CSE306 Assignment 1

Alice Aubert

May 2nd 2021

1 Introduction

The goal of this project was to create a ray-tracer in C++ without the use of libraries. The first part of this assignment consisted in understanding the basic principles of a ray-tracer by creating a scene of spheres. The second part of the assignment consisted in implementing a mesh renderer.

All of the code was run on a MacBook Pro with a 3.1 GHz Dual-Core Intel Core i5 processor, and 8 GB 2133 MHz LPDDR3 memory.

2 Code Structure

The code is structure is as follows. First we implement a vector class in `cector.cpp` with all the function needed for vector calculus. It also includes all external exports. The relevant object for scene creation are defined in the `objects.cpp` file, such as sphere, camera and light. The `helpers.cpp` contains all of the scene calculation functions, such as scene building, intersection and mirror calculation. Finally, the `main.cpp` initiates the computation, and the `image.cpp` file creates the image file.

3 Spheres

We begin by creating a scene of spheres. We first began with a simple white sphere (no mirror or transparency) in the middle of our scene. When implementing gamma correction, we tested out a three different values of gamma: 1, 2.2 (the usual value) and 4. In all three cases the run time is instantaneous.

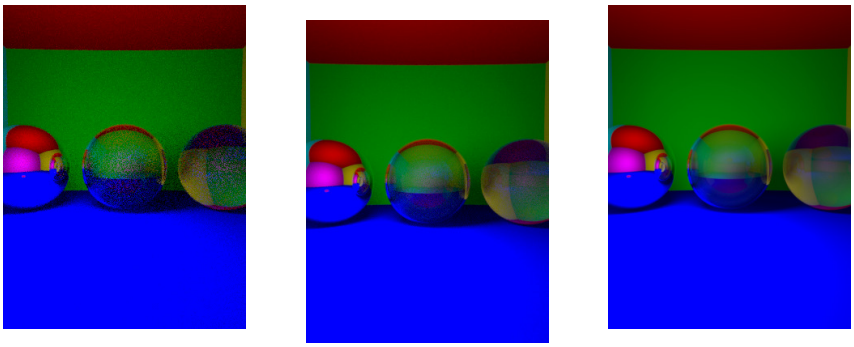
Having determined that a value of `gamma = 2.2` yields the best results, we will keep this value of all future images. We also determined that a light intensity of 100 000 worked best.

We then implemented some different sphere materials to add to our scene, notably mirrored spheres, hollow spheres and translucent spheres. The mirror spheres simply reflect the rays which hit them at the same angle relative to the normal of its surface. The rays of the transparent sphere are computed using the Snell-Descartes Law, assuming the air has a refraction index of 1 and the

(a) gamma=1 (b) gamma=2.2 (c) gamma=4

sphere has a refraction of 1.5 (the same as glass). The hollow sphere is made up of two concentric transparent spheres, with the inner sphere's refraction indexes switched to create a lens effect.

Since refractions has some inherent randomness, we use the Fresnel law to avoid computing the path of every ray. We incorporate the idea of running our simulations multiple times to get a smoother image. The results of increasing the number of rays per run is shown below.



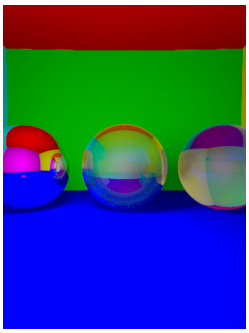
(a) 10 rays, instantaneous computation (b) 100 rays, 2s computation (c) 3000 rays, 5min computation

We can see that the more rays we add, the smoother the image is and the more the grain of the image diminishes.

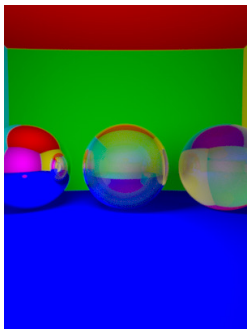
Lastly we move onto antialiasing. We use the Box-Muller method which smooths out the edges of the shapes. We tested values of 0, 0.3 and 0.5 for the standard deviation.

We chose to go with a standard deviation of 0.3 as it seemed to be a good middle ground between the graininess of the standard method and the "out of focus" feel of a value of 0.5.

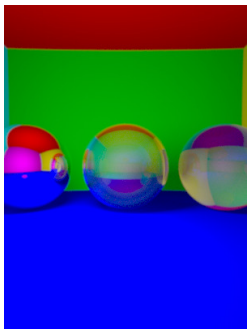
Lastly we created a slightly artistic point of image, by playing with the composition of our scene. We used three spheres, one mirror, one normal white sphere and one transparent sphere. We used 3000 rays for a more polished look



(a) std=0



(b) std=0.3



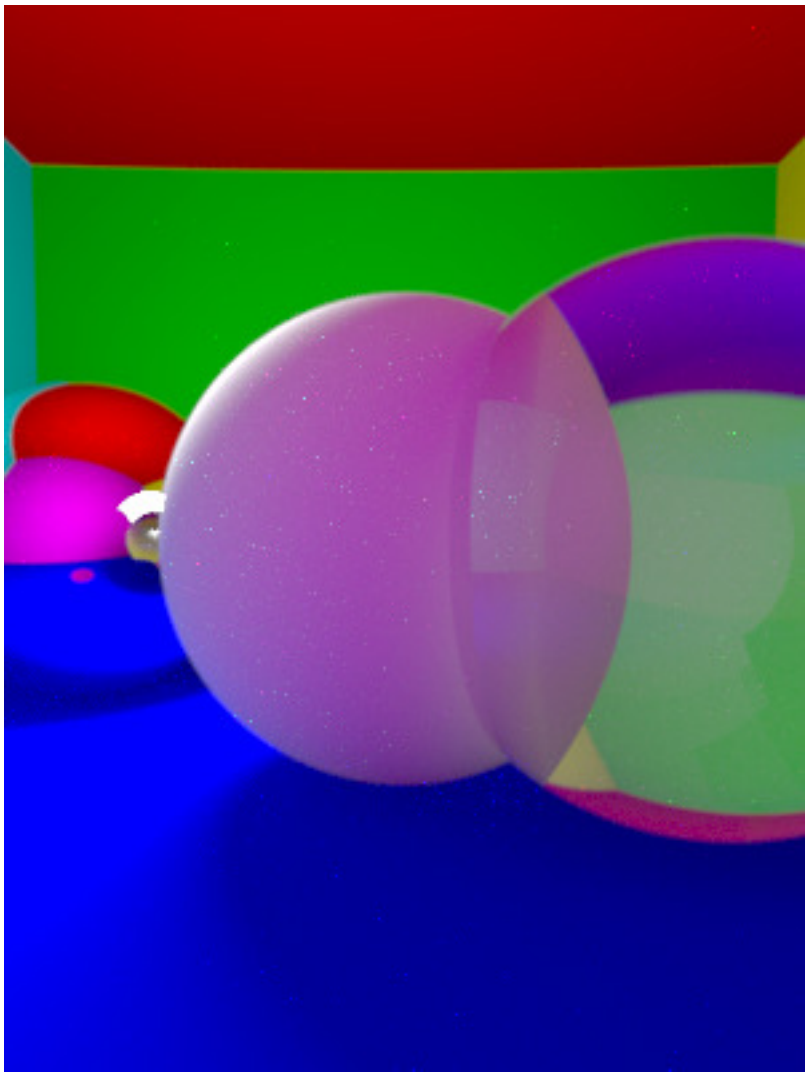
(c) std=0.5

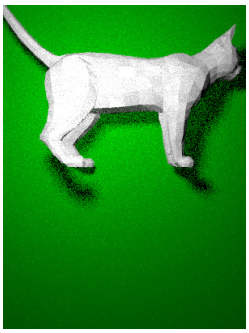
which brought our computation time up to 10 minutes

4 The Cat

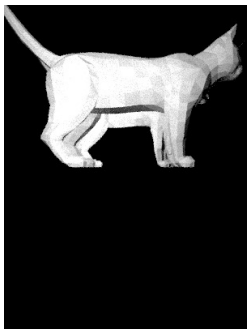
Although useful as a training exercise, spheres are do not allow us to create more complex objects. For this, we must utilise meshes. Due to the higher complexity of the problem, the programs take longer to run. In our case, we implemented only 10 rays per pixel.

We did not implement BVH optimization.

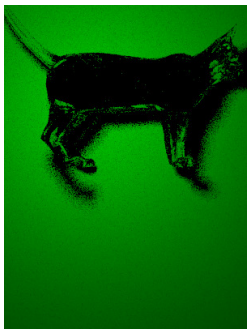




(a) 5 rays



(b) Mirror background for twice the cat



(c) Mirror cat