Assignment 1 : RayTracer

Todoskova Darya

May 2023

My project consists in a ray-tracer coded in C++. Its features include 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 images have the same dimension (512x512), with 64 rays per pixels and a maximum ray depth of 5. The final image was rendered in ...seconds.

Throughout this project, I used the PDF lecture notes, as well as `https://people.cs.kuleuven.be/~philip.dutre/GI/TotalCompendium.pdf`. I also revised lecture notes from INF443, a computer graphics course that helped me a lot during my research project in the field alongside Marie-Paule Cani (in which I implemented ray tracing from scratch as well).

1 Code Organization

All the necessary code can be found in a singular main.cpp file, which includes the vector class, intersection, bounding box, rays, and BVH nodes. There is an abstract parent class geometry with two different derived classes: one for spheres and another for triangle meshes. The primary difference between them is the way they handle intersections. Bounding boxes and BVH were only implemented for triangle meshes. There are also "utility" functions such as boxMuller for antialiasing and random_cos for the monte-carlo integration. The Scene class sets up a simple scene with walls, floor, and a ceiling.

2 Labs Breakdown

Lab 1

In this first lab, I rendered a basic sphere, and then 3 spheres side-by-side. I defined the basic classes (Vector, Sphere, Ray, Scene,...) in a main.cpp file. I then coded different methods in these classes to deal with ray-sphere and ray-scene intersections. All in all, I implemented diffuse and mirror surfaces and direct lighting and shadows, for point light sources.

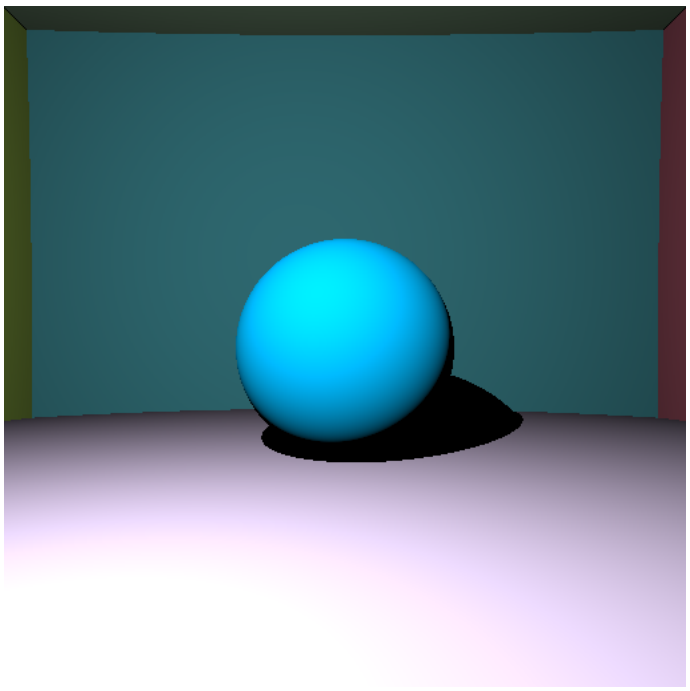

Figure 1: First basic rendered image (without gamma correction). Here, $I = 2.10^10$. About 150 lines of (verbose) code which runs in about 60ms without parallelization.

Finally, I worked on refraction and reflection before rendering the 3 spheres you can see below. The full sphere inverts the scene behind as it acts as a lens. The refraction index used is 1.5, corresponding to glass. By the way, I added each of these spheres (translated differently) in my final rendering with the cat.

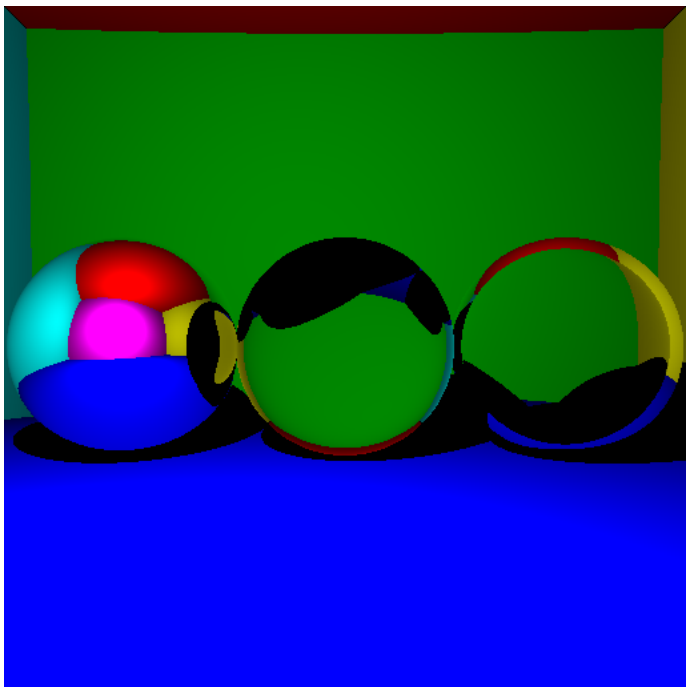

Figure 2: A sphere with reflection, a full sphere with refraction, and a hollow sphere with refraction. The image is computed in about 70ms (without parallelization) with about 280 lines of code. left:sphere with reflection; middle:refraction; right:hollow sphere with refraction.

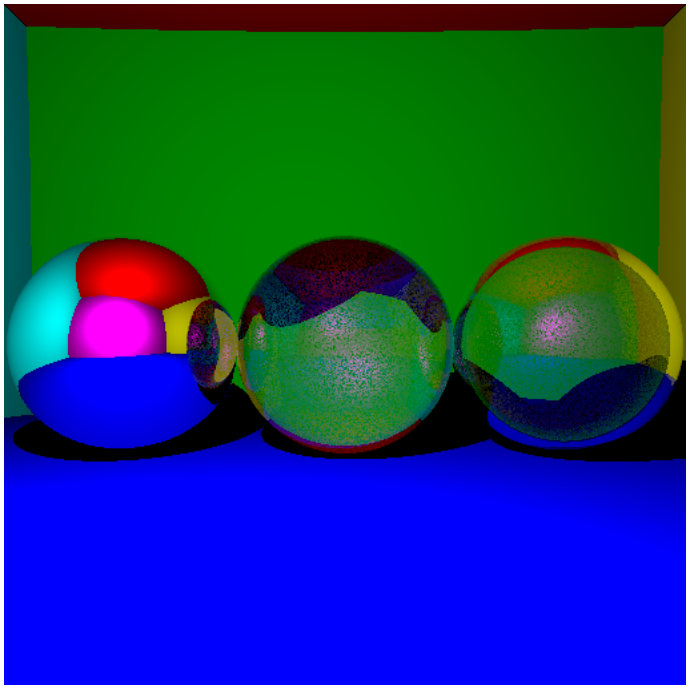

Figure 3: Same as Fig 2 but with Fresnel reflection. Duration: 29286.1ms

Lab 2

In this lab, I focused on indirect lighting for point light sources and antialiasing. I implemented the Monte Carlo numerical integration and added the indirect lighting to my path tracer (using random_cos). I then added antialiasing to improve the quality of the spheres :

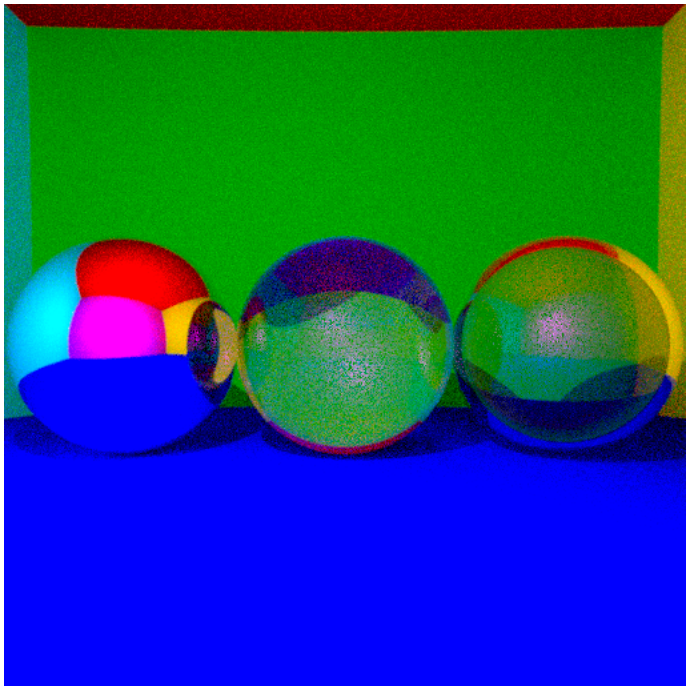

Figure 4: Same as Fig.3 but with Fresnel reflection, indirect lighting and antialiasing. Duration:132455.1ms)

3 THE CAT, a pretty big boi

Lab 3

Throughout this third lab, I focused on ray mesh intersection. I managed to render the cat by developing the meshes and the bounding box. I computed the bounding box of the mesh, and wrote a function for ray-bounding box intersection.

Lab 4

Finally, I spent the last lab working on the BVH. If the ray hits the bounding box of the mesh, we can further test if it hits the two bounding boxes containing each half of the mesh (and so on with a quarter of the mesh etc.). Hence, I built a binary tree, with the root being the entire mesh's bounding box. We then take the longest axis of the bounding box. Then for each triangle, we determine if its barycenter is within the first half or the second half of this axis. This determines two sets of triangles, for which we can compute their bounding boxes and that can be set as the two children of the root node. This process is recursively performed for these two children nodes, until some criteria is met (for instance, until the number of triangles in a leaf node is smaller than some threshold).

This is the final cat rendering :

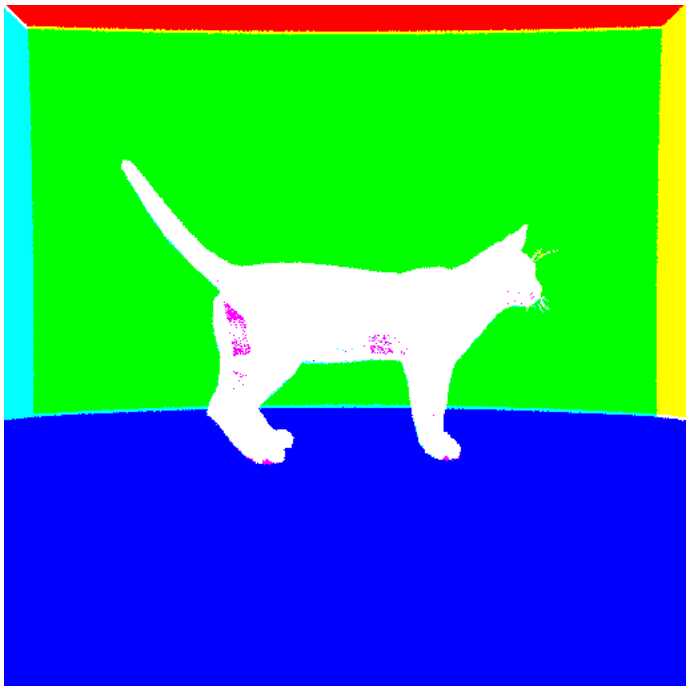

Figure 5: Cat scaled by a factor 0.6, translated by the vector (0,10,0), albedo is Vector (0.3,0.2,0.25), intensity 3e10, 64 rays, 5 light bounces, field of view 60 degrees, BVH, with antialiasing, Fresnel refraction and indirect lighting (time taken around 23 min)

The cat images above respect the values you gave on moodle. However, this is a rendering with a lower light intensity (I=1e7) :

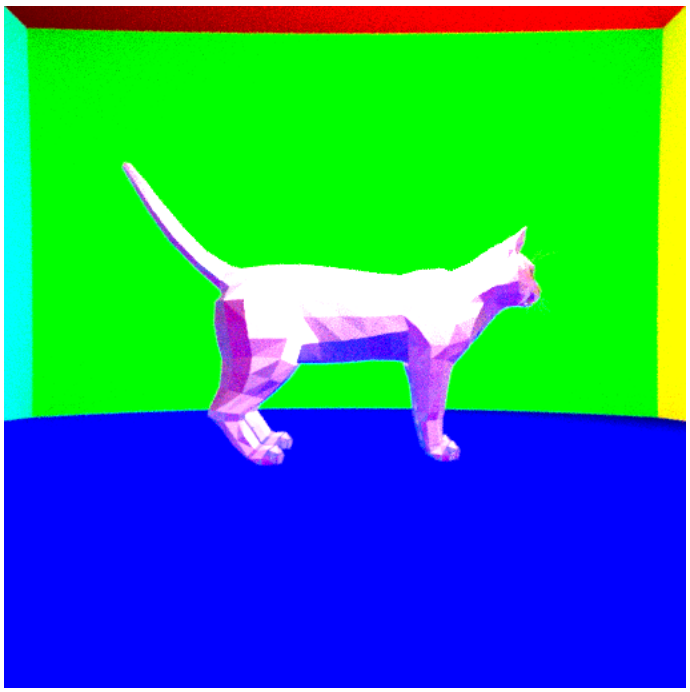

Figure 6: Same as above but with I=1e7. Function duration: 255124 ms.

Same as above but with a light source :

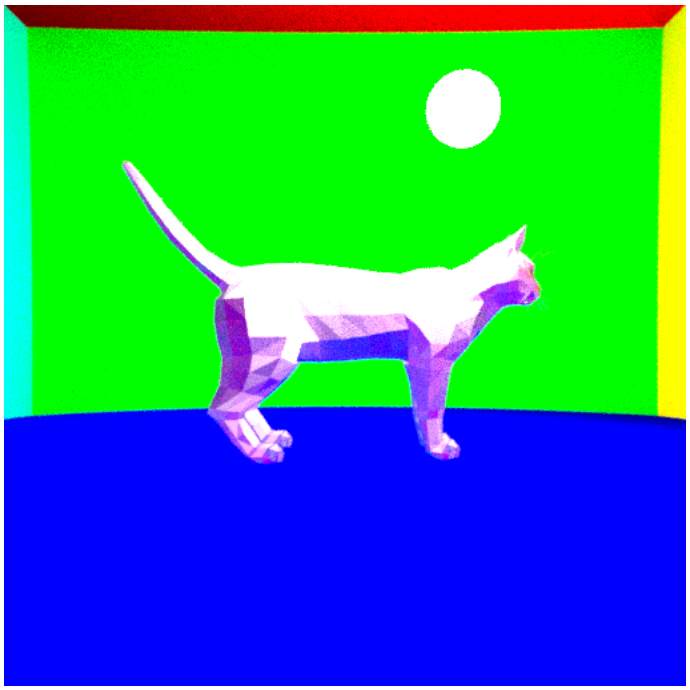

Figure 7: Light_center of (10, 20, 5), light_radius of 3, albedo of (1., 0., 0.). Function duration: 234451 ms.

Same as above but with the spheres : (inspired by the first lecture powerpoint in which there was a gun beside the cat's head)

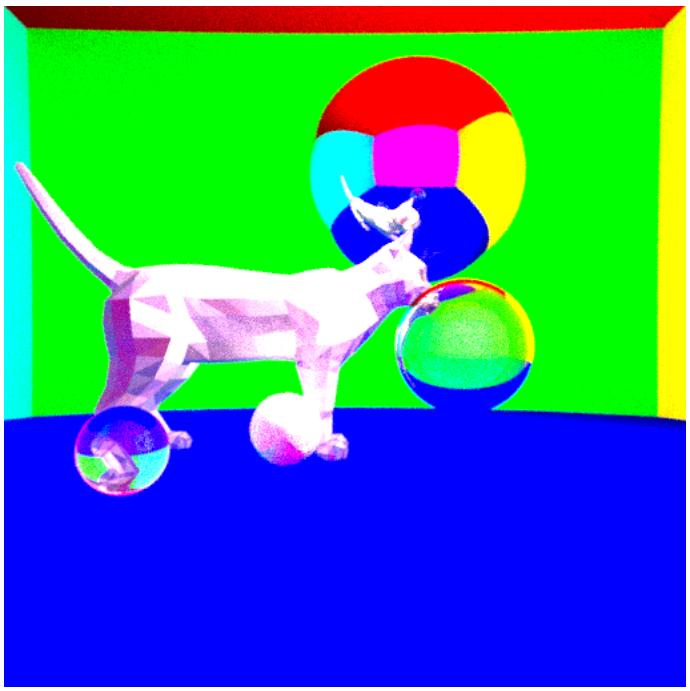

Figure 8: Translated cat with a white sphere, a reflective sphere, a solid refractive sphere and a hollow refractive sphere. Function duration: 96194 ms.

4 Giving the big boi a big friend

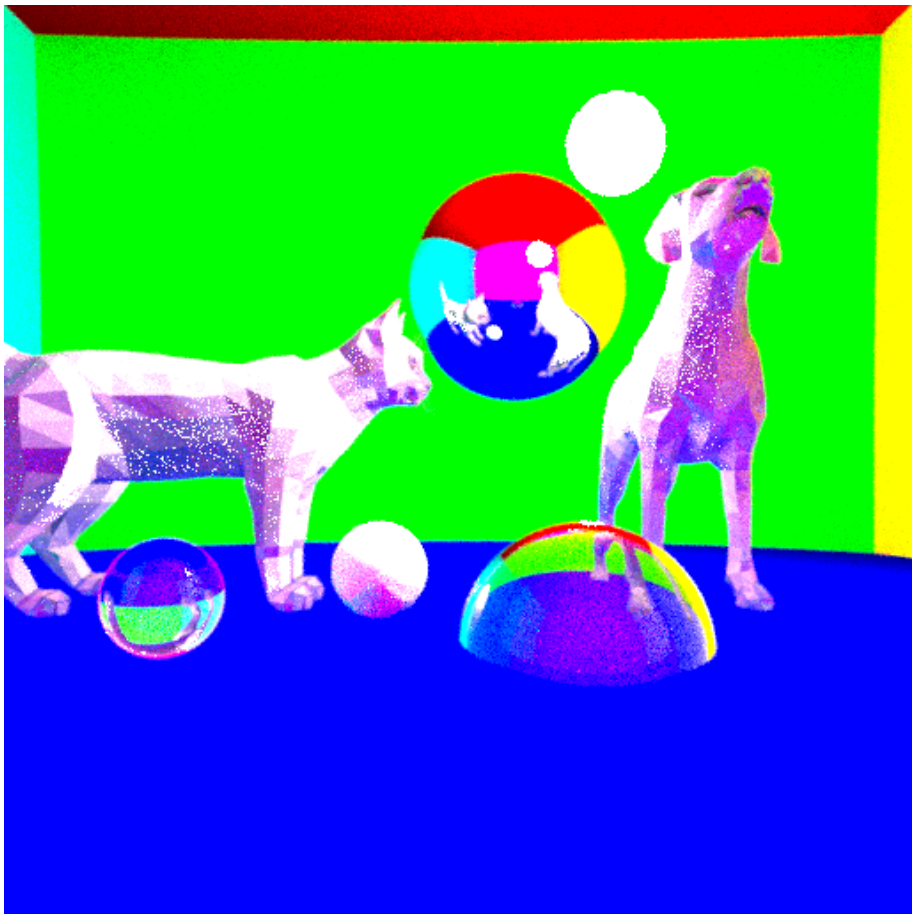

Figure 9: Dog scaled by a factor of 4 with a white sphere, a reflective sphere, a solid refractive sphere and a hollow refractive sphere. Function duration: 134989 ms.