CSE306: First Assignment Report

Duc Hieu Le

1. Introduction

In this project, I implemented a Ray-Tracer in C++. It has the following features:

- Diffuse, mirror, refraction (solid and hollow) surfaces with Fresnel's law
- Direct lighting and shadows, for point light sources
- Indirect lighting for point light sources
- Spherical light source
- Antialiasing
- Ray mesh intersection

All renders are of size 512 x 512. Maximum ray depth was set to 5.

2. Code structure

To test the code, simple just run the script **run.sh**

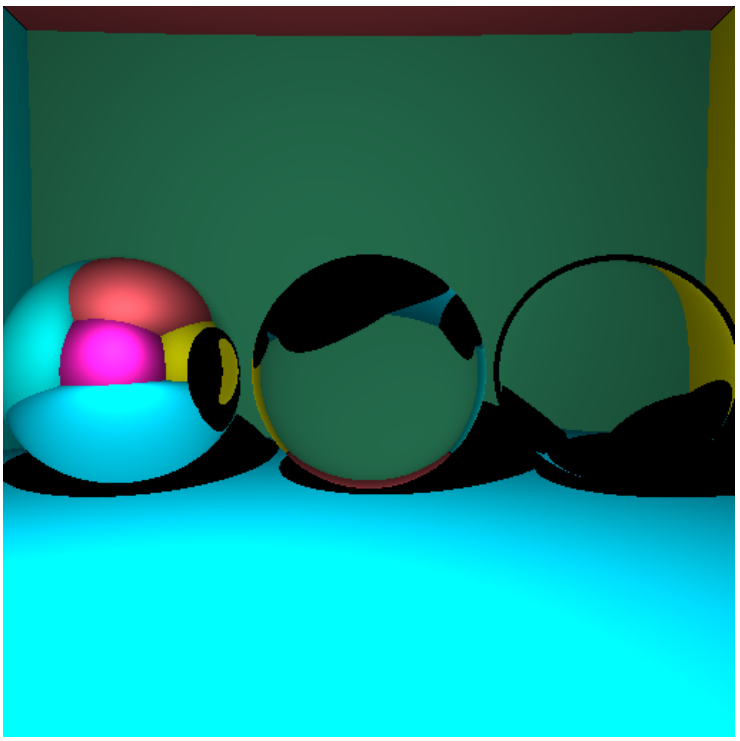
- **vector.cpp**: contains the class **Vector** with useful vector-related functions
- **objects.cpp**: contains the class **Intersection, Ray, BoundingBox** and an abstract class **Geometry** (to be inherited by **Sphere** and **TriangleMesh**). The only virtual method of **Geometry** is the **intersect(const &Ray aRay)** method.
- **mesh.cpp**: contains the class **TriangleIndices, Node** (to represent a BVH node) and **TriangleMesh** with its own implementation of the **intersect(const &Ray aRay)** method
- **sphere.cpp**: contains the class **Sphere** with its own implementation of the **intersect(const &Ray aRay)** method
- **utils.cpp**: contains utility functions including **boxMuller** for antialiasing, **randomCos** for Monte-Carlo integration and **square** for making the code more readable.
- **scene.cpp**: contains the class **Scene** with background set up in the constructor with walls, floor and ceiling. It has a **getColor** method and also an intersect method to return intersections of the elements of the scene with a ray.
- **main.cpp** contains a main function to render images.

3. Result images

All rendering were done on my laptop, with specs:

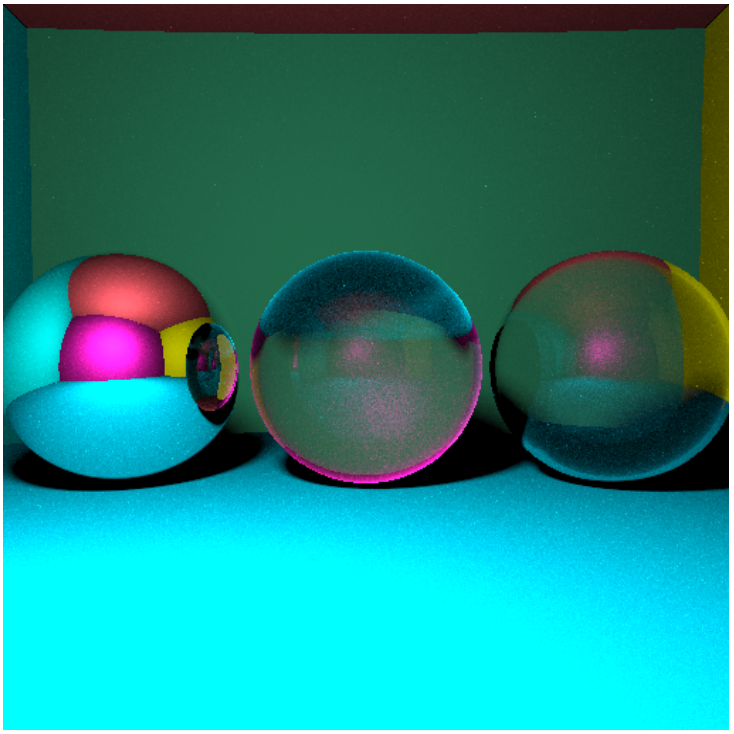
- Processor Intel(R) Core(TM) i7-8665U CPU @ 1.90GHz 2.11 GHz
- 16GB Ram
- Intel(R) UHD Graphics 620

By follow the instructions of the course, this is the image after I successfully implemented surfaces



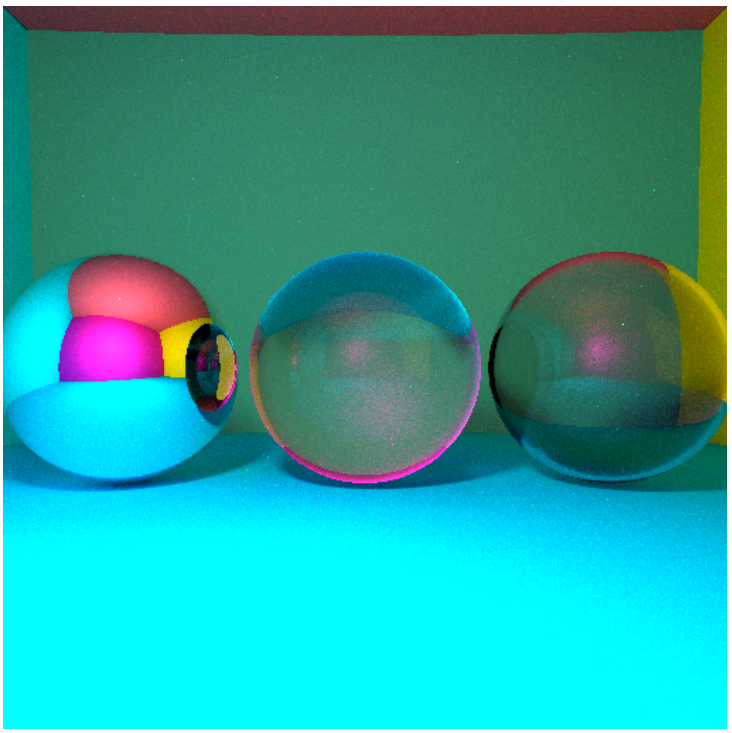
Reflections & Refractions with Fresnel (200 rays per pixel, 39s render time)

After that, I implemented direct lighting and here's the change



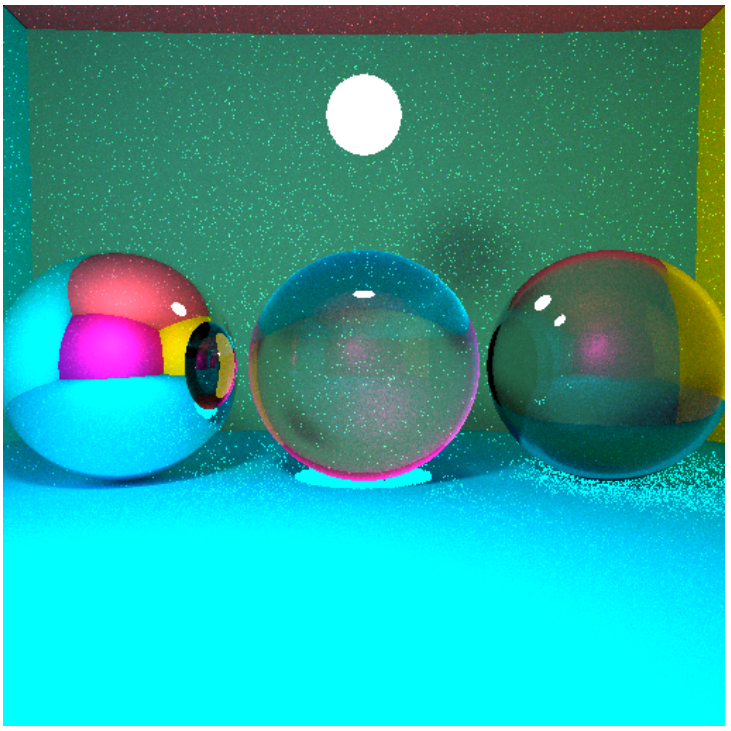
Direct lighting (200 rays per pixel, 38s render time)

Then, I implemented indirect lighting, this increases the render time hugely



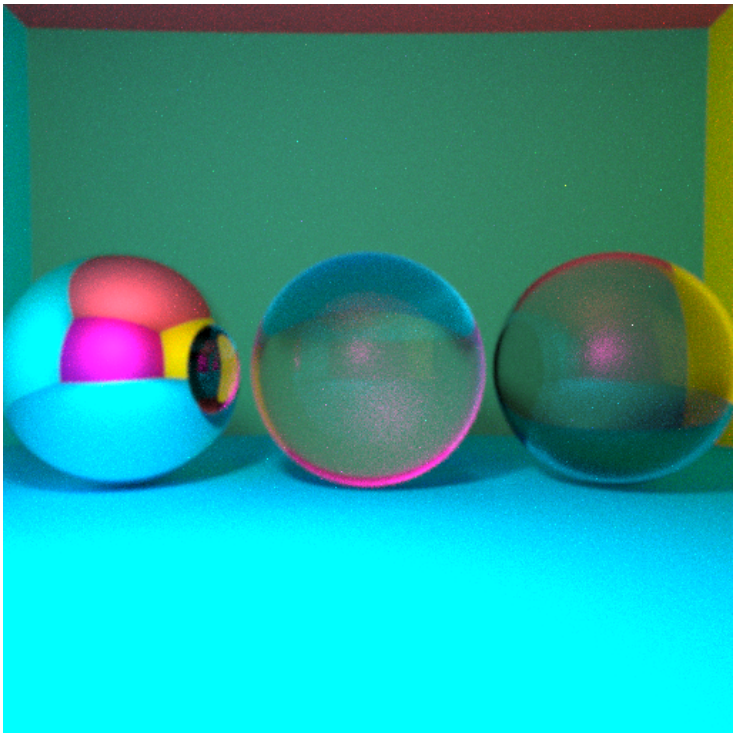
Indirect Lighting (200 rays per pixel, 190s render time)

Then, I implemented a spherical light source. I consider a spherical light source simply an object of sphere class that has **theIsLightSource** flag being true. I just need to change the Intersection class, by adding **theIsLight** flag to know if it is an intersection from a light source or not, then I change the **getColor** method of the **Scene** class. But the resulting image is quite noisy.



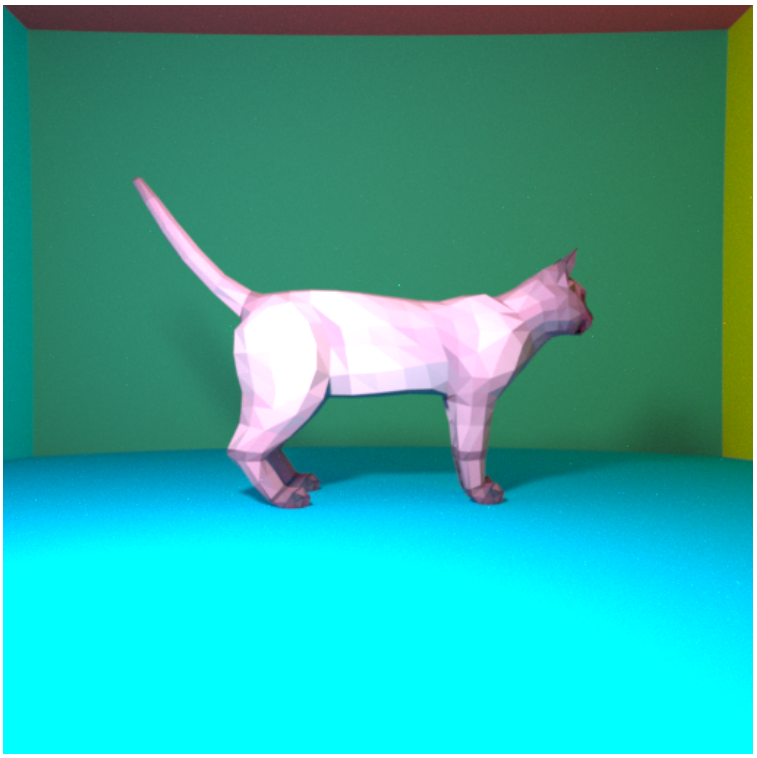
Spherical Light (200 rays per pixel, 193s render time)

After that, I implemented anti aliasing with boxMuller, and here's the resulting image with boxMuller set to 1.



Anti Aliasing (200 rays per pixel, 190s render time)

Finally, I added the cat to the scene and implement ray-mesh intersection with BVH to boost the render time and here is the image



Ray-Mesh Intersection with BVH (1000 rays per pixel, 1700s render time)