CS116A Final Project: Endless Sky

Course Instructor: Kevin Smith

Nhu Nguyen

Computer Science Department

Introduction

Mirroring effect, in certain circumstances, can create beautiful natural phenomenon.

This effect is caused by a still, large body of sea water reflecting the sky, creating an “endless sky”.

My project aims to recreate this scene using merely computer graphics to show case the computer’s ability to simulate real life visual effects. More importantly, to pay tribute to the aesthetic of Earth’s environment. Lastly, this project demonstrate how different renderer techniques work in a 3D virtual environment.

Methodology

General

As a class, we use C++ and OpenGL, a library that supports 2D/3D computer graphics. In addition, we use openFrameworks, a more advanced library that provides technical environmental support to build a renderer.

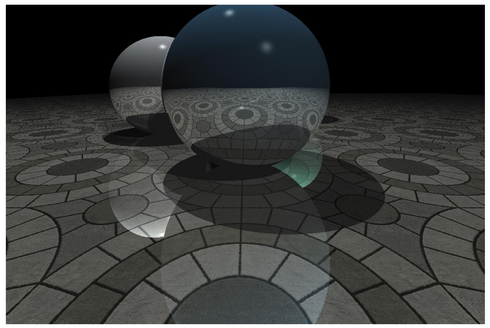
Mesh Renderer

A mesh is a structure containing vertices connected by edges. The vertices connected create faces. Different polygons could be used to construct a mesh, but in our class, we use triangles. Each triangle has three vertices and one face. The mesh .obj file could be drag and drop by the user, this means that my program could build any mesh, given the .obj file with appropriate formatting. Below is a prototype of a rendered mesh.

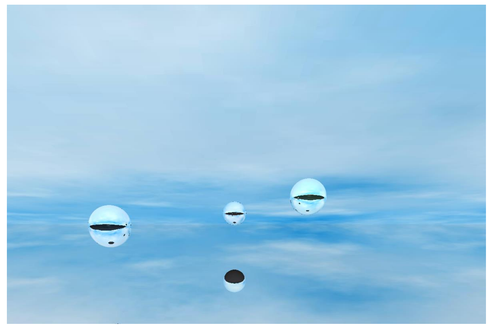
Methodology

Mirroring

To achieve the reflective surface effect, a rendering technique called recursive rendering is used. The ray tracing function is continuously being called recursively until the end of the recursive depth. This leads to colors of a point on one object is being passed on to another object. Below is a prototype of reflective objects.



However, we don’t want all objects in the scene to be reflective. In my case, I only want the “water surface” and the “glass spheres” to be reflective, not the mesh.

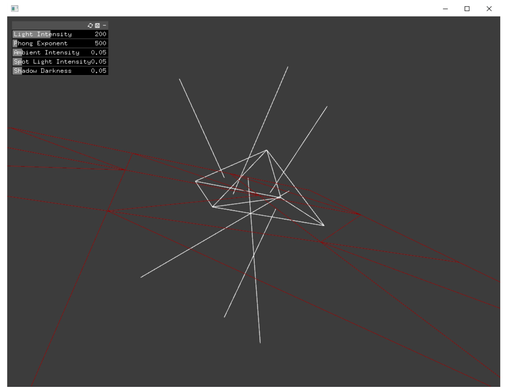


Analysis and Results

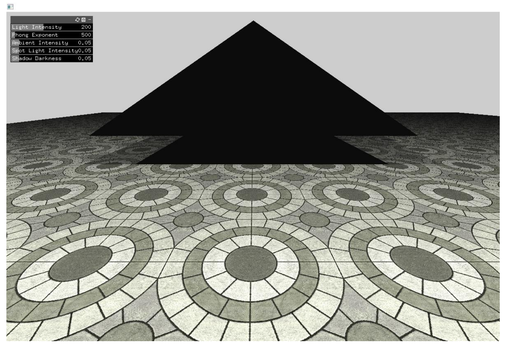
Challenges

A variety of challenges were encountered. First, to support rendering a mesh that contain thousand of faces, each of them are ray traced with 1200x800 rays (image resolution), using a single thread is too inefficient. I overcome this challenge by multithreading the ray tracing process and cleaned up the code to eliminate unnecessary nest for loops. Moreover, the multithreaded ray tracing provides a better user response interface where I could see the process of rendering in real time, as opposed to only seeing the end result after sitting and waiting without knowing if things work.

With mesh rendering, I initially faced the challenge of determining a triangle’s normal. To compute the correct normal vector, the vertices need to be in a correct and consistent order. Below is how the normal vectors were before I fixed it.



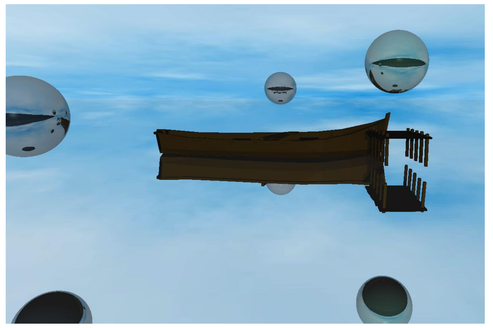
The OpenGL library provides convenient function to check for intersection between a ray and geometries, yet the glm::intersectRayTriangle() would return true even if the ray is 0.001 (epsilon) away from the triangle surface. This caused the surface to be recognized as an obstacle between a point and the light, turning it to a shadow point, resulting in a darken out mesh.



To achieve the horizon, I decided to use one orthogonal plane and a tilted plane. Making them intersect at the -z direction is easy enough yet texturing them is challenging. On the other hand, I have the mesh object available. Hence, a 3D primitive object quickly turned to a customized mesh of two triangles.

Result

The final result is adequate to fulfill the project requirement. However, as future work-to-be-done, a Gaussian blur effect is worth looking into if I want to apply a blur on the water surface to more accurately recreate the scene.



Summary/Conclusions

Though not all aspects of the natural scene were represented, the result shows the basic idea and overall look of the scene. Future works are needed but the result demonstrates all proposed areas.

Acknowledgements

The project could not be accomplished without the support and guidance of my instructor and project advisor, Professor Kevin Smith.