CSE306 Computer Graphics

Report on Lab 1 - Lab 4

Introduction

This project implements a Ray Tracer in C++. This report will outline my work throughout the first 4 lab sessions. The code for the labs is available in my git-hub (https://github.com/Leal-Koeksal/CSE306—Computer-Graphics). Each Lab has a designated folder containing results as well as code up to that point. The complete code for all four labs is found in the file *main.cpp*. I would also like to add a note on the rendering time. Towards the end of the project I had to change laptops. Since then, the rendering time has slowed. I have tested my code on other devices, where it ran a bit faster.

Lab 1 - Surfaces, Light & Shadows

The figures below depict three basic spheres. There is one transparent sphere (with refractive index 1.5), one mirror, and one solid. The figure on the left took 3.6 seconds to render with 1 *ray-per-pixel (rpp)* of 1 and the figure on the right 278.108 seconds with a rpp of 100. A ray-depth of 20 was used for both figures. The implementation of refraction includes Fresnel.

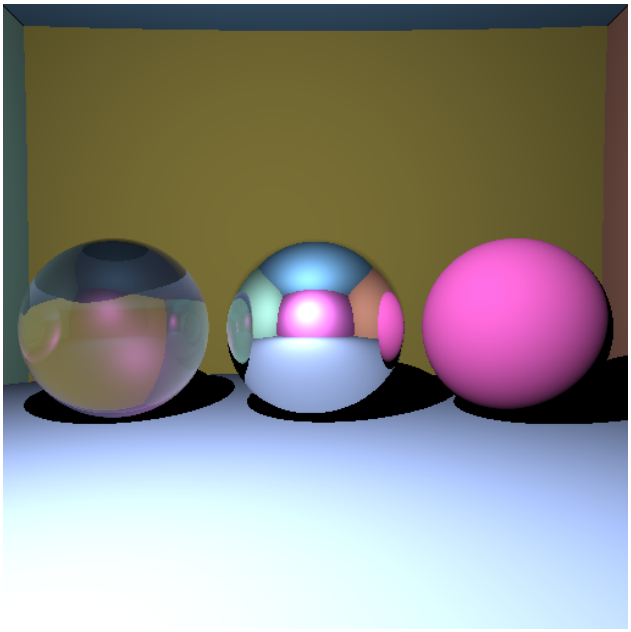

Figure 1: rpp of 1, depth of 20, time: 3.6s

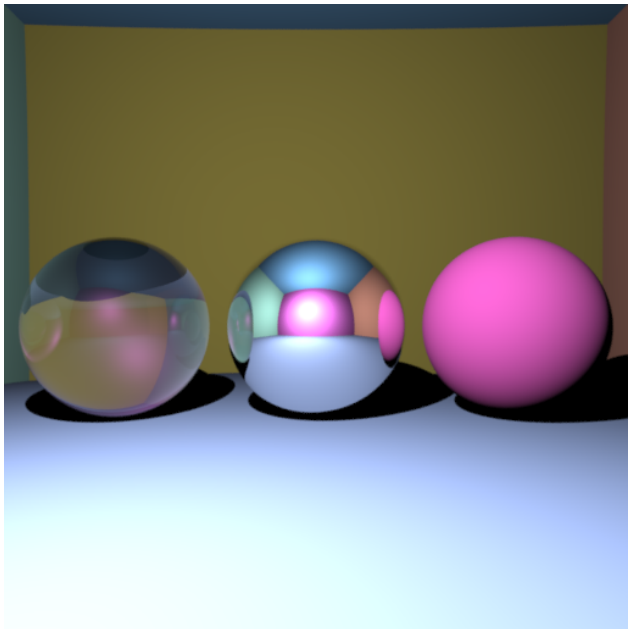

Figure 2: rpp of 100, depth of 20, time: 278.108s

Lab 2 - Indirect Lighting & Antialiasing

In this Lab I implemented indirect lighting, parallelisation, and antialiasing. All figures in this section use a ray depth of 10 and a rpp of 30.

The figures below shows the same image as above but with antialiasing. The standard deviation (stdev) values are detailed in the captions of the figures. These images also include indirect lighting.

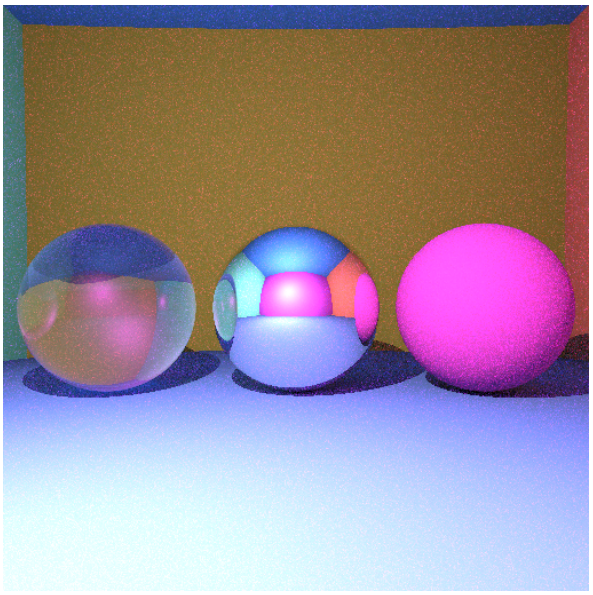

Figure 3: stdev: 0.02, time: 28.621s

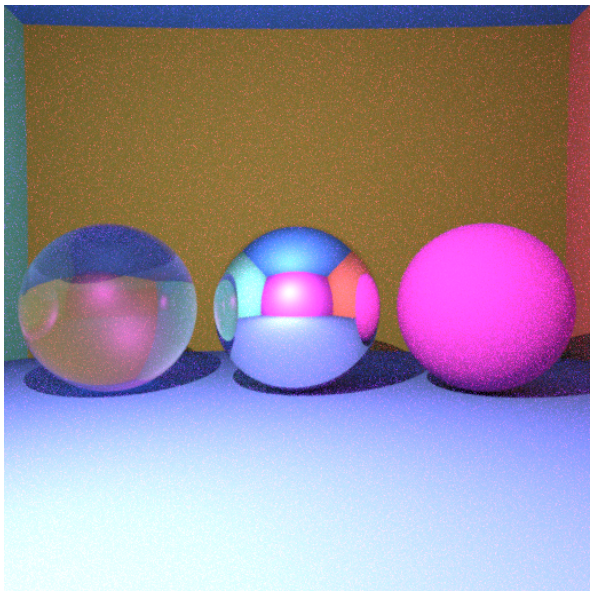

Figure 4: stdev: 0.5, time: 28.092s

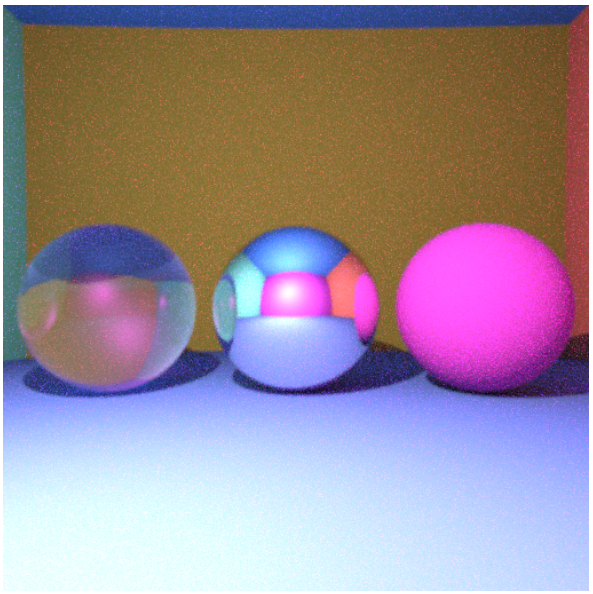

Figure 5: stdev: 1.0, time: 45.446s

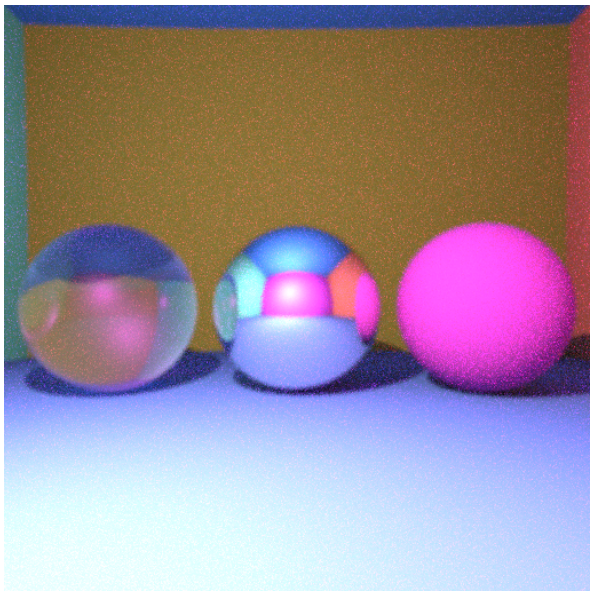

Figure 6: stdev: 1.5, time: 60.350s

I sped up the code by parallelisation `#pragma omp parallelfor`.
I also implemented depth of field. This was done by simulating a lens camera instead of a pinhole camera

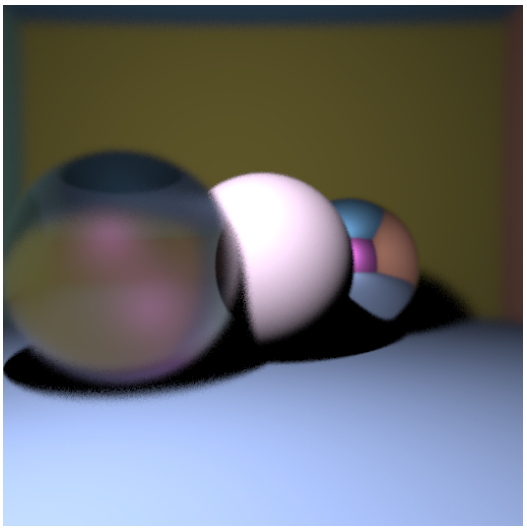

Figure 7: rpp: 50, ray depth: 5, time 8.492s.

Lab 3 - Meshes

In this lab I implemented the cat as well as the bounding box class in order to speed of the code. The images below was rendered with an rpp of 1 and ray depth of 5.

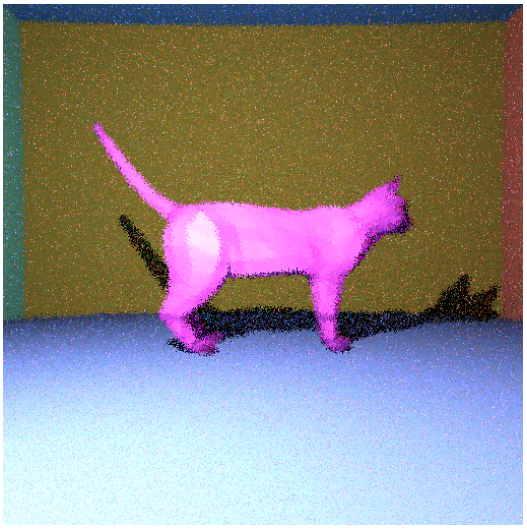

Figure 8: Time 39.824s

Lab 4 - BVH

Finally, to optimise the rendering process of the cat, I implemented Bounding Volume Hierarchy (BVH) which is used to speed up intersection tests between rays and objects (in this case triangle meshes). The BVH method was mainly interated into the function `TriangleMesh::intersect()`. The image below is the cat rendered using BVH and with an rpp of 1 and ray depth of 5.

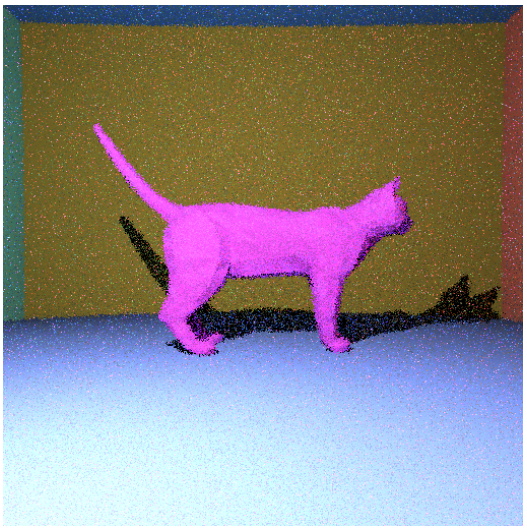

Figure 9: Time 1.082s.

In my ray tracer implementation, I encountered an issue where the program would occasionally crash or fail to save the final image using stbi_write_png. I believe the problem originates from the BVH traversal logic, which, although it works most of the time and produces correct images, sometimes leads to undefined behaviour. I was unable to determine the cause of this problem in time, but wanted to mention it for the sake of transparency.