**Maxwell Mowbray**
**27005334**

# Software Engineering Project Report

**SOEN 491 Winter 2017**

## OVERVIEW

In computer graphics, ray tracing is a well-known technique for generating high quality renders of a 3D scene. It involves a lighting approximation that more closely resembles real-world phenomena. For each pixel on the screen, a primary ray is shot from the camera position and intersected many times over with the other objects in the scene. The advantages of this illumination technique are the ability to generate renders that are much more photorealistic. However, the increased computational complexity means that this technique is generally not capable in real time.

Figure 1. Ray-traced scene. [1]

## GOAL

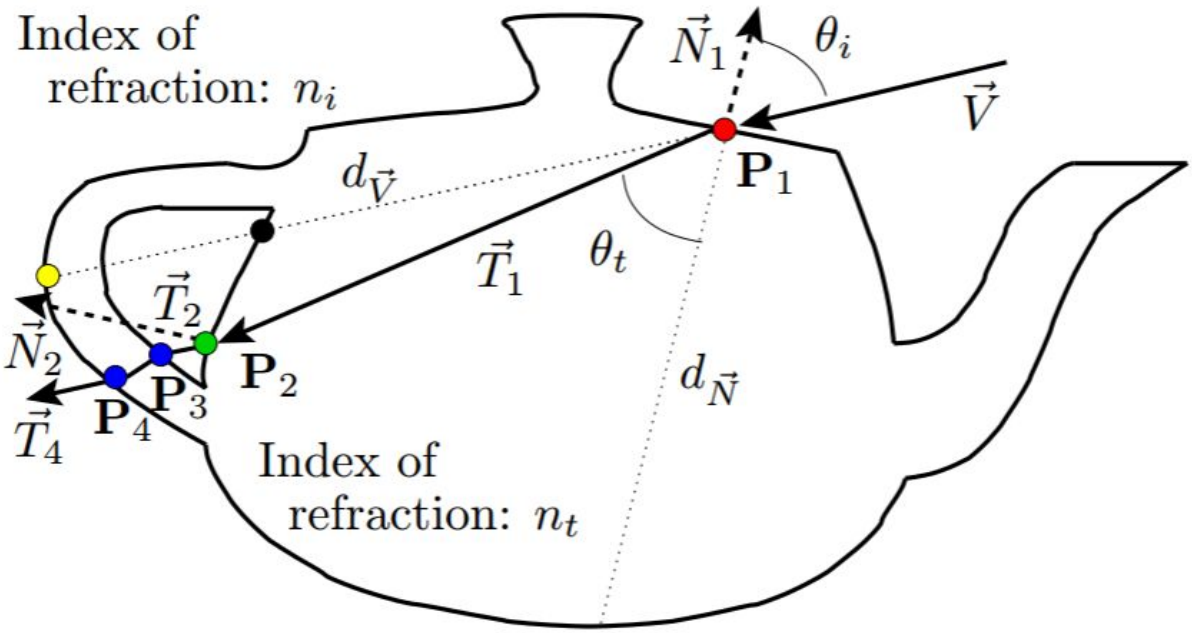
The goal of my project is to attempt to apply Wyman's algorithm to draw a diamond and cube in an approximately-raytraced fashion.

In 2005, Wyman published a paper detailing an algorithm to approximate the ray tracing algorithm in real time.

His technique can be described simply by this diagram, which comes from the paper itself.

Traditional ray tracing follows the solid black vector lines. Starting from **V**, the vector from to camera to the point of intersection with the object, the ray undergoes its first refraction and bends inward at point **P1**. After applying Snells's law, the first refracted ray **T2** travels on and intersects with a backface at **P2**. Depending on the number of back faces, the ray continues to intersect and refract, before eventually exiting the object in direction **T4**.

Wyman's algorithm deviates from the traditional one by refracting only twice. The location of **P2** in this case is approximated with **Dn** and **Dv.**



The interesting aspect of the algorithm is the idea of rendering the object in two passes.

In the first pass, the back face is rendered. This occurs in a separate, off-screen framebuffer. The depth information is saved in a depth buffer, and the normal information is saved in a colour buffer.

In the second pass, the front face in rendered. Wyman's approximation can be conducted because for each fragment, the appropriate backface normal and depth can be sampled from the textures generated in the previous pass. The final fragment colour is determined.

## IMPLEMENTATION

The following software resources were used in realizing the project:

- OpenGL 3.3
- Modern C++
- Visual Studio
- LucidChart

### 3D Scene

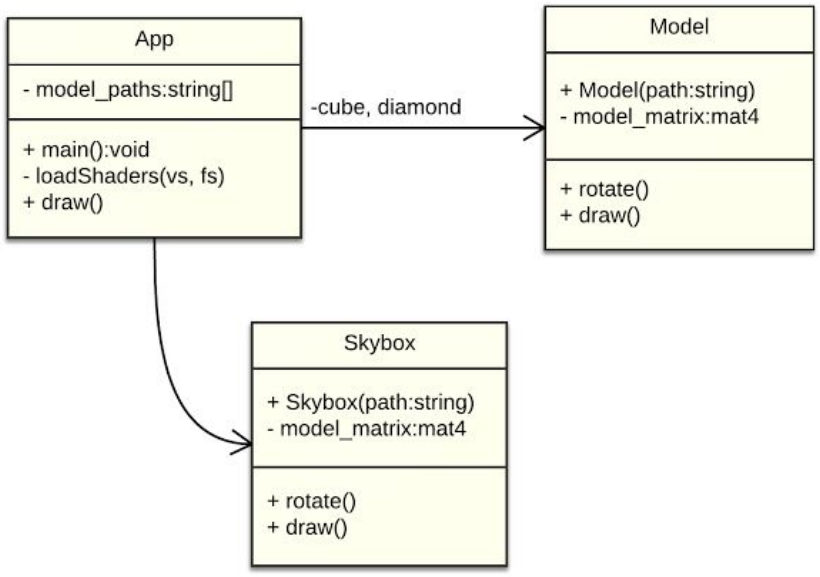
A 3D model sits in the centre of 3D world. A skybox is also included. The camera can zoom in and out, and the skybox can be rotated to better investigate the lighting effects. The model can also be freely rotated. Diffuse and specular lighting can be toggled.

### Controls

- Click and drag to rotate the model
- C key changes to the cube model
- D key changes to the diamond model
- Scroll wheel to zoom in and out
- Left/Right to rotate the skybox
- 1 for raytracing approximation
- 2 for front normals
- 3 for back normals
- 4 for front depth
- 5 for back depth
- Space toggles the lighting

## SOFTWARE ARCHITECTURE

The following class diagram describes the contents and architecture of my software solution.




***App*** is the application entry point which conducts the program. ***Model*** represents an actual Model that can be approximately raytraced. Finally, ***Skybox*** is similar to the ***Model*** class, but is specifically designed to contain a skybox cubemap.

## OPENGL DIAGRAM

Based off of a UML domain model diagram, this figure demonstrates how the algorithm is implemented within the context of the OpenGL graphics pipeline.

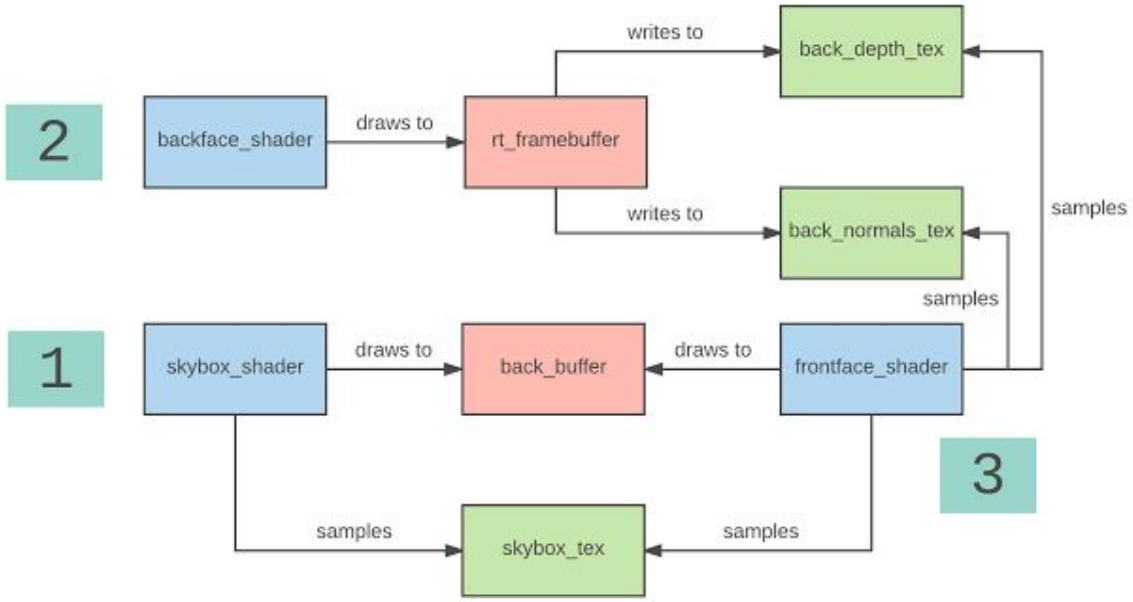
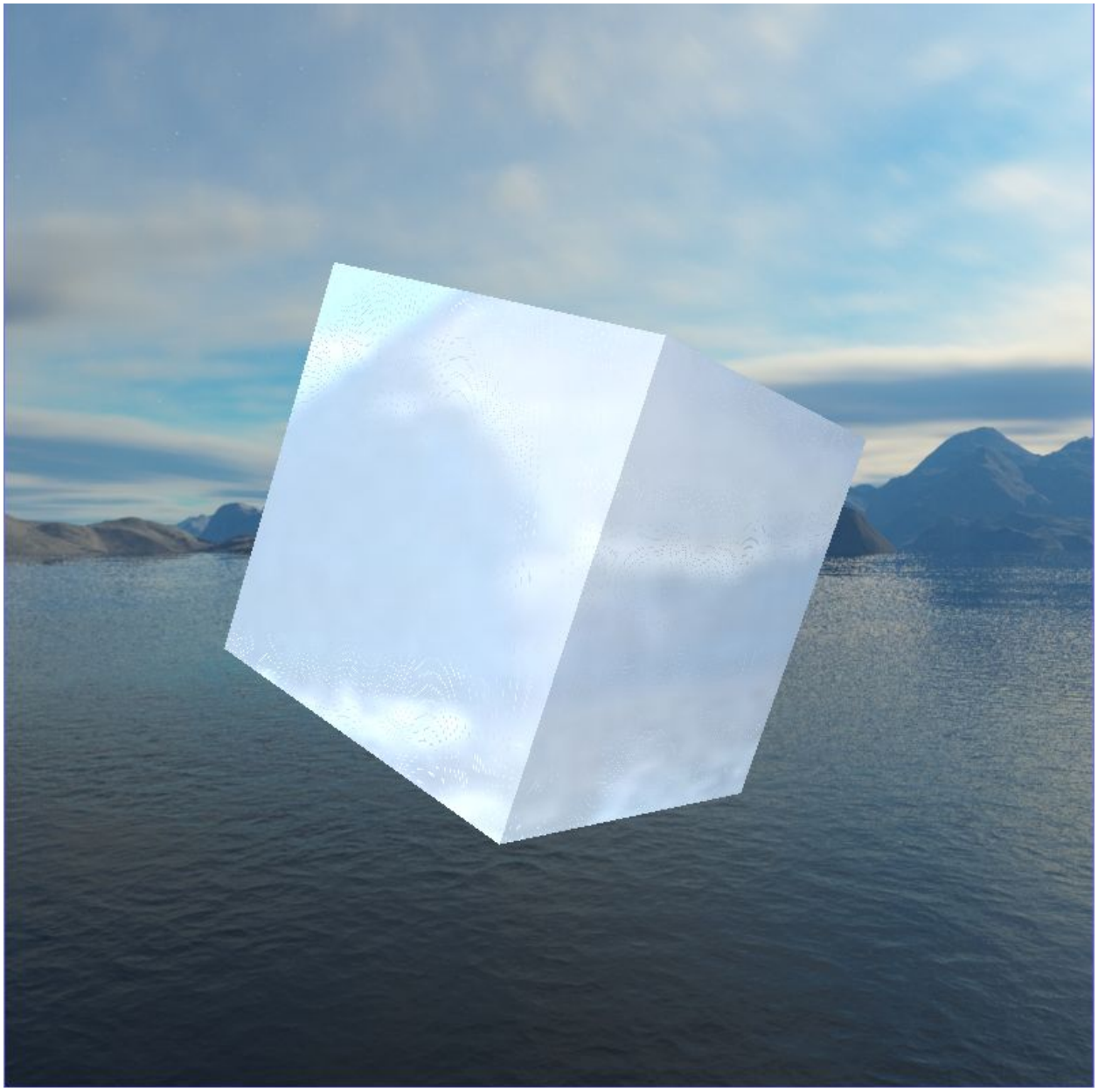



Figure 2. Pipeline implementation. Custom Content.

## RESULTS

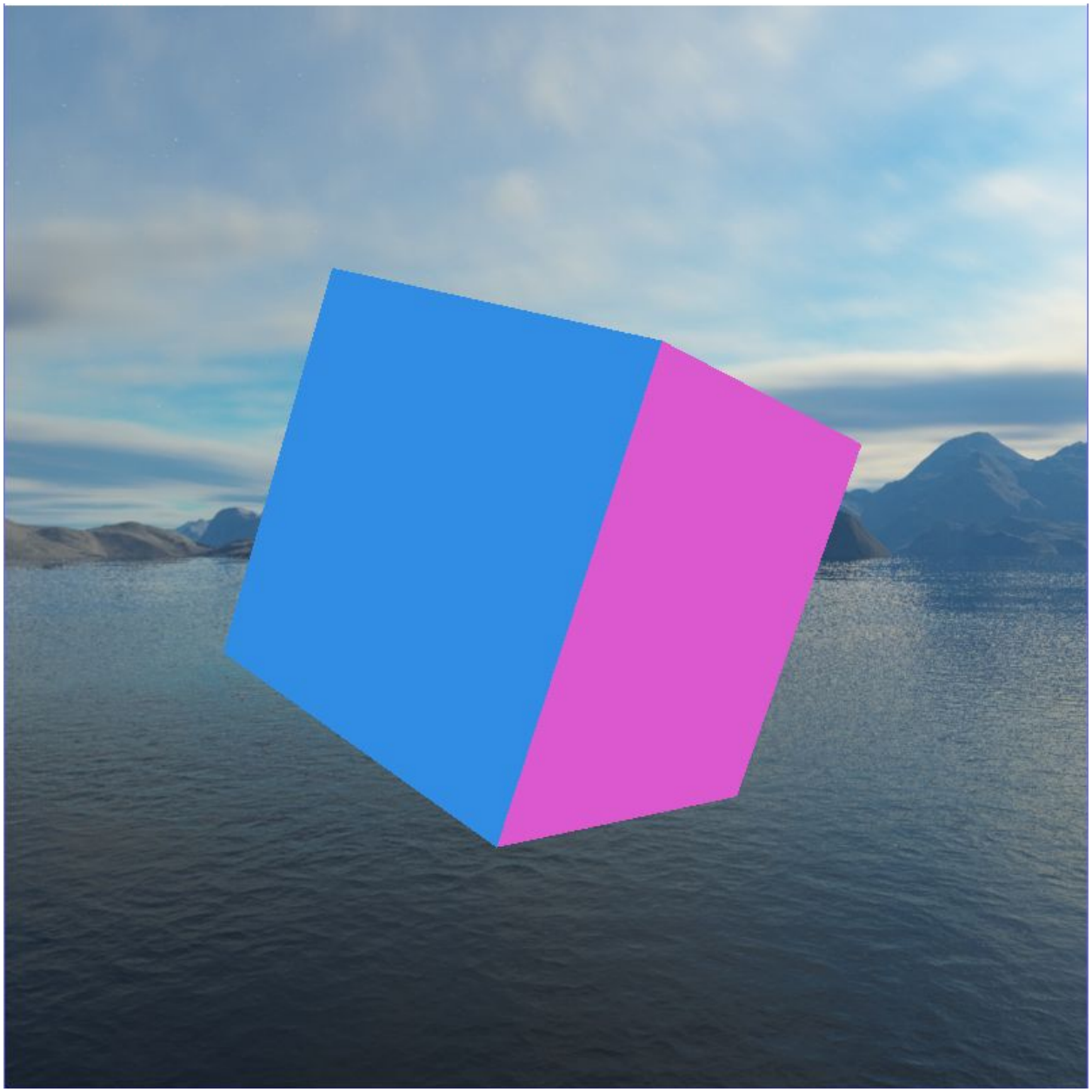
### Cube

Raytracing approximation and lighting

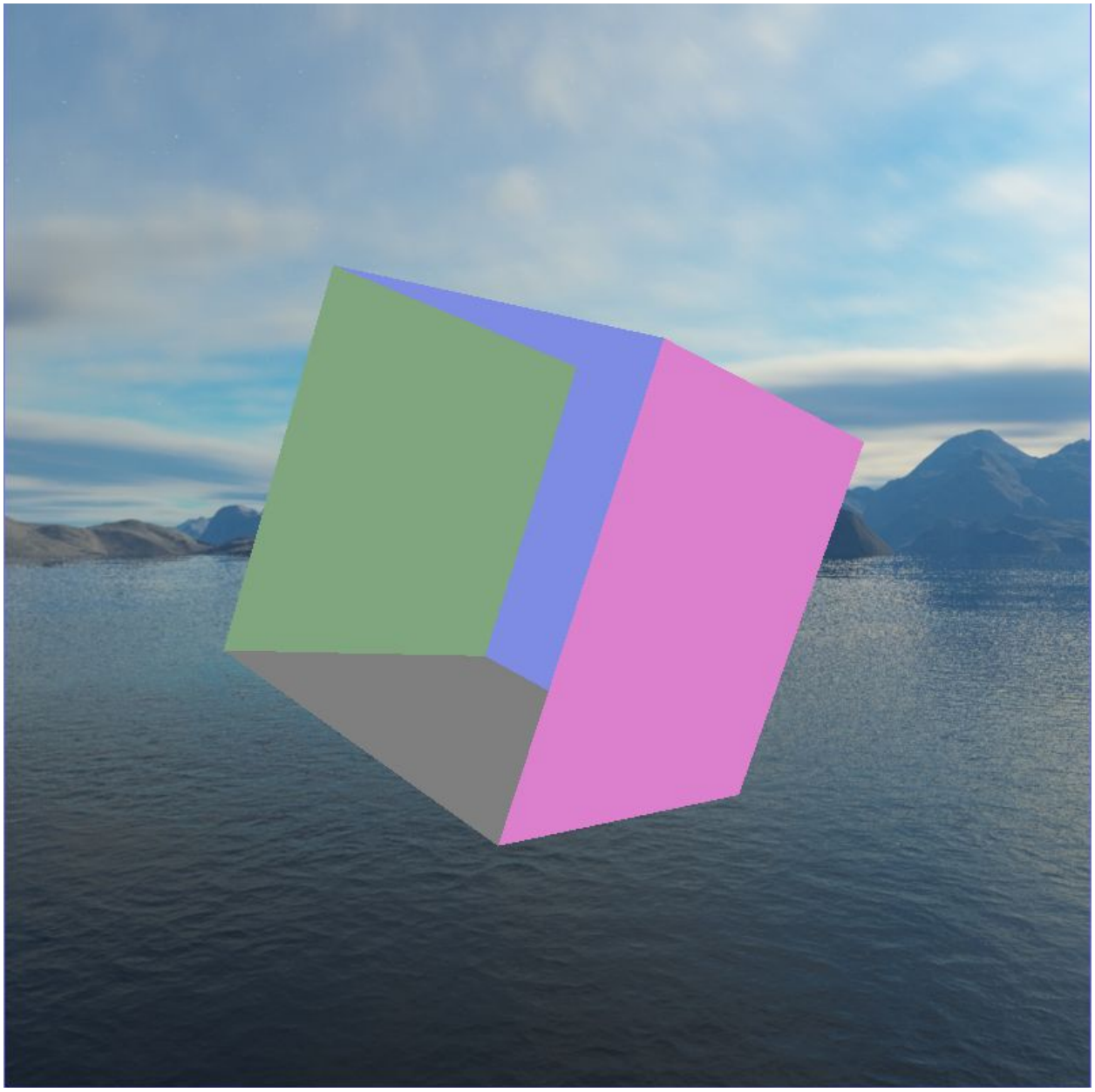


Raytracing approximation without lighting

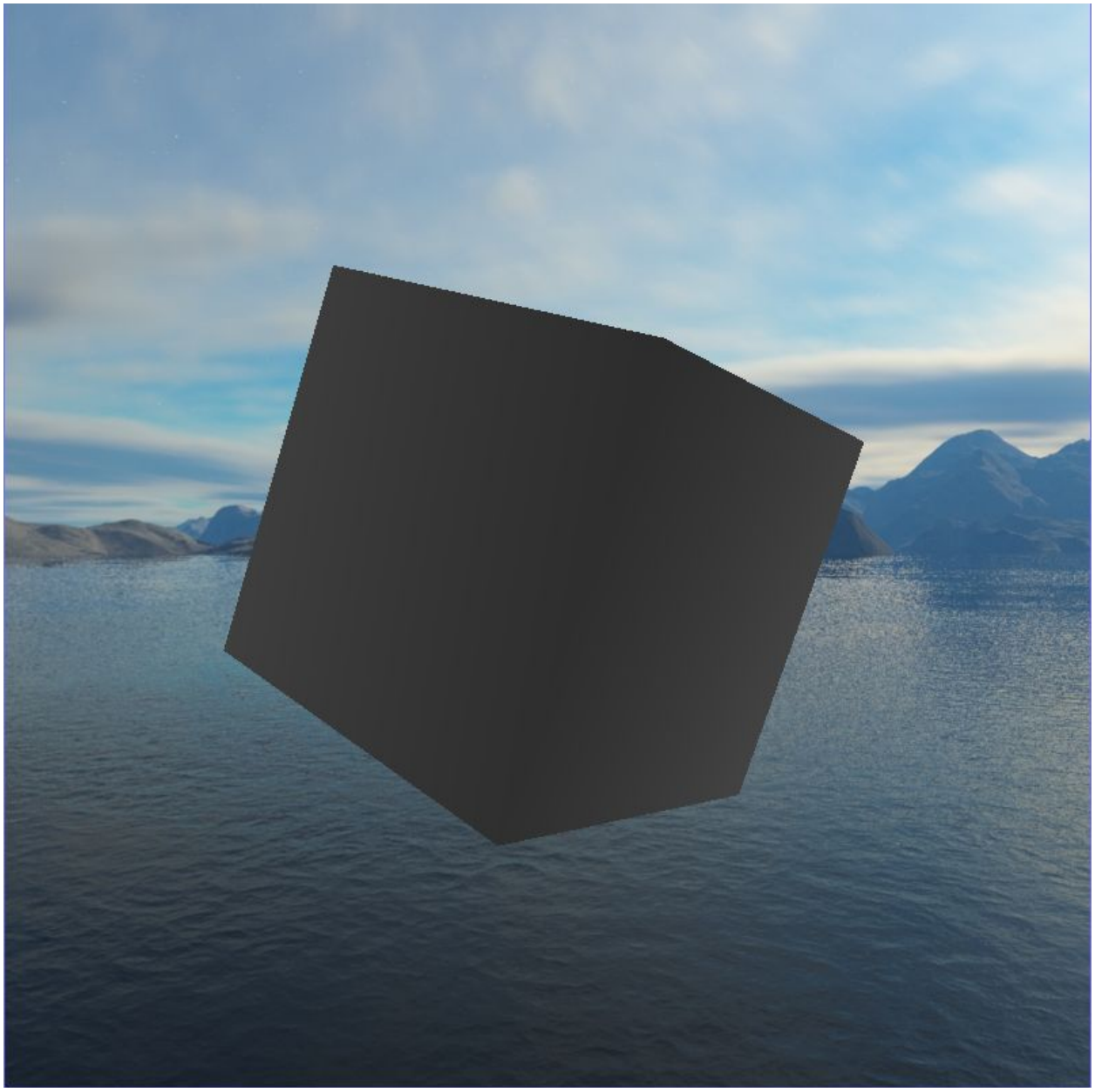
Front Normals



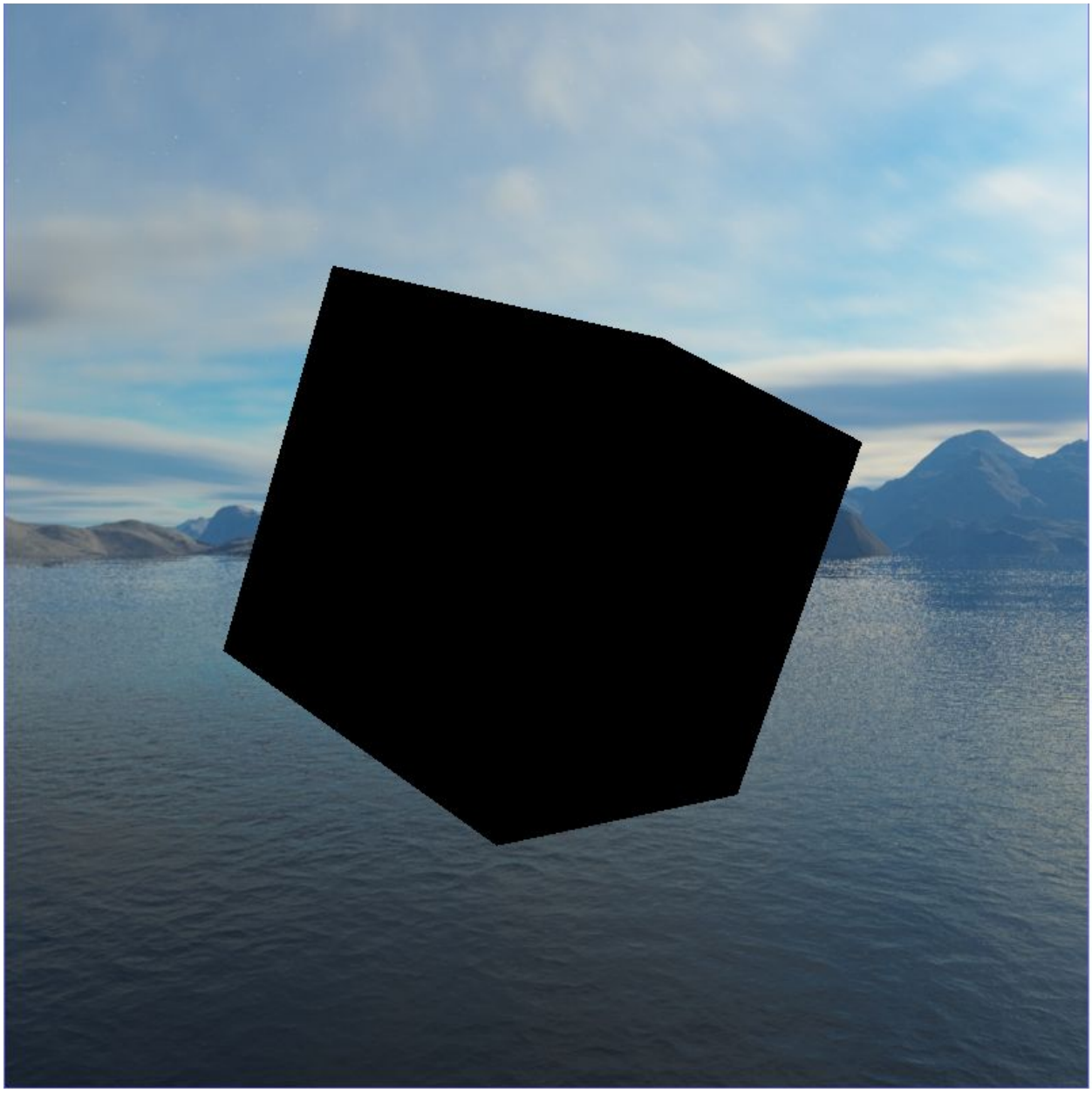
Back Normals



Front Depth

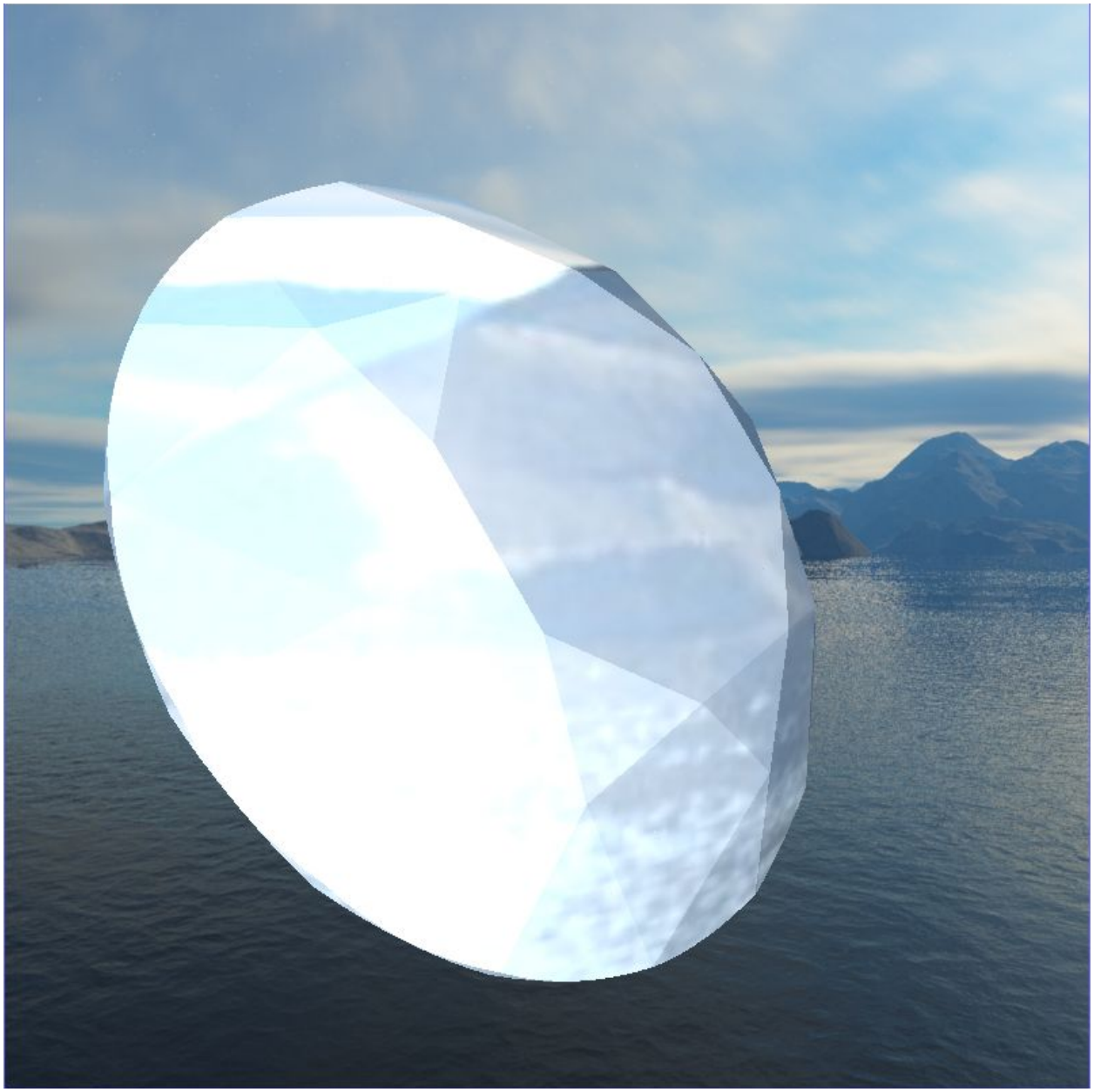


Back Depth



## Diamond

Raytracing approximation and lighting

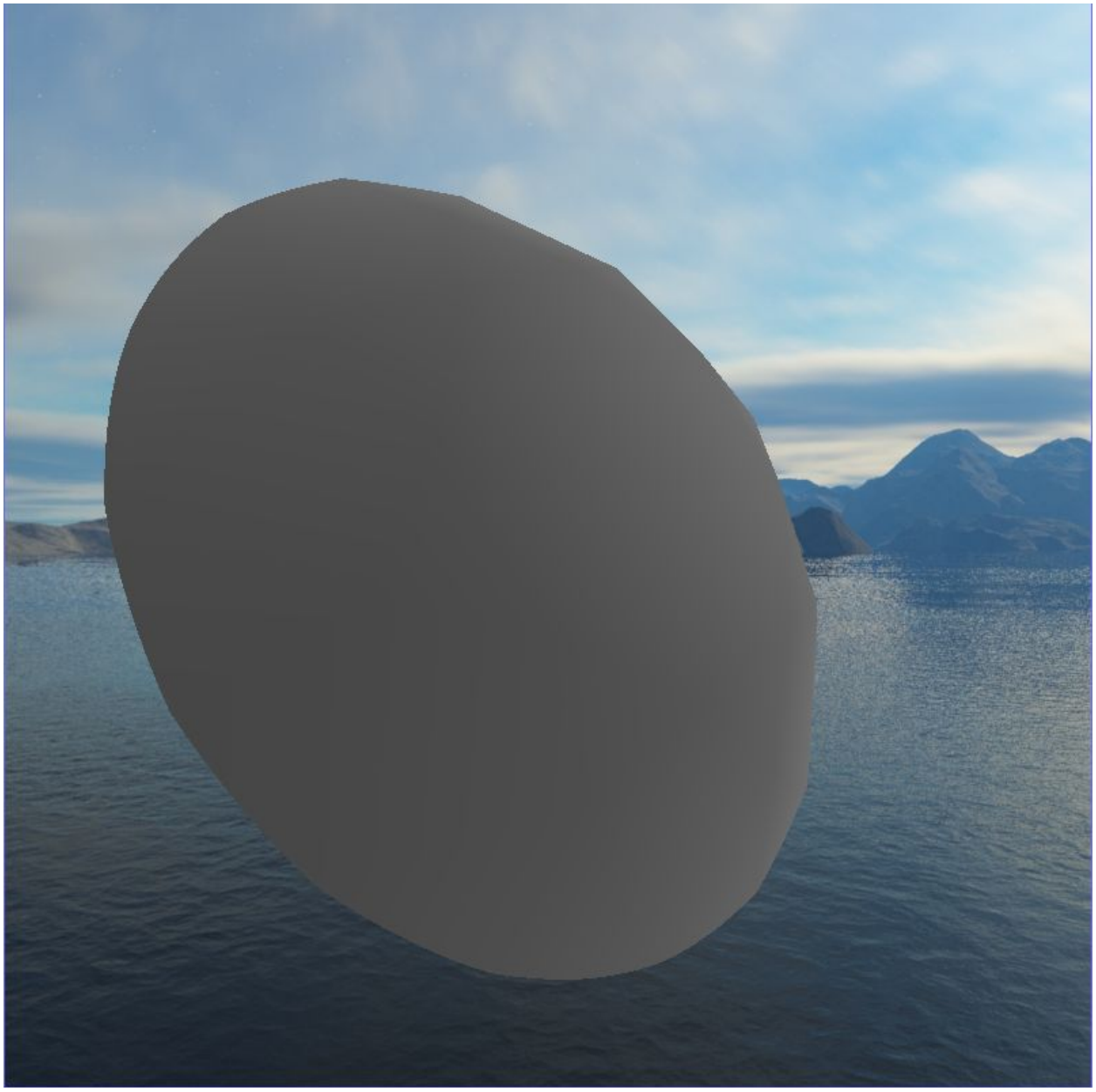


Raytracing approximation without lighting

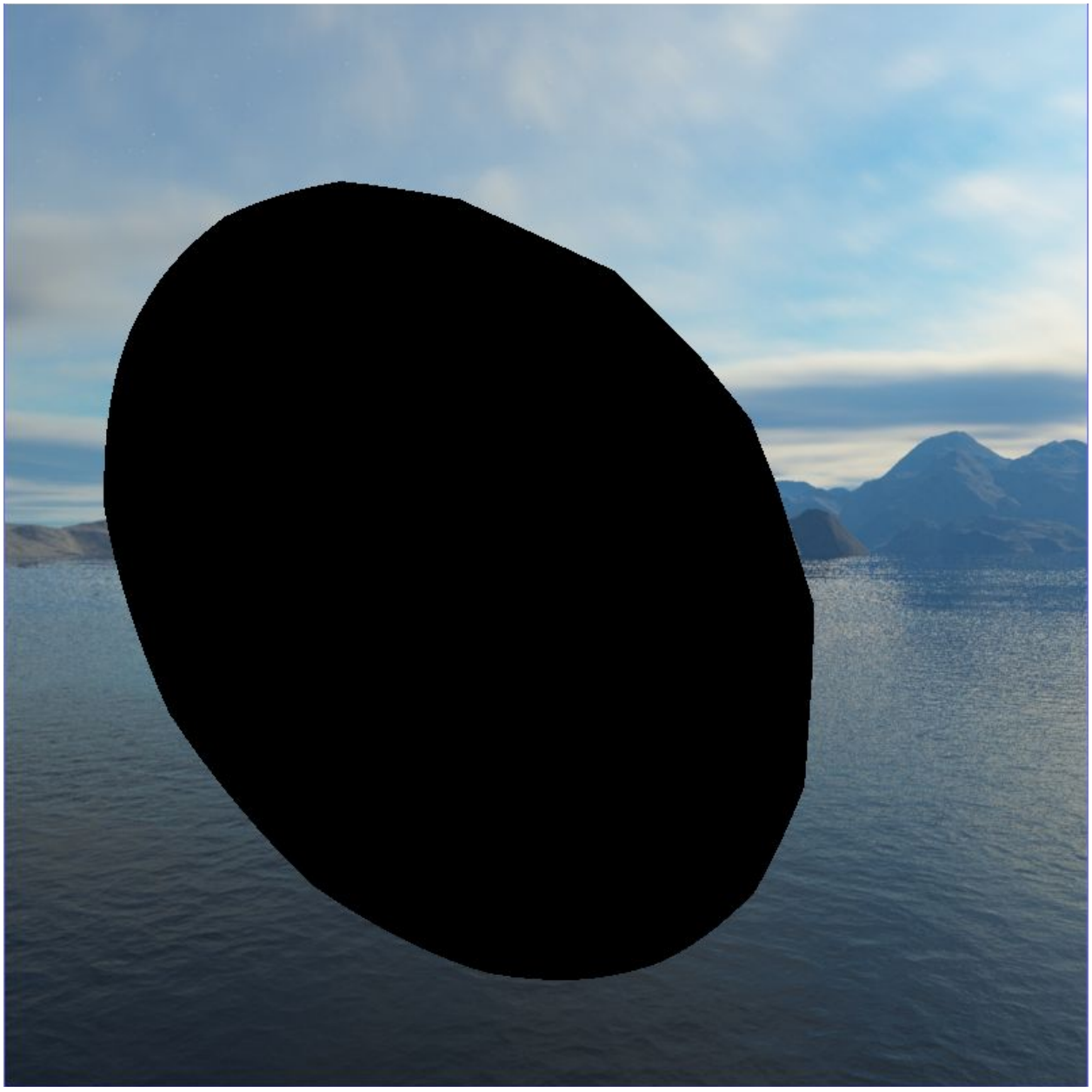
Front Normals

Back Normals

Front Depth



Back Depth



## DISCUSSION

Overall, my implementation is very close to correct, with one or two implementation specific error remaining.

The lighting distortion appears to be behaving correctly for the diamond, but for the cube, the backface distortion should be more visible and distinct from the other faces.

The front normals are correct, but the depth testing for the back normals appears to be slightly erroneous. The front depth display looks good.

Some issue remains with the framebuffer depth test, and after much research and effort I was unable to determine why the backface depth does not look like the front face depth. I suspect an error in the way the backface depth is being saved (format issue) or the way I am sampling it.

## CONCLUSION

Despite small issue in my implementation, I am very happy with the raytracing approximation I have implemented. I can sort these small issues out at a later date.

This project has increased my experience with working with the OpenGL graphics library, and more specifically, framebuffer objects. Rendering off-screen to a texture has many other uses, including stenciling, convolution, and lighting. I am sure to use this concept again in the near future.

This is also the first time I attempted to implement a research paper on my own, without any direct help or instruction in the context of a class.

Overall, it was challenging but fun experience and I am glad I had the chance to partake in it.

## References

1. http://www.oyonale.com/modeles.php?lang=en&page=40