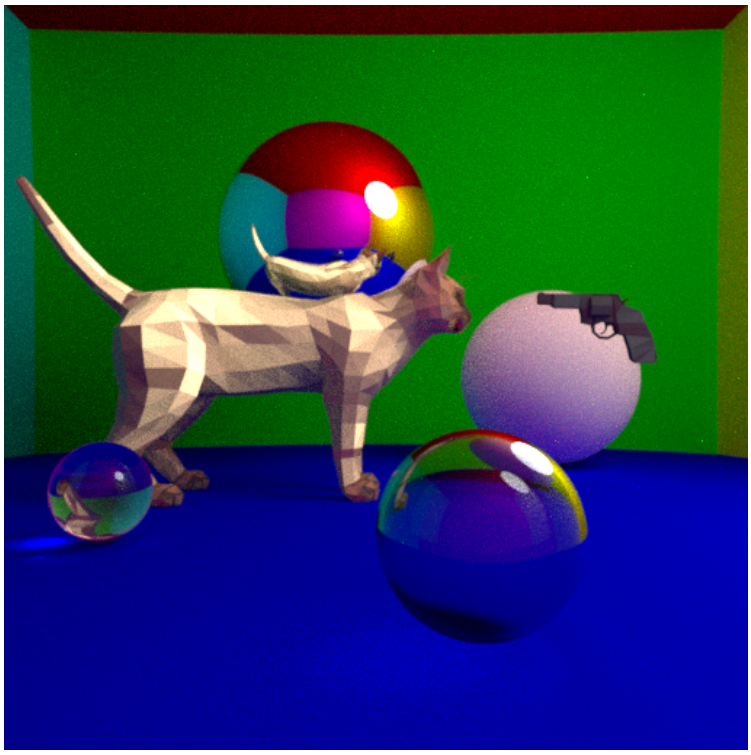
# CSE306: Ray Tracer Project Report

Joshua P. Jacob (BX21)



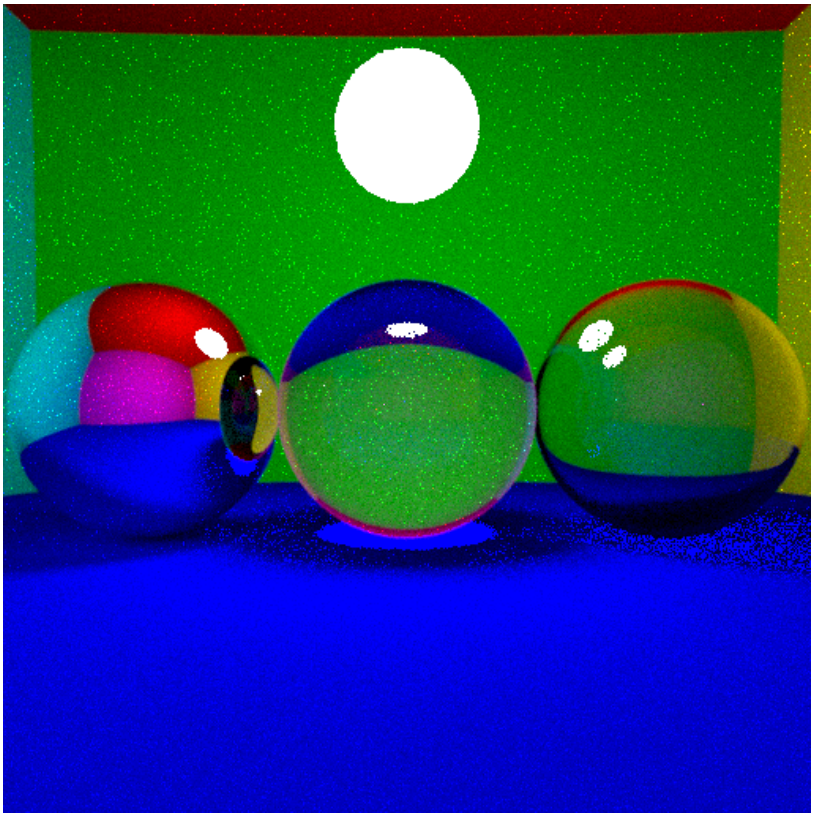
## Introduction

In this project, I implement a simple ray-tracer from scratch in C++. The following were implemented: spheres, diffuse surfaces, mirror surfaces, refraction (solid and hollow) with fresnel's law, indirect lighting through monte-carlo integration, spherical lighting (soft shadows), parallelization, antialiasing, simple triangle meshes (without textures), and bounding volume hierarchies acceleration. **All renders in this report are of size 512 x 512 pixels.** Maximum ray depth was set to 5. The final render above took 32min to complete with 1000 rays per pixel.

## Code Structure

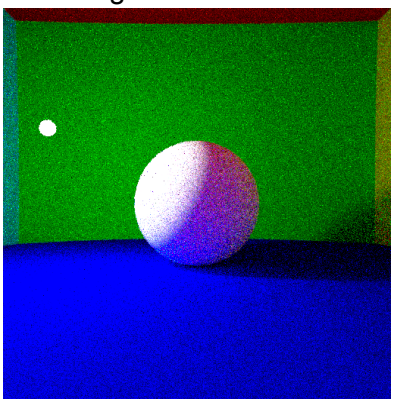
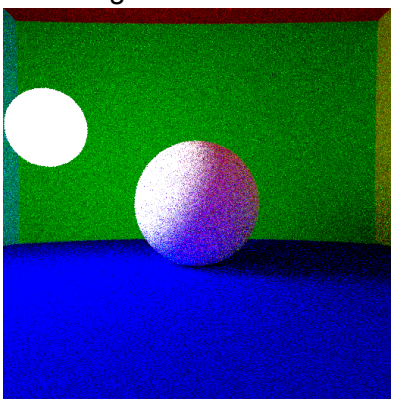
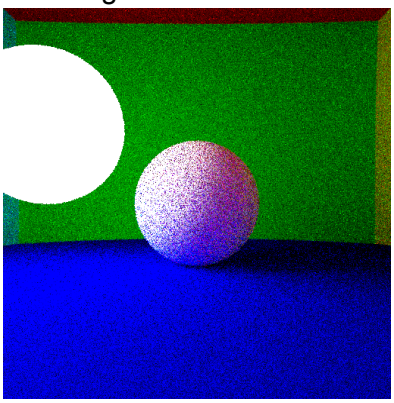
The code is organized very simply: most of it is in one file, main.cpp. The implementation of Vector class is in vector.cpp. In main.cpp, you will find the basic objects: intersection, bounding box, rays, and BVH nodes. There is an abstract parent class geometry with two different derived classes: one for spheres and one for triangle meshes. The primary difference between the two derived classes is the way they handle intersections. Bounding boxes and BVH were only implemented for triangle meshes. Utility functions include boxMuller for antialiasing and random_cos for monte-carlo integration. The BasicScene class sets up a simple scene with walls, floor, and ceiling.

## Reflections & Refractions (with Fresnel)



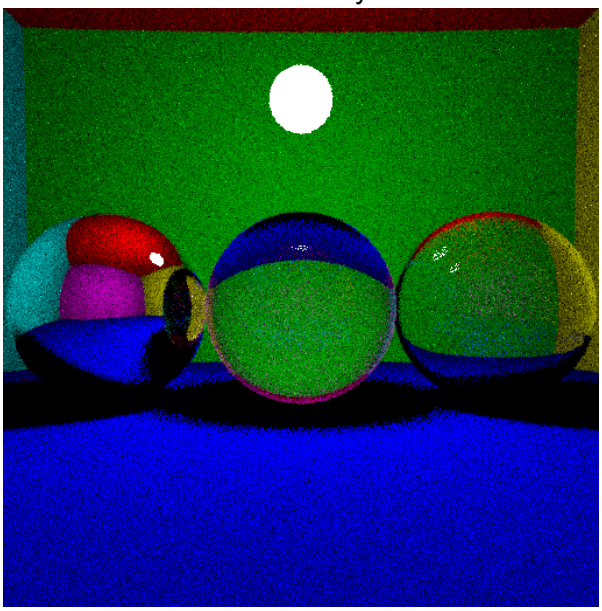
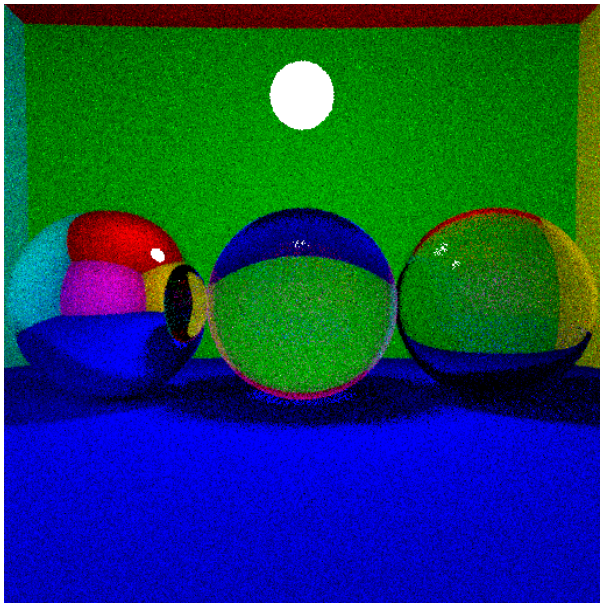
148s (100 rays per pixel)

## Spherical Light Sources

| Light Radius = 1 | Light Radius = 5 | Light Radius = 10 |
|---|---|---|
|  |  |  |
| 16s (10 rays per pixel) | 16s (10 ray per pixel) | 16s (10 rays per pixel) |

## Direct vs. Indirect Lighting

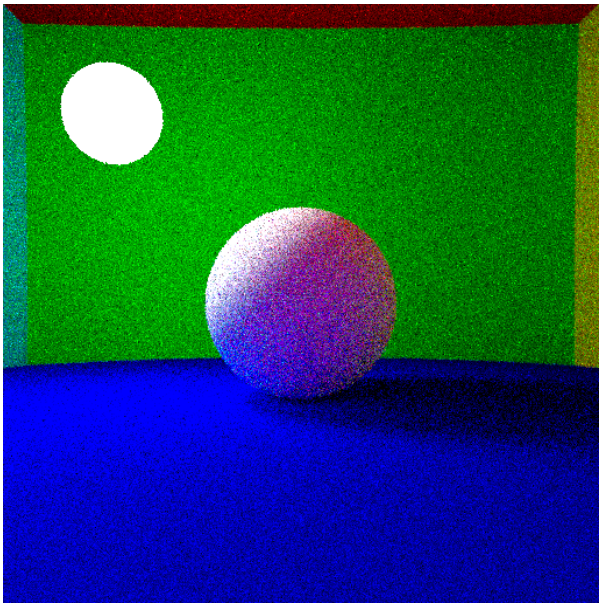
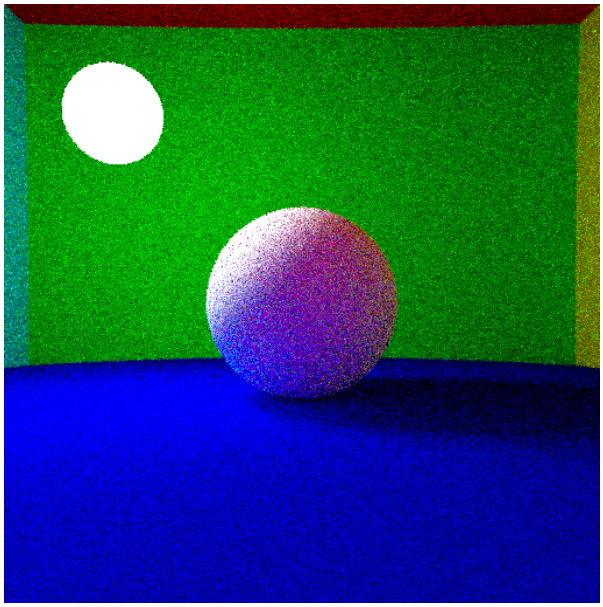
*Lighting is still spherical in both cases.*

| Direct Only | Direct + Indirect |
| --- | --- |
|  |  |
| 2.8s (10 rays per pixel) | 13.5s (10 rays per pixel) |

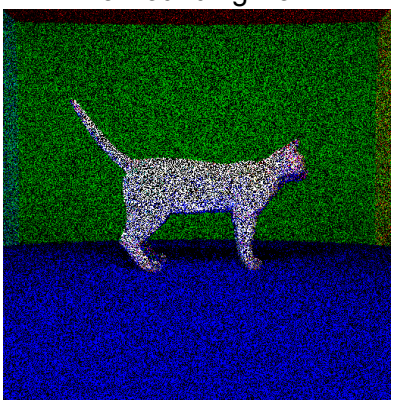
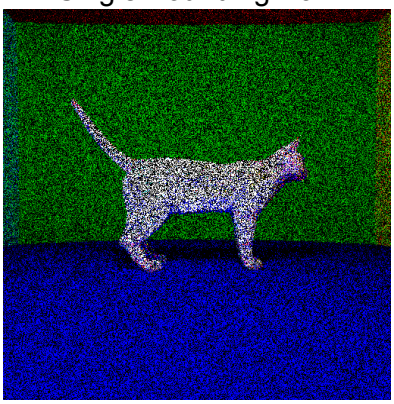
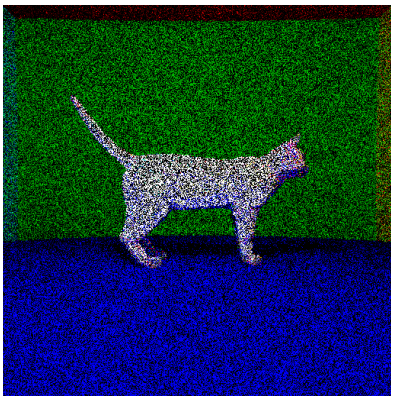
## Parallelization

In this section, we disable parallelization and compare rendering times.
My computer has an [Intel i7-6700HQ] processor with eight cores.

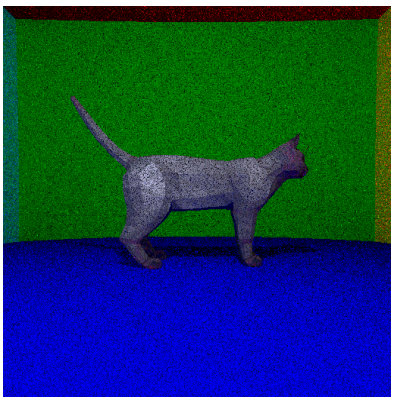
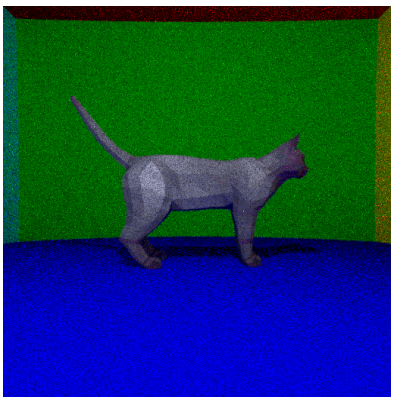
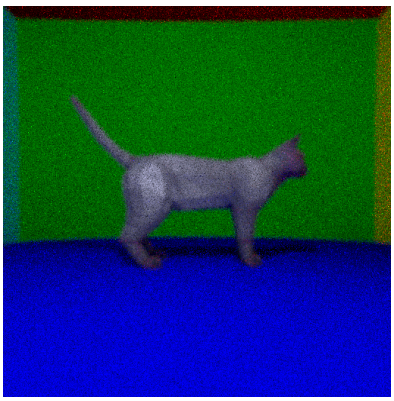
```
  1[||||||||||||||||||||||||||93.8%]     5[||||||||||||||||||||||||||93.2%]
  2[||||||||||||||||||||||||||92.3%]     6[||||||||||||||||||||||||||93.2%]
  3[||||||||||||||||||||||||||93.2%]     7[||||||||||||||||||||||||||92.1%]
  4[||||||||||||||||||||||||||93.3%]     8[||||||||||||||||||||||||||93.1%]
Mem[|||||||||||||||||||5.87G/7.54G]   Tasks: 104, 675 thr; 8 running
Swp[                          0K/0K]   Load average: 3.47 2.41 2.10
                                       Uptime: 1 day, 11:20:26
```

| With parallelization | Without parallelization |
| --- | --- |
|  |  |
| 16s (10 rays per pixel) | 110s (10 rays per pixel) |

## Triangle Mesh Bounding Box

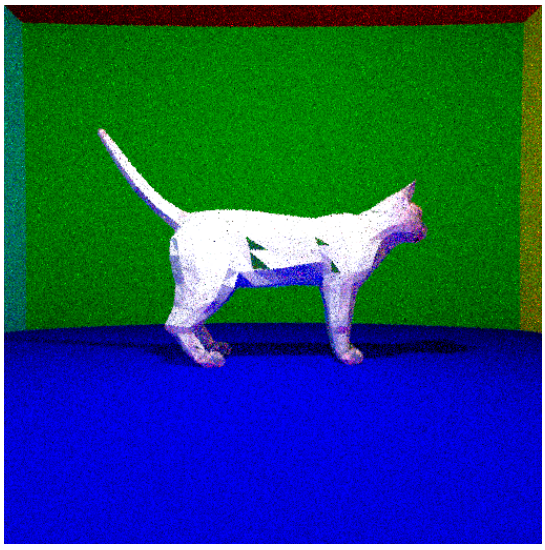
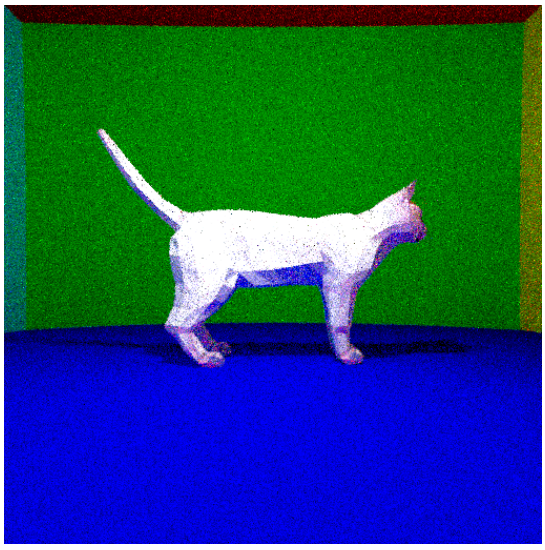
| No Bounding Box | Single Bounding Box | BVH |
| --- | --- | --- |
|  |  |  |
| 160.6s (1 ray per pixel) | 23.4s (1 ray per pixel) | 1.7s (1 ray per pixel) |

## Antialiasing

| Box Muller Stdev = 0 | Box Muller Stdev = 0.5 | Box Muller Stdev = 2 |
|---|---|---|
|  |  |  |
| 16s (10 rays per pixel) | 16s (10 rays per pixel) | 16s (10 rays per pixel) |

## BVH Issue

In my implementation of BVH, there seems to be a one-off error somewhere in my code since there are a few triangles missing in the rendering. I was not able to find the exact error in my code. However, I noted that when I adjusted the stopping criteria of build_bvh from "pivot_index >= ending_triangle - 1" to "pivot_index >= ending_triangle - 5", the cat looks normal. I understand that there is probably still a bug somewhere in my code and this is just a “dirty” fix.

| "pivot_index >= ending_triangle - 1" | "pivot_index >= ending_triangle - 5" |
|---|---|
|  |  |

Both images above still had very similar render times (16s with 10 rays per pixel). Although, perhaps the difference is only significant with a large number of rays per pixel.

I also played around with other free low-poly 3D .obj models I found online. Unfortunately, since I didn't implement rotation (only scaling and translation) for my triangle meshes, most of these models looked kind of boring from one single side. Here's a nice revolver pistol though. You can notice the nice soft shadows from spherical lighting too.