Ray Tracer - Computer Graphics

Ángela Garcinuño Feliciano

April 29, 2025

The objective of this project was to build a ray tracer. Starting from simple implementation with only spheres, we added different surface types (reflection and refraction), indirect lighting and antialiasing. We then implemented meshes, and improved performance with bounding boxes and BVH algorithm.

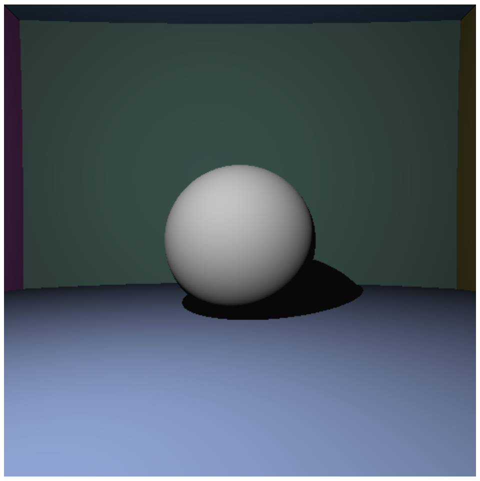
This project was built using polytechnique computers. Therefore, the time taken for each image is affected by the number of people using that computer at the same time. For example, at a certain time, it took 6 minutes to generate the image with the BVH implementation, while later it took only 20 seconds.

1 Basic Ray Tracer

The basic implementation is very simple. We begin with a scene which contains the information of light sources (intensity and location) and objects (center, radius and colour for the basic sphere). We then add a camera (location and angle) from which we send rays in a determined direction (depending on the angle). For each of these rays, we run a ray-sphere intersection procedure, where we determine if the first sphere the ray hits (if any). We then check if there is an object between this point and the light source (to determine if it's a shadow), and return a colour depending on the light intensity, angle and the sphere's colour. Each ray represents a pixel in our image.

2 Indirect Lighting

We began with a scene containing several spheres, the camera and a light. With a straightforward ray-sphere intersection we obtained the first images. The first improvement was indirect lighting, where we send rays from the intersection point recursively, obtaining a more realistic image.


(a) Without indirect lighting. Time taken: 150 milliseconds.

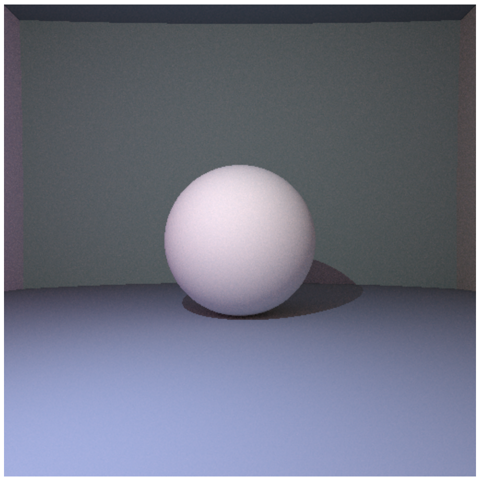
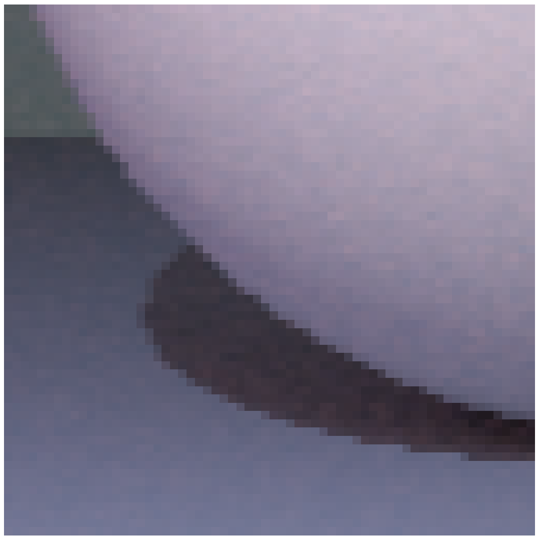

(b) With indirect lighting (5 depth). Time taken: 12 seconds.

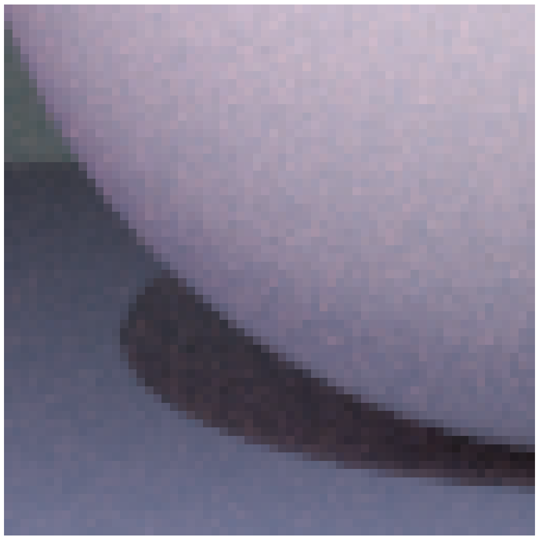
Figure 1: Comparison of direct and indirect lighting in a 512x512 image.

3 Antialiasing

We then implemented antialiasing to obtain smoother colours and shapes. To do this we send several rays for each pixel with a small, random difference in the direction, and take the average.



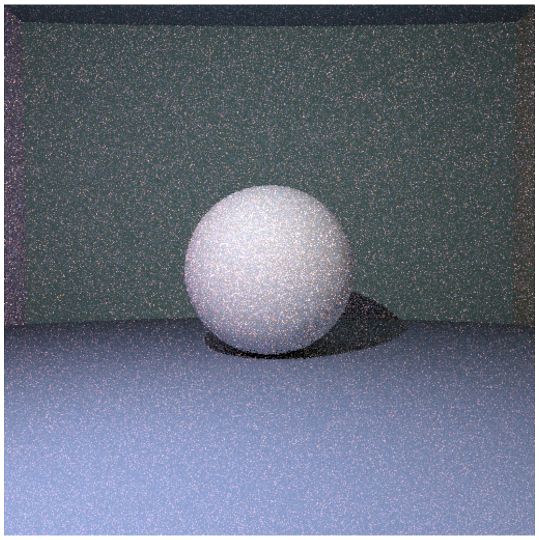
(a) Without Antialising (same image as indirect). Time taken: 12 seconds



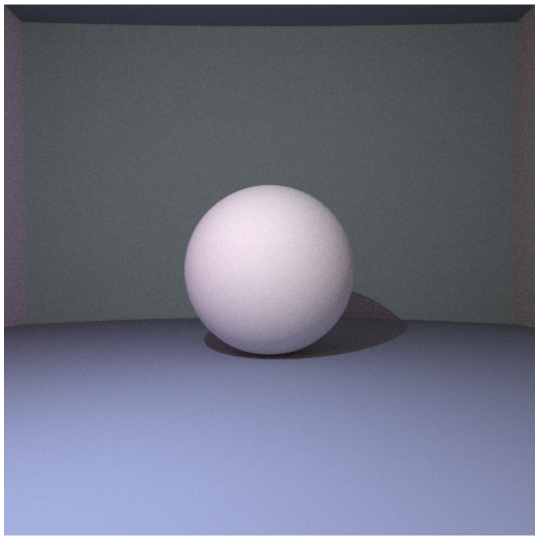
(b) With Antialising (64 rays/pixel). Time taken: 10 seconds

Figure 2: Comparison of Antialising in a 512x512 image, 5 depth.

Since for antialiasing we send random rays for each pixel, sending a single ray results in an image where certain pixels have random colours and there is no smooth colouring. It is therefore necessary to send multiple rays. More rays makes a smoother image but it also takes longer to run.



(a) Antialising (1 rays/pixel). 512x512 image, 5 depth. Time taken: 150 milliseconds



(b) Antialising (64 rays/pixel). 512x512 image, 5 depth. Time taken: 10 seconds

Figure 3: Comparison of Antialising in a 512x512 image.

4 Mirror Surfaces and Refraction

We also implemented different types of surfaces. Mirror surfaces (leftmost sphere) send a ray from the intersection point as if it were the camera. Surfaces with refraction (center sphere) let the ray pass through, modifying it's direction (snell's law). We implement a hollow sphere by placing a smaller (reversed) sphere inside another one (rightmost sphere).

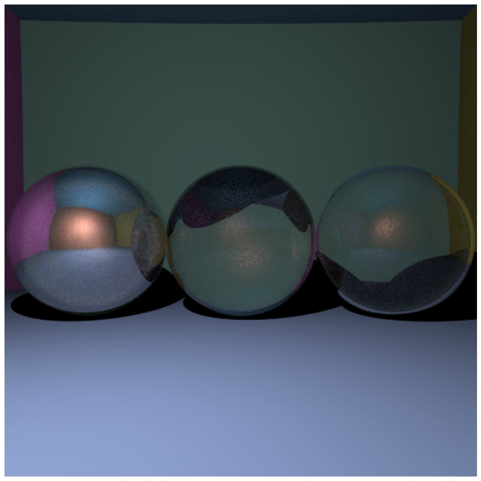


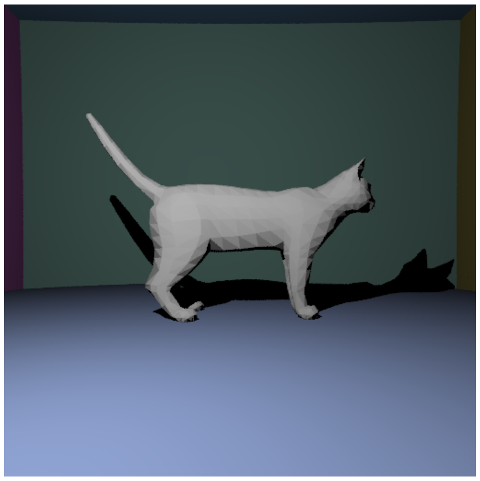
Figure 4: 512x512 image, 64 rays, 5 depth. Time taken: 9 seconds

5 Meshes

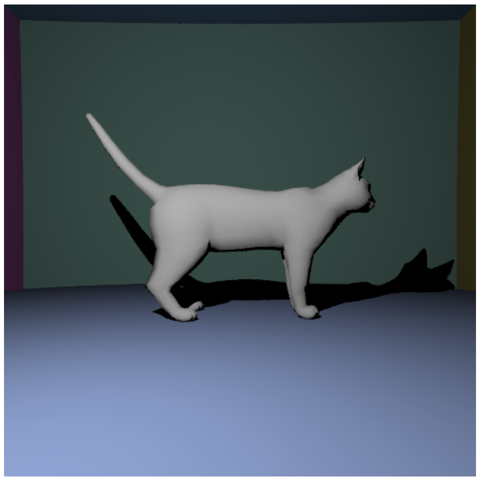
We then implemented meshes, using .obj files to obtain new shapes. Meshes are built through many triangles, so the ray-mesh intersection algorithm checks ray-triangle intersection for each triangle in the mesh and returns the best (closest) point. This process is extremely slow, and the triangles are visible on the resulting surface.

6 Interpolation of Normals

We smoothen the image with interpolation of normals, where we adapt the normal taken for the intersection point.



(a) Without Interpolation.



(b) With Interpolation

Figure 5: 512x512 image, 5 rays/pixel, depth. Time taken: 2 minutes 20 seconds.

7 Bounding Box

To speed up the algorithm, we first implement a bounding box, to reduce the number of rays for which we check the mesh for intersections. We find the smallest box containing the whole mesh and check ray-box intersection before checking for all the triangles. Although there is a speed-up, it is still relatively slow.

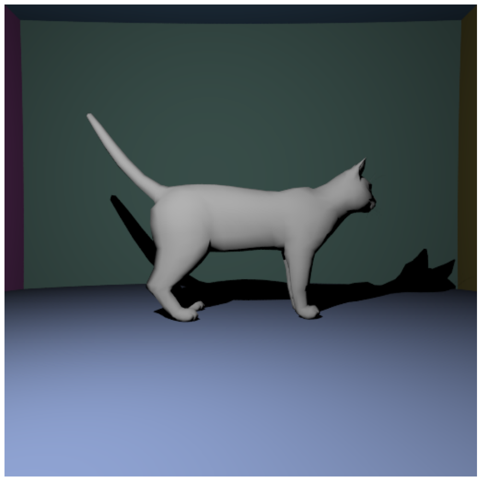


Figure 6: 512x512 image, 64 rays/pixel, 5 depth. Time taken: 10 minutes.

8 BVH algotithm

For further improvement, we implemented the BVH algorithm, splitting the mesh triangles into smaller bounding boxes using a binary tree structure. Each of the tree's node is associated to a bounding box and a set of triangles. When checking for ray intersection, we use a depth-first traversal of the tree and only check ray-triangle intersection at leaf nodes for the triangles associated to the node. This greatly reduces the number of triangles we check the intersection for, improving performance. We also use indirect lighting to obtain soft shadows.

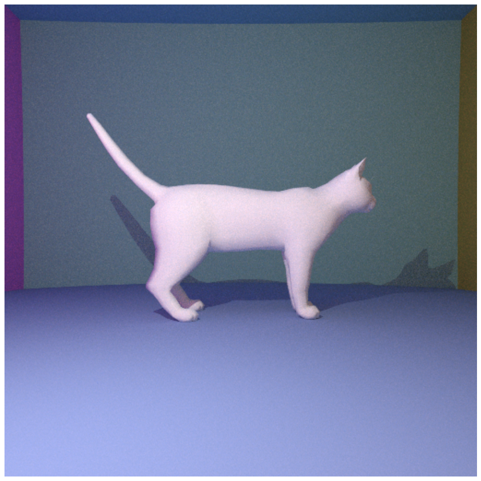


Figure 7: 512x512 image, 64 rays/pixel, 5 depth. Time taken: 19s.

9 Final Image

We finished by adapting the code to have multiple light sources, and generated a final image. This image has a transparent (refraction) cat, a mirror sphere and two light sources (one behind the sphere and one above to the right).

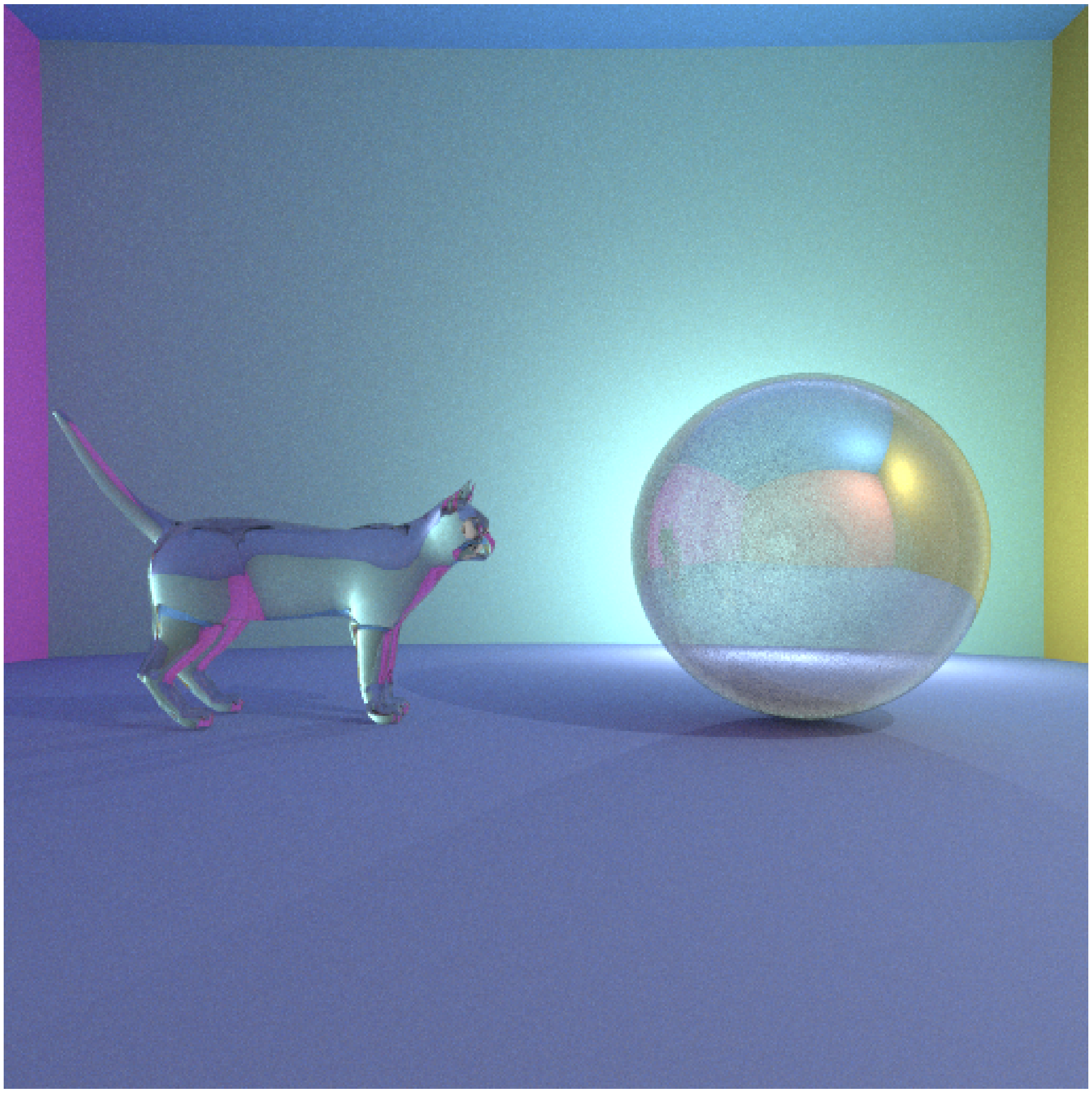


Figure 8: 512x512 image, 128 rays/pixel, 5 depth. Time taken: 34s.