CSE306 Computer Graphics - Assignment 1

Francisco Moreira Machado

May 14, 2024

1 Introduction

In this project, we provide a raytracer in C++ that projects 3D scenarios into 2D images. Among the key features implemented are:

- Rendering of Spheres and Triangle Mesh-based objects.
- Diffuse, Mirror, and Refraction surfaces for objects.
- Direct lighting for point light sources.
- Approximation for Fresnel law for refraction surfaces.
- Indirect lighting for point light sources with Monte Carlo Estimations.
- Antialiasing.
- Pinhole and depth of field camera models.
- Bounding Volume Hierarchy (BVH) optimization for rendering mesh-based objects.
- Phong interpolation for smoother rendering of mesh-based objects.

These will be showcased later in this report.

2 Implementation Details

The main logic behind the rendering and raytracing is implemented in class `Scene` in files `scene.`{`h, cpp`}. Notably, in `scene.h` most of the global features such as indirect lighting, antialiasing, and depth of field can be toggled on and off, as well as the key parameters of rendering can be modified. Indeed, `Scene` can render any object that implements class `Shape` defined in `shape.h`.

Both `Sphere` and `TriangleMesh` are instances of interface `Shape`, which only enforces that shapes implement an `intersect` function, that takes a `Ray` and yields information about whether the ray intersects the shape and properties of this intersection. The general class also ensures information about the material of the shape are included. `Sphere` and `TriangleMesh` are implemented in the files `sphere.`{`h, cpp`} and `mesh.`{`h, cpp`}, respectively. We can easily toggle the usage of BVH or Phong interpolation through booleans of the class `TriangleMesh` in `mesh.h`.

Finally, we extend the template for the `Vector` class in `vector.`{`h, cpp`} for convenience of use, implement the `Ray` class alongside the `Shape` class, and create a `random.h` that implements useful thread-safe distribution functions for the random simulations.

3 Examples of Features

For this section, unless explicitly stated otherwise, we use 100 rays per pixel and a maximum number of 5 bounces per ray. We will showcase the features incrementally here.

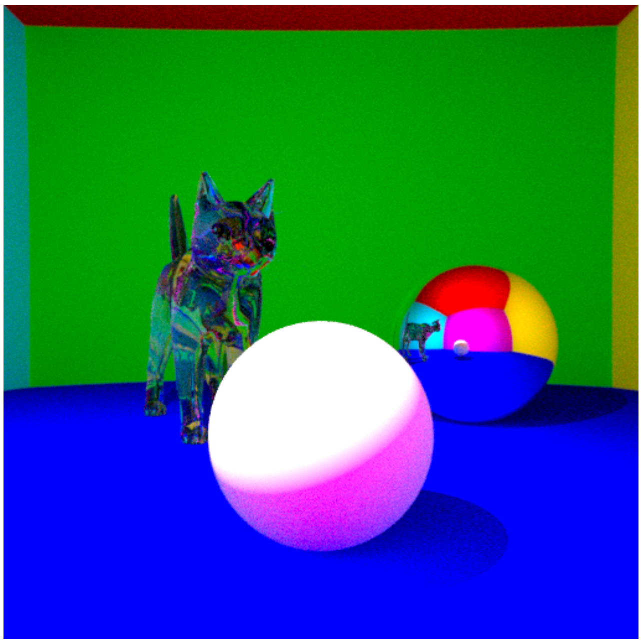

Figure 1: Image showcasing most of the features implemented by the ray tracer. It took 77.8s to render with 8 bounces per ray.

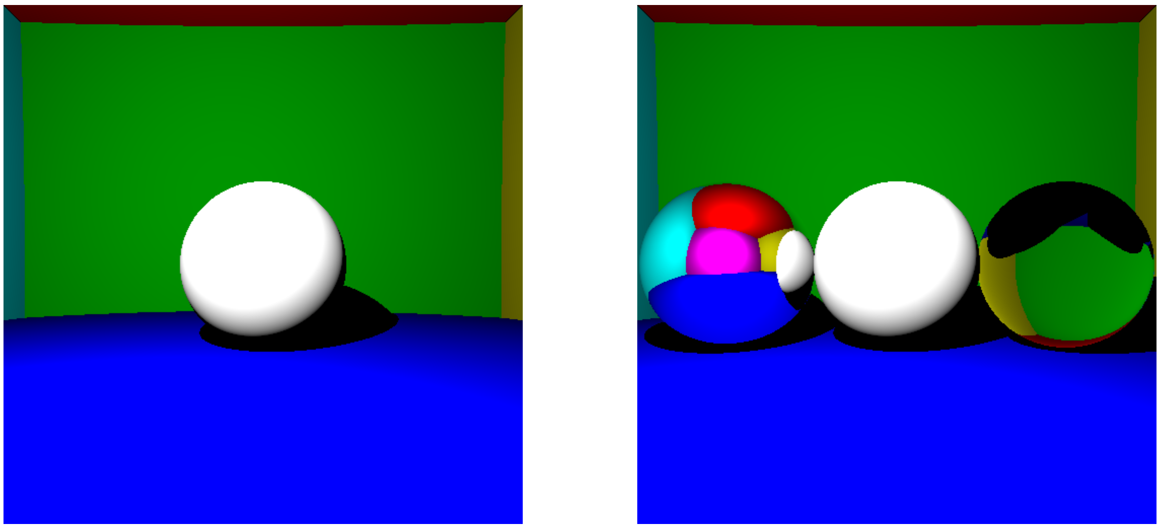

Figure 2: **Direct Lighting and Diffuse, Mirror, and Refraction surfaces.** On the right we have the most simple setting, with only direct lighting on a single diffuse sphere. On the right we have also only direct lighting with Mirror, Diffuse, and Refraction surfaced spheres, respectively. The left image took 2.0s to render and the right one took 2.8s.

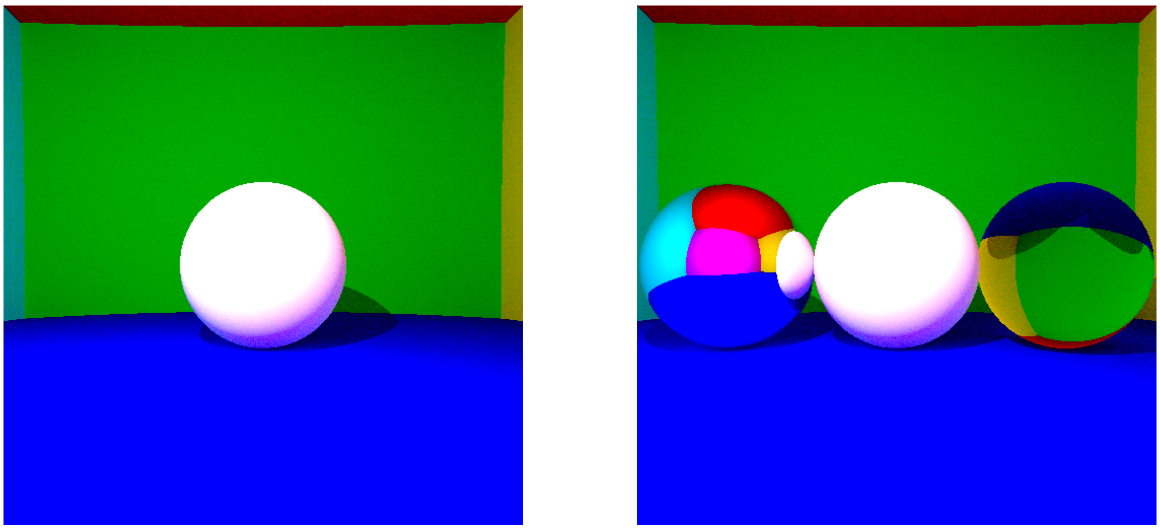

Figure 3: **Indirect Lighting.** Other than what was showed in the previous figure, we add indirect lighting to both images, focusing especially in the white diffuse sphere and the shadows projected on the refraction sphere. The left image took 15.6s to render and the right one took 16.9s.

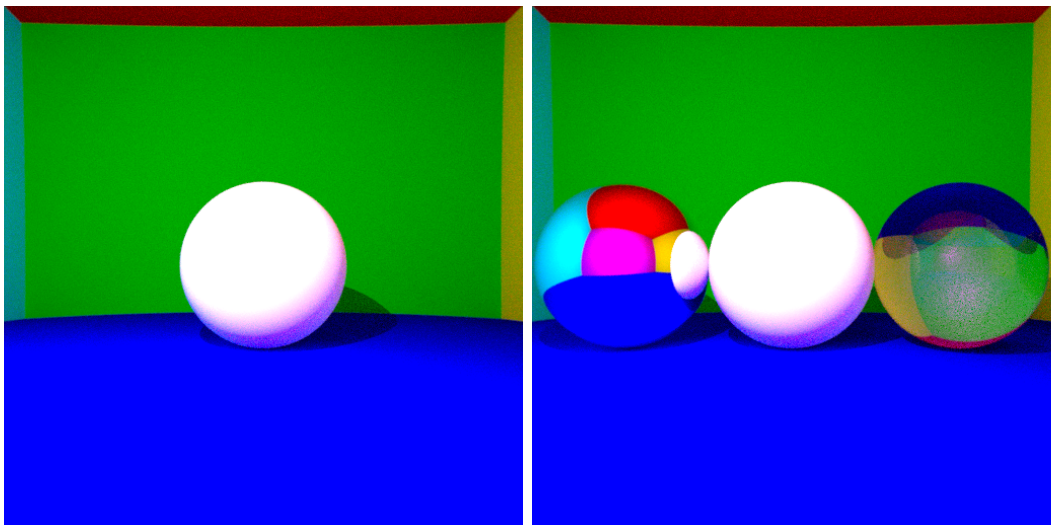

Figure 4: **Antialiasing and Fresnel.** Incrementally on the previous images, here we show the impact of antialiasing on both images and Fresnel on the right image's refraction sphere. The left image took 16.2s to render and the right one took 17.3s.

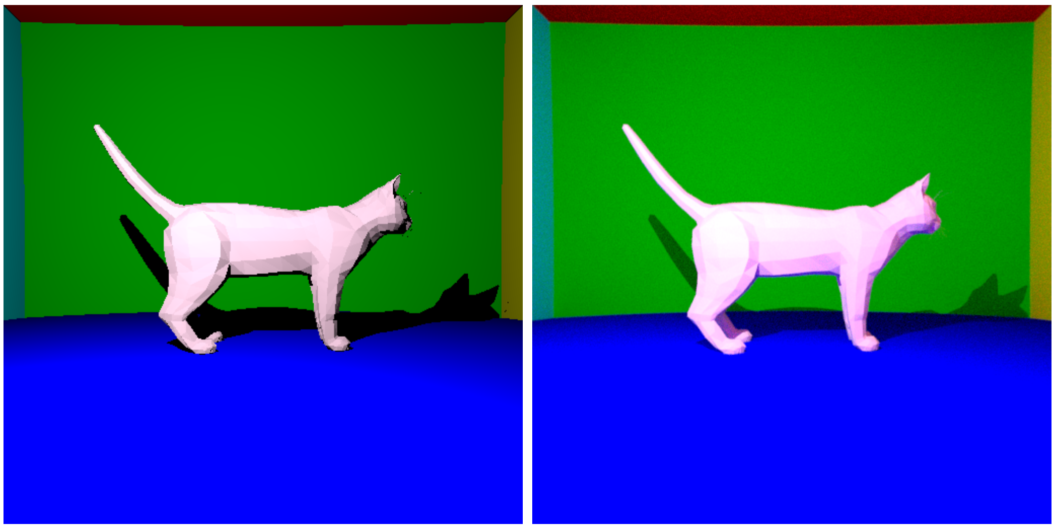

Figure 5: **Mesh Rendering.** Rendering of the Triangle Mesh based cat figure. The left represents the cat with direct lighting and nothing else, while the right has antialiasing and indirect lighting enabled. Using the BVH optimization, the left image took 11.3s to render and the right one took 46.3s.

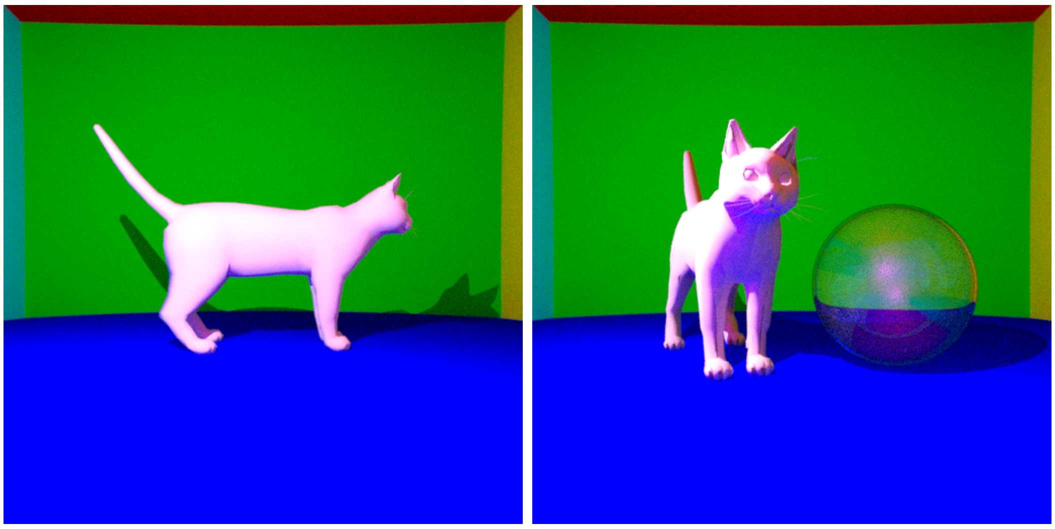

Figure 6: **Phong Interpolation.** The cat rendered with Phong interpolation, indirect lighting, and antialiasing turned on. On the right an example of a cat and a hollow sphere with Fresnel. Using the BVH optimization, the left image took 47.3s to render and the right took 120.1s with 10 bounces per ray.

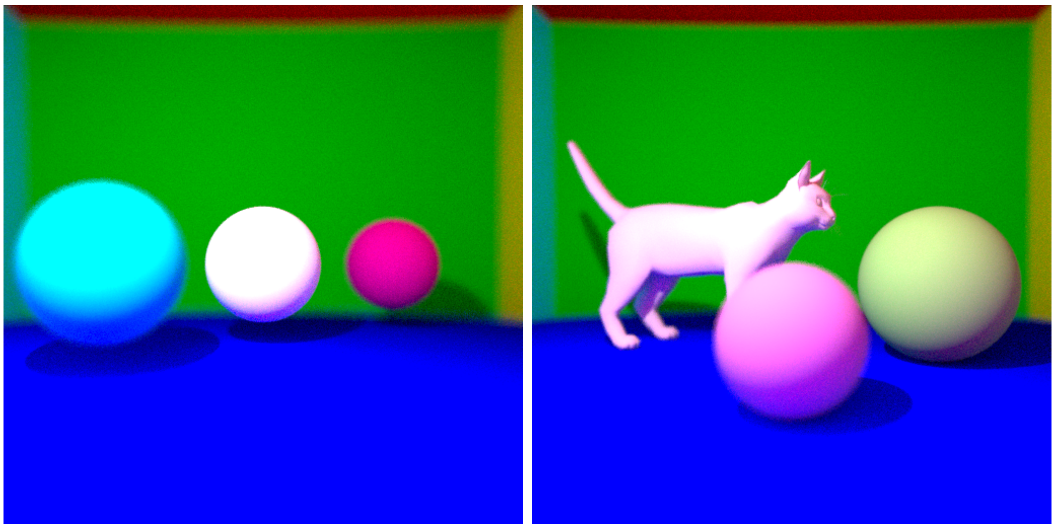
Figure 7: **Depth of Field.** We have two examples of depth of field figures. The left has the central sphere in focus while the other two are blurred. The right image has the cat partially blurred due its depth and the right sphere in focus, while the central is blurred. Both have antialiasing and indirect lighting enabled, and the left one has Phong interpolation enabled for the cat. The left image took 18.8s to render and the right one took 50.3s.