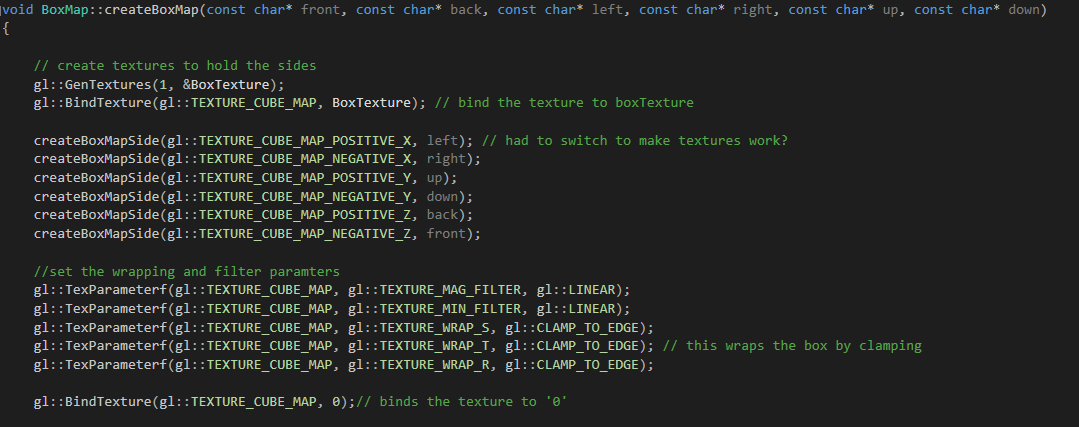
We create the box and its side, with the createBoxMapSide() function, shown below



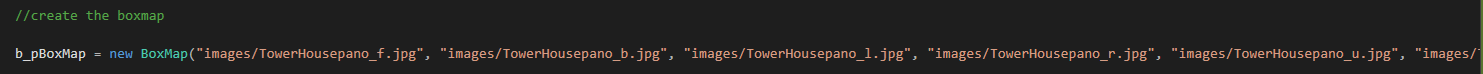
Here we take the target, (the position this particular texture will be facing on the cube), as well as the file reference, which will be implemented when calling the class.

We then bind the texture BoxTexture once again, to make sure we’re working on that, then creating a bitmap object from the class we have, pass the files location to allow it to attack itself to the program. We then similar to 2D images, must call TexImage2D, with our parameters, creating the texture for one side of the cube face. This will be repeated six times.

After calling the function for each side of our CubeMap, we then need to set the filter parameters. Here we had the Mag, Min and TEXTURE\_WRAP, where due to hardware constraints we won’t have edges completely next to each other, so to avoid this problem, we will wrap the texture around the edges, leaving no gaps. The TEXTURE\_WRAP\_R will be wrapping the image on the z-axis.

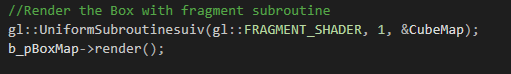


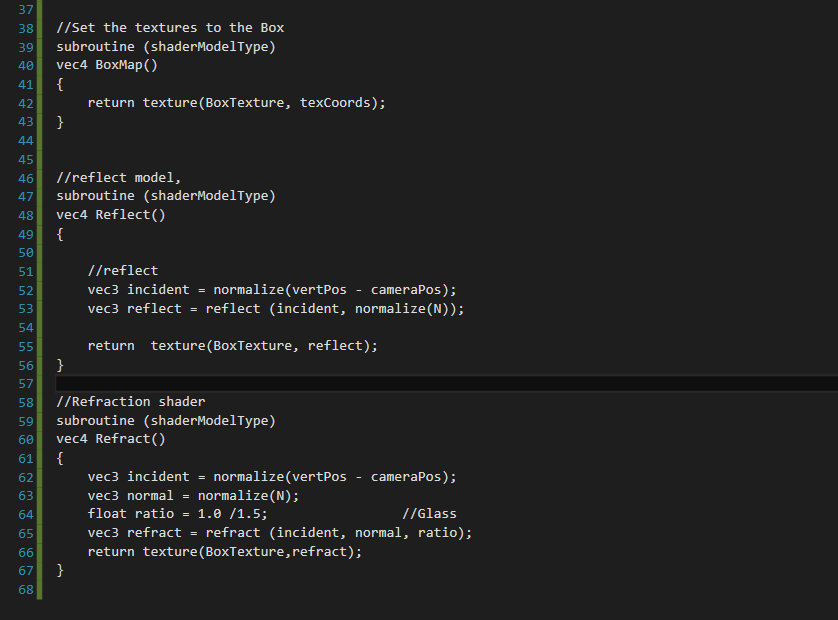
Here is an example of how to initiate the CubeMap Object:



Where we are passing the location of the images that we’re using, as well as the file type.

To render the CubeMap we are using the fragment shader, and vertex shadering, using subroutines as a method:

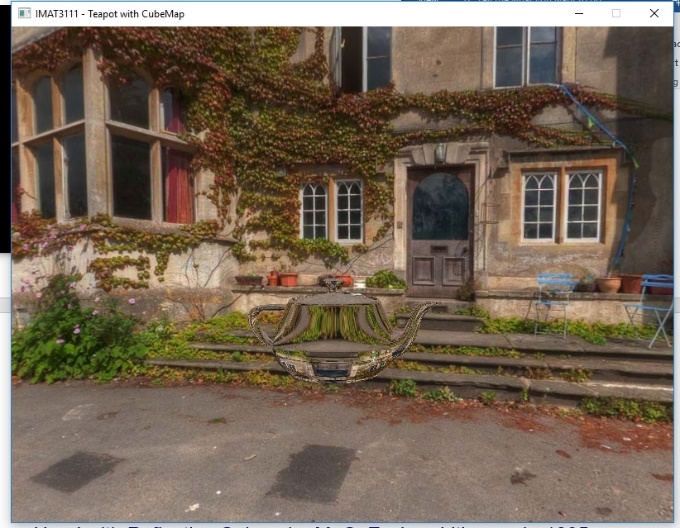




Here is an example of subroutines, and how they are used, here I have them being used to change between reflecting, and refracting my teapot within the environment, allowing me to change seamlessly between the two, without having to change shader, furthermore it allows me to create the CubeMap Textures, as we can see from the subroutine BoxMap(); Which maps the texture co-ordinates with the textures.

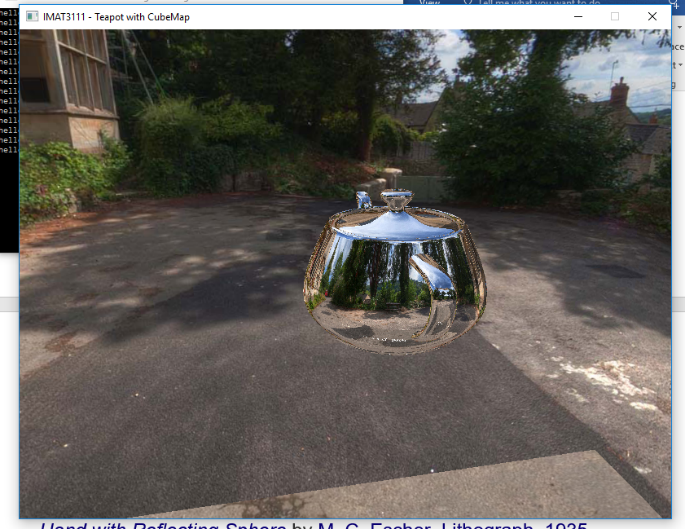
All of the code comes together to produce the CubeMap effect within OpenGL, and here is a example of the project together:

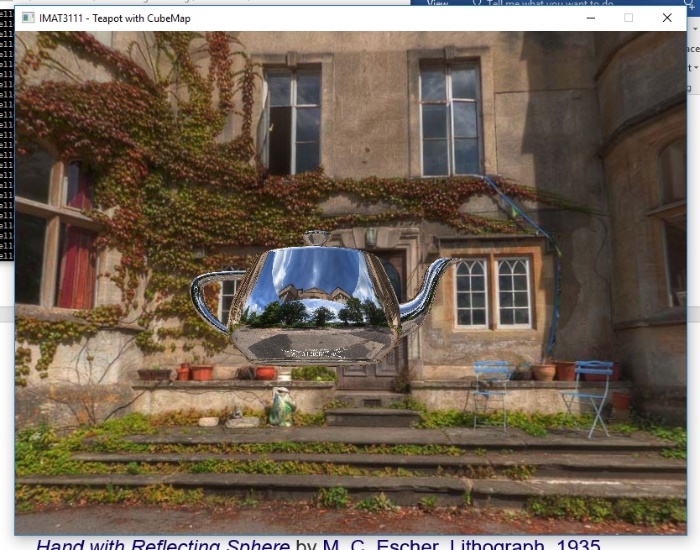
**Refraction:**





**Reflection:**





**Conclusion**

Holistically, cube mapping is a great feature, especially Sky boxing, which has been used in many different games to date, and allows for an easy, yet impressive outcome within the program. Although that texture mapping can be tedious and the fragment shader can be quite picky at times also, it is surely worth the time spent trying to perfect it. I have implemented what I would expect out of the reflections, refractions and the skybox itself.

**References**

**[1]**

**[2]**

**[3]**

**[4]**

**[5]**

[*Hand with Reflecting Sphere*](https://en.wikipedia.org/wiki/Hand_with_Reflecting_Sphere) by [M. C. Escher](https://en.wikipedia.org/wiki/M._C._Escher). [Lithograph](https://en.wikipedia.org/wiki/Lithography), [1935](https://en.wikipedia.org/wiki/1935).

**http://www.mcescher.com/**