Benjamin Montagnes BX21 – G2

CSE306 - Assignment 1

The goal of this first assignment is to code a simple raytracer using C++ and improving it at each tutorial by implementing new features such as diffuse surfaces, mirror surfaces, refraction, indirect lighting, antialiasing and finally a simple triangle mesh rendering. In my implementation, I have the opportunity to change the image width and height, the number of rays, the ray depth, the light intensity and position, and finally the camera angle and position.

For the implementation, I wrote my different classes in different files to make the code cleaner and more efficient. I define the scene and the objects all in the main.cpp file. The ray.cpp file contains the Ray class as well as the BoundingBox class, the Geometry class and the Node class. These classes are all in the ray file as they are small classes and are all classes that defines the base of my raytracer. This is why ray.cpp is not included in my main.cpp file but it is included in the other files. The scene.cpp file contains the Scene class, the sphere.cpp file contains the Sphere Class and the vector.cpp file contains the Vector class. Finally, the resources from stb are found in the stb folder and the TriangleMesh class provided in the lecture notes is in the triangle.cpp file along with other functions I implemented to complete this class.

For the following results that I obtained, the camera angle and position as well as the light position or intensity did not change. Moreover, the ray depth was set to 5 and the size of the image to 512x512. I will now compare the different results I obtained in the different TD's as well as some interesting comparisons. I did not keep every image from every time I ran my raytracer so here are some results:

This first observation is a comparison between the first tutorial where we implemented a simple raytracer and rendered a white ball with some shadows and then the same ball but with everything applied to it. This tutorial showed me how raytracing works and how we launch rays from the camera center and look for the points where those rays intersect with the objects in the scene and put their color on a 2d image. For the shadows, we use the visibility of the intersection point with the lighting.

White ball with no indirect lighting

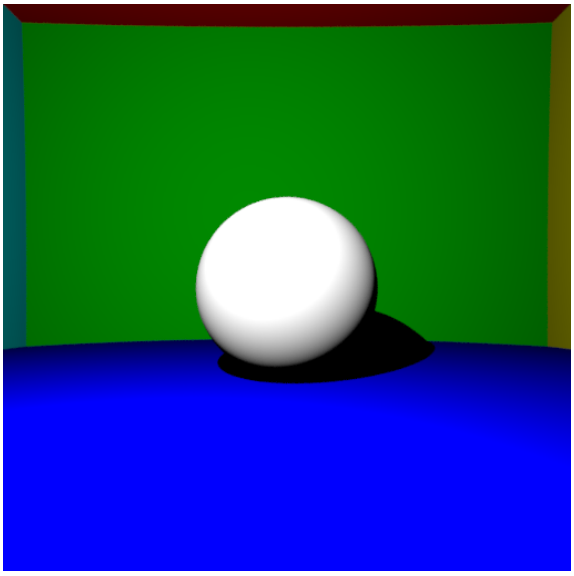

Image Size : 512x512
NB_PATH : 32
Runtime : 2 447ms

With indirect lighting

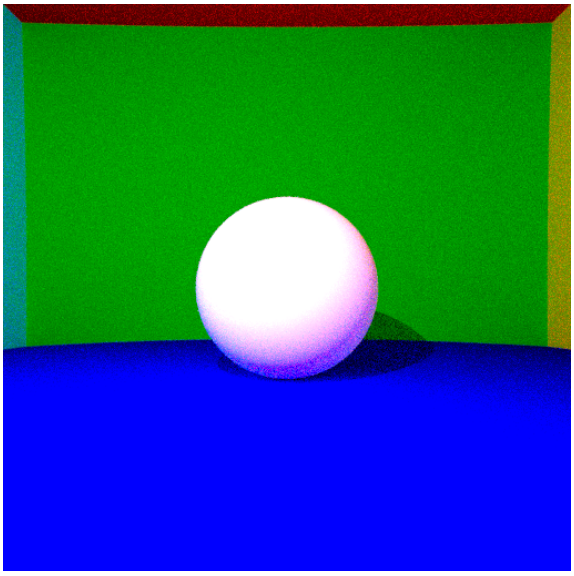

Image Size : 512x512
NB_PATH : 32
Runtime : 18 926ms

We can see that for the same settings, the rendering with no indirect lighting is much more efficient which shows us how indirect lighting has very big computational costs due to its recursive nature and I could really see how indirect lighting is thus a problem for real-time rendering applications.

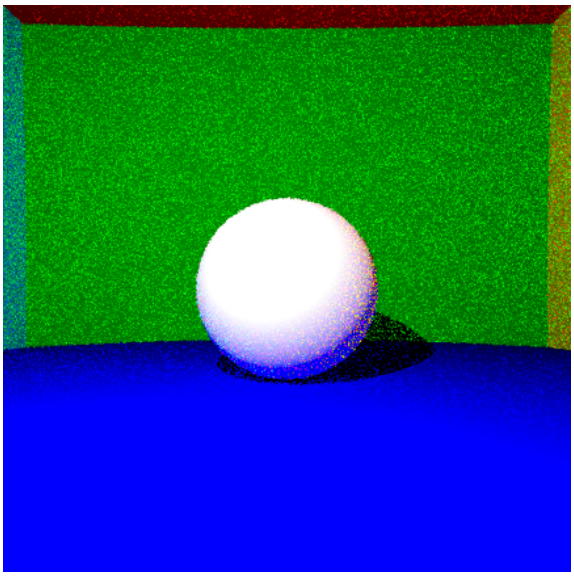

Image Size : 512x512
NB_PATH : 1
Runtime : 712ms

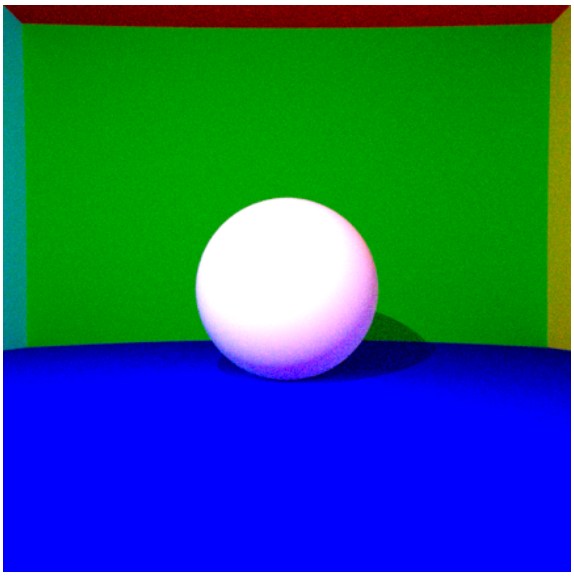

Image Size : 512x512
NB_PATH : 64
Runtime : 45 683ms

For indirect lightening, we performed a simple Monte-Carlo integration by sampling vectors around the normal. Finally, even though I did not show the difference in this report as it was too small to be seen, we added antialiasing to smoothen the surfaces and make the object more realistic.

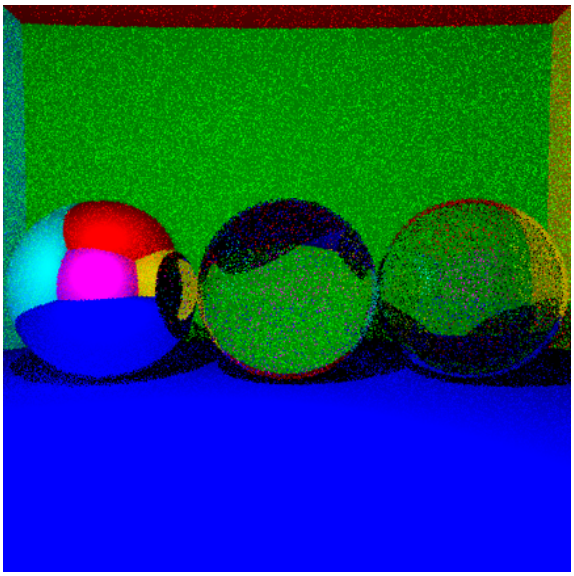

Image Size : 512x512
NB_PATH : 1
Runtime : 676ms

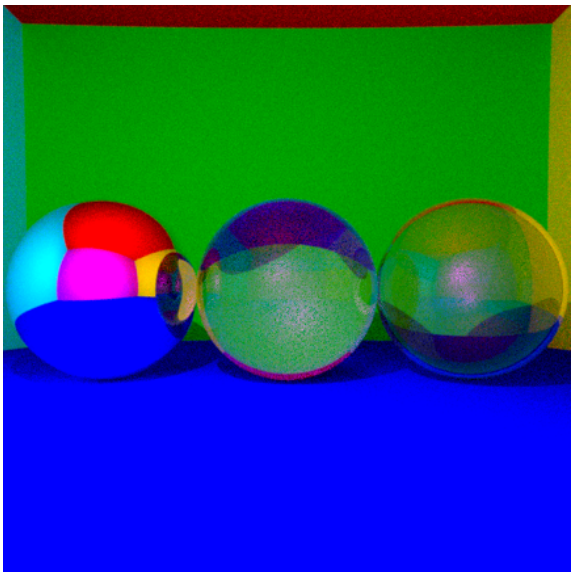

Image Size : 512x512
NB_PATH : 64
Runtime : 40 947ms

On these results, we can see that the number of rays increases the quality of the final image but also the runtime. For the reflection and the refraction, we just look at the intersection points after that the ray has bounced or passed through the surface of the ball. The colors of the final image were corrected by applying gamma correction.

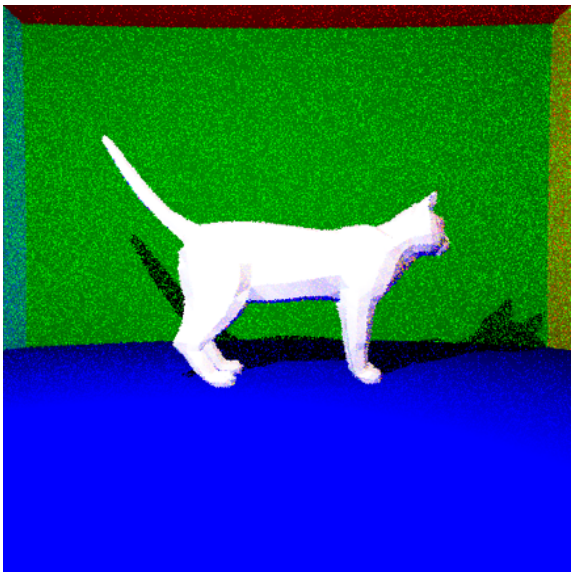

Image Size : 512x512
NB_PATH : 1
Runtime : 7 251ms

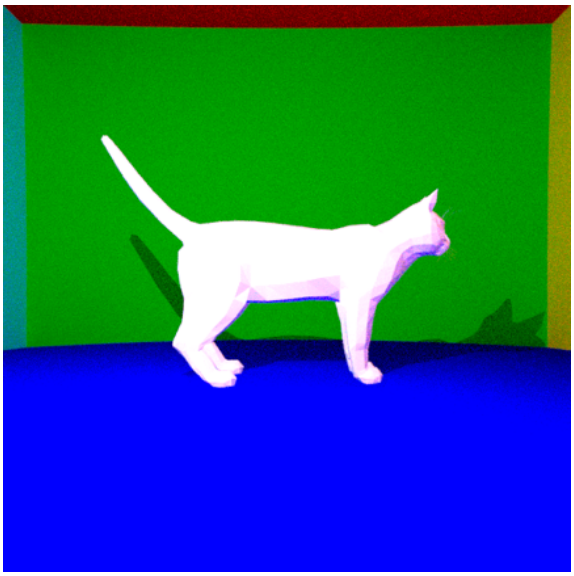

Image Size : 512x512
NB_PATH : 64
Runtime : 479 475ms

Finally, for the cat mesh, we used the Moller-Trumbore intersection algorithm as well as a single bounding box and a bounding volume hierarchy to reduce the computational costs. The other results not appearing in this report (such as the comparison without antialiasing) can be found in the result folder on my Github.