Assignment 1

In this assignment, we developed a raytracer in C++ that provides different features such a transparent or reflexives surfaces. We managed to implement diffuse surfaces, mirror surfaces, refraction, indirect lightening, antialiasing and simple triangle meshes rendering.

Throughout the assignment, we fixed the size of the image to 512x512 pixels, the ray depth to 5, the angle of the camera to 60 degree and the light intensity to 2e10.

I did not manage to implement parallelization. I understand the concept and I think that I know how to do it, but I couldn't make it work on my computer. I have a MacBook with the new chip, and it seems like that it is not compatible with the M1 chip. I put the runtimes as indication even though I know that they are completely off. From my understanding, implementing parallelization only demanded to add the line *"#pragma omp parallel for schedule(dynamic,1)"* just above the for loop where we launch the rays in the main file.

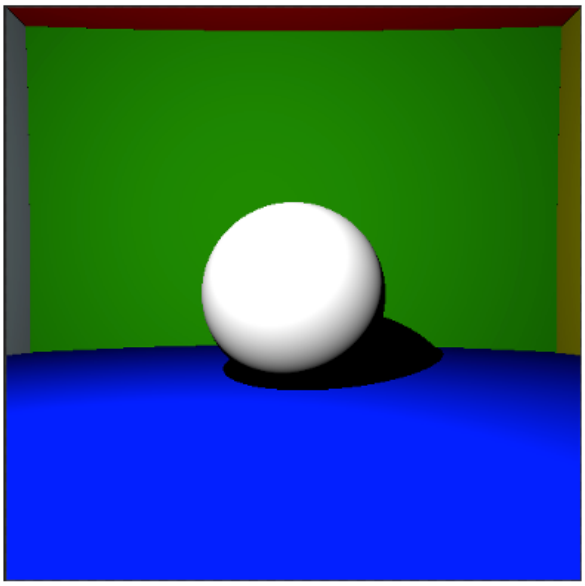
We created 8 different classes, each contained in a separate folder:

- Scene
- Geometry
- Sphere
- Light
- Mesh
- Vector
- Camera
- Ray

Diffuse, mirror and transparent surfaces

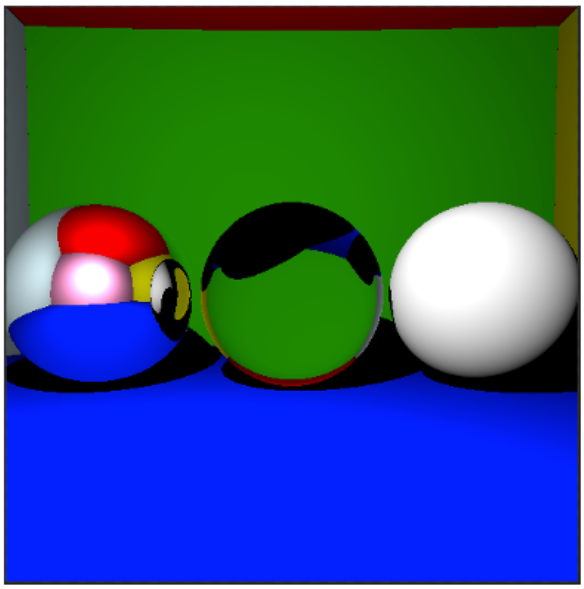
The first tutorial was to implement a simple raytracer and render a white sphere with a background and some shadows. In order to achieve this result, we launch rays from the camera center, then we look for points of intersection with the objects in the scene. Then, we just need to take the closest point of intersection and display the color of the object. To add shadows, we computed the visibility of the intersection point along with an intensity parameter. For the reflection and the refraction, we are interested in the intersection points after the ray has bounced or passed through the surface of the ball. We added some gamma correction to make the colors look more realistic.

White sphere with diffuse surface



- 512x512
- 112 ms

Mirror, transparent and diffuse surface

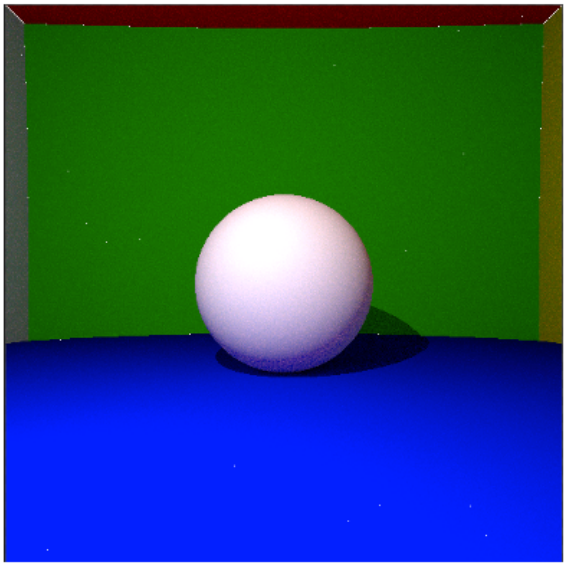


- 512x512
- 143 ms

Indirect lightening

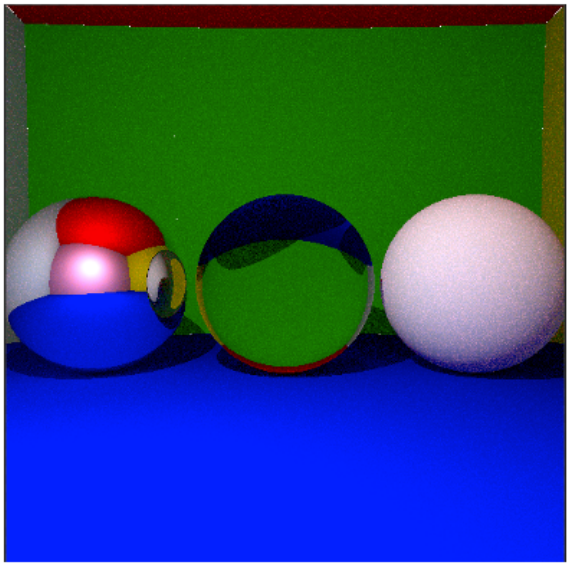
For indirect lightening, we used the Monte-Carlo integration technique to compute in real time the colors of the scene.

White sphere with diffuse surface



- 512x512
- 10 rays per pixel
- 30073

Mirror, transparent and diffuse surface

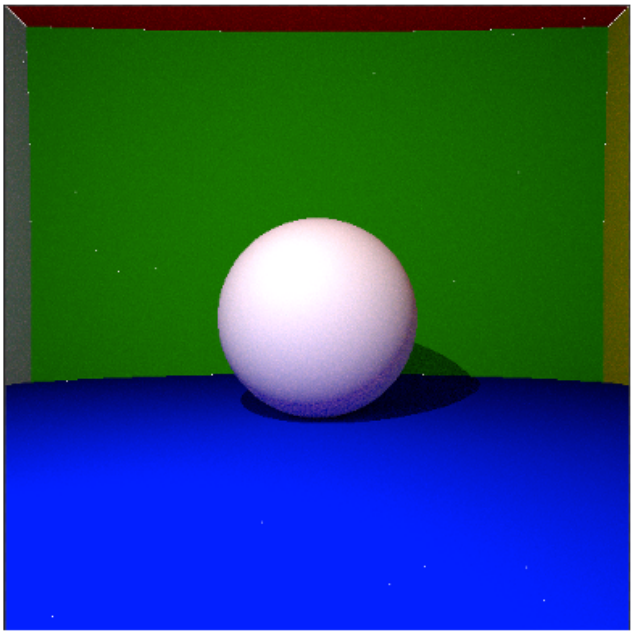


- 512x512
- 10 rays per pixel
- 34469

Antialiasing

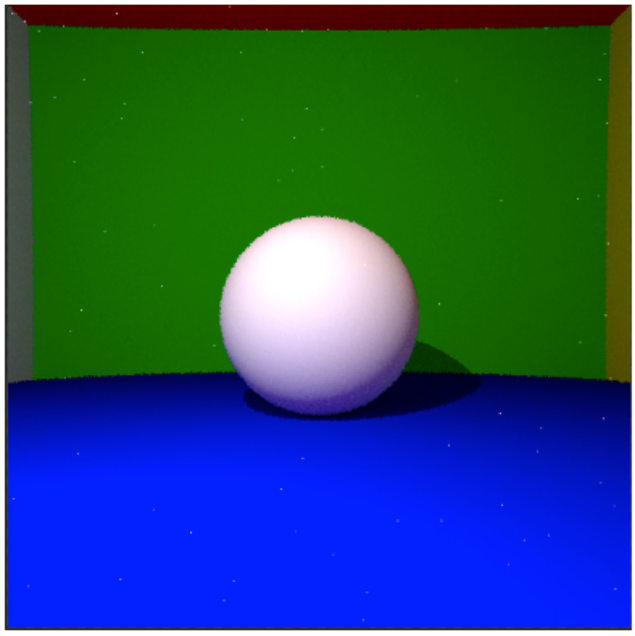
To smoothen the scene and give it a more realistic look, we added antialiasing.

White sphere with diffuse surface



- 512x512
- 10 rays per pixel
- No antialiasing

White sphere with diffuse surface



- 512x512
- 10 rays per pixel
- antialiasing

Meshes

To handle the ray/mesh intersection, we used the Möller-Trumbore intersection algorithm on. At first, we were going through all of the triangles in the mesh which was quite long. Then, we created a bounding box to lower the number of computations. Lastly, we used the Bounding Volume Hierarchy (BVH) technique to achieve decent performances.

Cat mesh

- 512x512
- 10 rays per pixel
- BVH
- 3 minutes