Assignment 1: Project Report

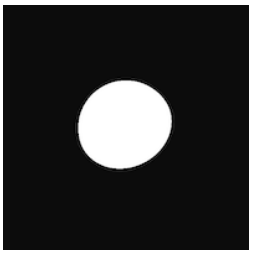
John Levy
May 9, 2023

I. Introduction

This report will present an overview of the raytracer project developed during the first four labs of the CSE306 – Computer Graphics course. In the introduction we will discuss the general concept that lies behind the raytracer model before diving into the specifics of each implemented feature. We have divided those features in two parts: first, the ones that apply to any kind of object we wish to render, and second, the ones that are more specific to the rendering of meshes.

The goal of the raytracer is to create a 2D-image representing a 3D-scene as if it was seen through the lens of a camera. The image that we are rendering is the projection of this scene on a virtual screen located in front of the camera.

To do so we compute the coordinates of each pixel on the image as a function of the initial position of the camera and the given field of view angle named alpha in the implementation. We shoot a ray towards each pixel and compute the intersection with any object between the camera and the virtual screen behind. If there is an intersection we use the Lambertian model to retrieve the color value of the corresponding pixel. By taking into account the space coordinates of the objects, we will also be able to compute the shade areas and render different lighting intensities. The most basic drawing one can obtain from this model is one of a white circle on a black screen shown in the figure below.



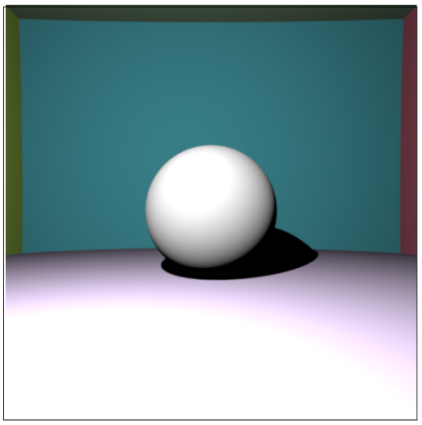
II. Overview of the Implementation

The implementation of the raytracer heavily relies on a Vector class used to represent 3-dimensional vectors; we build upon this class by creating Ray and Sphere classes. The background of the scene is made up of very large spheres centered far away from the camera to give the impression of walls, floor and ceiling stored in the Scene class. In the following sections, we present examples of the various features implemented in our raytracer (computed using the most advanced version of the code, unless otherwise specified).

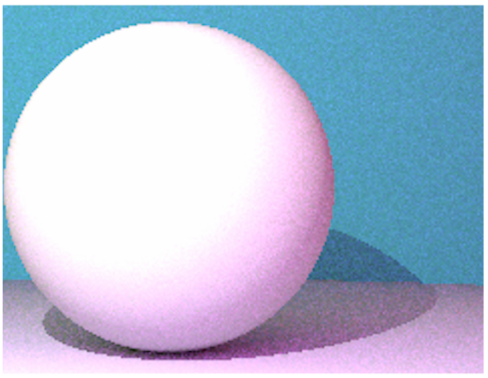
III. General Features

A. Lighting and shadows

The implementation of shadows is the first challenge one encounters when building a ray tracer. Seeing as the implementation of shadows feels necessary to us to obtain a realistic image, we have not shown an example with no shadows. The example below shows the rendering of a white sphere without indirect lighting on the left, where we can see that the shadow is solid and completely dark; while the example on the right has indirect lighting. With indirect lighting the shadows are much softer, which comes from the fact we recompute the direction of the ray using importance sampling where the light contribution is higher. In addition, the example on the right is implemented without antialiasing, that is, a way to make the rendering smoother by avoiding discontinuity between adjacent pixels using a random sampling.



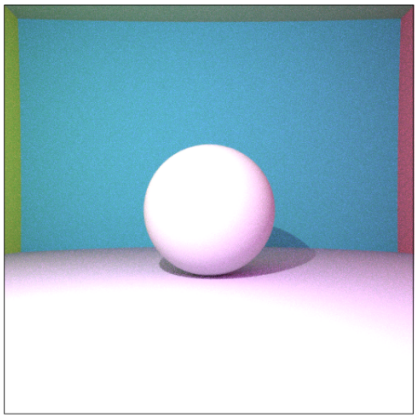
White Sphere (no indirect lighting), 0.9 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of transparent sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.



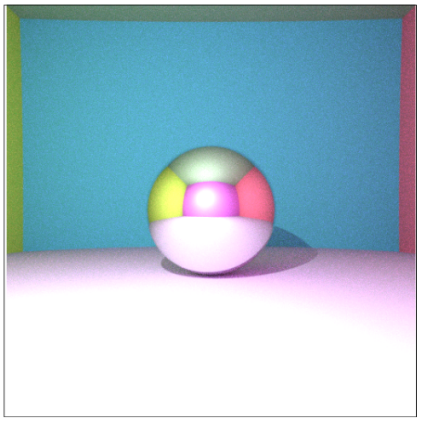
White Sphere (no antialiasing), 0.9 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of transparent sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.

B. Types of surfaces

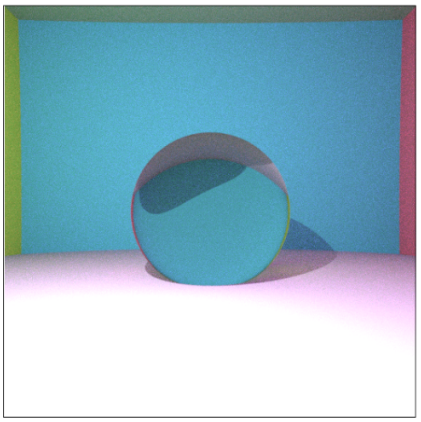
We can implement different types of surfaces by modifying the normal components of the intersections then used to re-compute the direction of the rays after an intersection during the color interpolation.
For mirror surfaces, we recompute the ray according to the laws of reflection.
For transparent surfaces, we recompute the ray according to the laws of refraction.
Finally, hollow objects can be represented as two objects centered on each other, with the outer one having a negative normal. Let us add that this re-computation of the color can sometimes lead to infinite recursions, which is why rays can only bounce a given number of times between objects before exiting the recursion. The images are shown below.



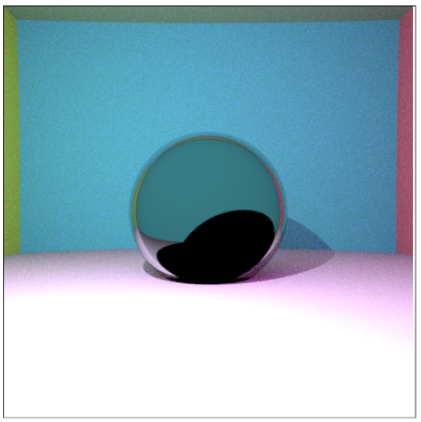
Simple white Sphere, 3.9 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.



Mirror Sphere, 3.8 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of mirror sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.



Transparent Sphere, 3.6 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of transparent sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.



Hollow Sphere 3.7 seconds of computation. 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of hollow sphere: Vector(0.5 0.5 0.5). Light intensity: 3E10, f.o.v = 60 degrees.

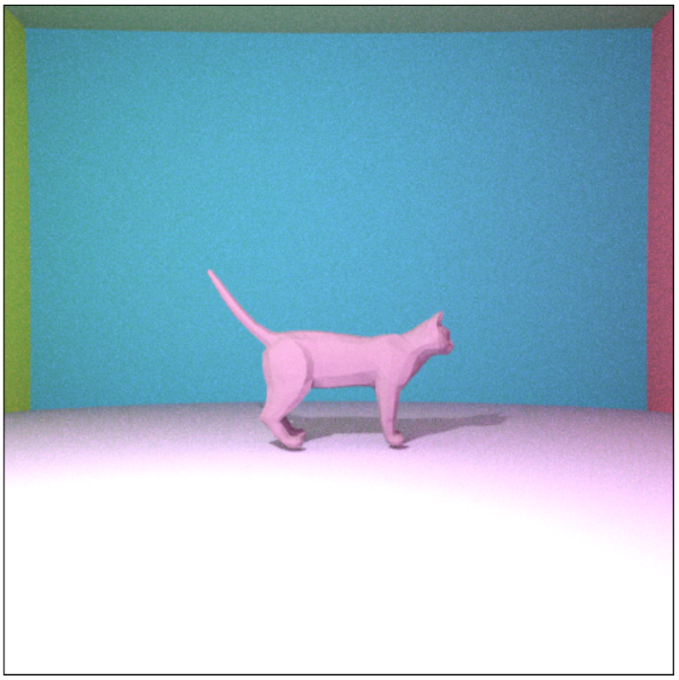
IV. Mesh specific features

Meshes are represented as a collection of triangles, each triangle made up of three vertices. To compute the intersection between a ray and a mesh, we thus iteratively compute the intersection between the ray and all the triangles in the mesh.

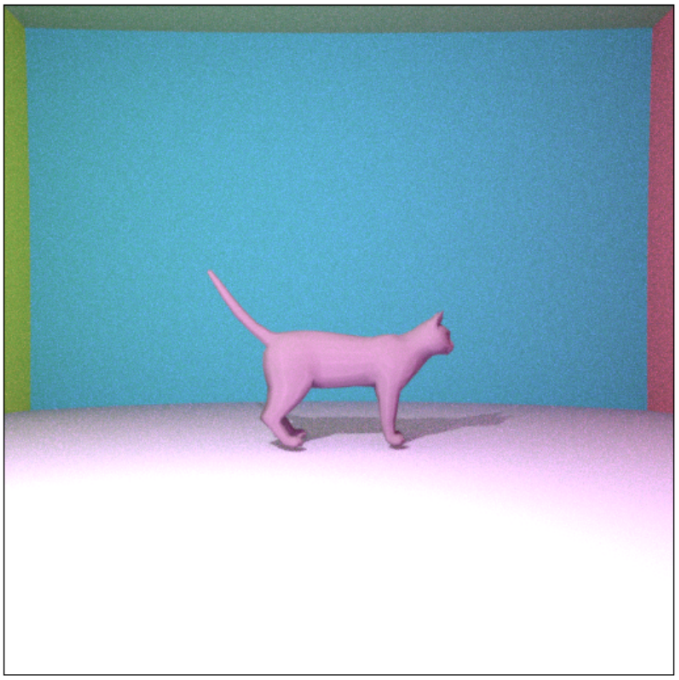
Due to the large number of vertices that usually make up a mesh – such as the cat we present here, these computations can quickly become very time consuming, even with parallelism. This motivates the implementation of the Bounding Box (essentially computing a virtual box surrounding the mesh and only computing the ray-mesh intersection if the ray hits within that box). The Bounding Box can be upgraded using the Bounding Volume Hierarchy structure. This structure recursively computes the Bounding Box of smaller and smaller sets of triangles, so that the intersect method only searches for intersections in those small sets rather than in all the space containing the mesh. The speed-up obtained from such a structure is significant (of order 10^2).

Finally, we implemented the phong interpolation of normals, which allows a smoother rendering of an object mesh, using the normal components of vertices when computing the normal of an intersection.

In the descriptions of the examples shown below, no bounding box means no acceleration structure at all (except parallel code), while the standard times are computed with the full bounding volume hierarchy structure.



Cat Mesh without phong interpolation
45.69 seconds of computation (30 minutes without bounding box and only one ray).
60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of cat: Vector(0.3 0.2 0.25). Light intensity: 3E10, f.o.v = 60 degrees.



Cat Mesh with Phong interpolation 48.43 seconds of computation (30 minutes without bounding box and only one ray). 60 rays per pixel, 512 x 512 image, 5 bounces. Albedo of cat: Vector(0.3 0.2 0.25). Light intensity: 3E10, f.o.v = 60 degrees.