CSIS335 Final Project Progress Report

We have successfully created a serial ray tracer. We now need to flesh it out a little bit more by adding functionality for other shapes, such as triangles, and actually parallelize it. It has been made with parallelization in mind so, despite being serial, it should be very easy to continue down this road. The only other steps that need to be taken as of now, are “housekeeping” such as moving the writing to a file into its own function.

The most recent addition is the processing of different materials so that some things can reflect light differently. Some objects, like rubber, don’t reflect as much light and don’t reflect it in the same way as ivory. This is mostly to make the actual ray tracer more of an actual ray tracer and to produce nicer output as it is not required for the parallelization aspect. This will, however, make it so that we can have lots of different things happening with different objects. This will make the processing more interesting and will then impact the parallelization aspect.

We are completely on schedule and have seen no bumps in the road, but there is obviously still a lot to be done. We hope that the addition of the ability to process triangles will allow us to create very large and complex shapes that will introduce parallelization related problems regarding dividing the work, as well as making large enough problems that the program is worth parallelizing in the first place.

Rough Outline of Final Write-Up

1. Overview

This project is a smaller and simplified version of a raytracer, a program that simulates light effects and is able to create reflections and refraction using many rays that are sent out of a camera point. This is used to generate images or even videos of complex situations, like a shiny metal object that is also reflected in a mirror.

2. Laws of Physics

This project requires the use of some Laws of Physics, but they also make some things much easier. Some, such as Snell's Law, allow us to write the code to do what the Law states and it will then work for the other instances of materials that we have.

3. Implementation

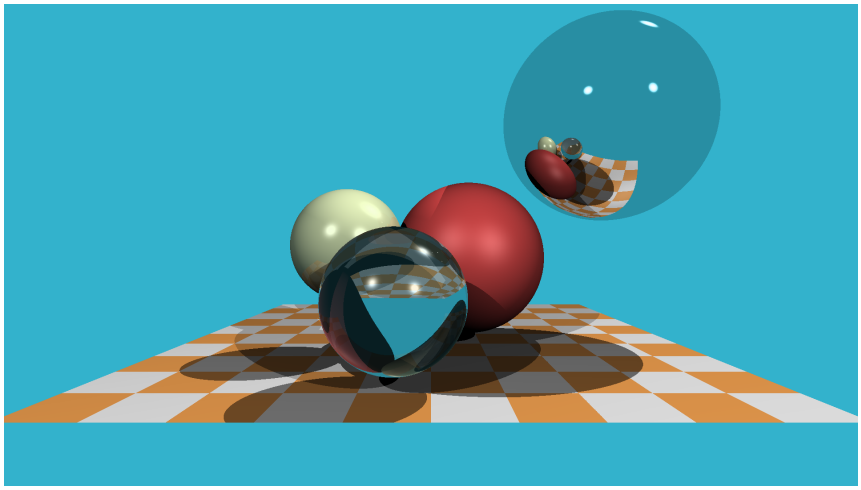
[This is where we will discuss the implementation of the code. As this still has much to go, this is left blank in the outline.]

4. Parallelization

[This section will be used to explain how the code was parallelized and hurdles that were encountered.

5. Output

Here are some of the output frames that we created. [This will be completed at the end to get the most up to date output and hopefully will include short animations.]



6. Conclusion

New Timeline:

Nov. 28 - Multithreaded single frame renderer.

Dec. 3 - OpenMPI parallel frame rendering.

Dec. 4 - Finalize parallel frame rendering for videos and discuss script creation.

Dec. 5 - Paper drafted and script to conglomerate frames into video.

Dec. 6 - Start Presentation

Dec. 9 - Finish Presentation

Dec. 13 - Final Submission

Current Bibliography:

tinyraytracer (Inspiration for the project)

https://github.com/ssloy/tinyraytracer/wiki/

Portable Pixmap Format (Current Output Format)

https://en.wikipedia.org/wiki/Netpbm

cglm (header-only graphics math lib for c)

https://cglm.readthedocs.io/en/latest/

Random distribution of rays among threads

https://ieeexplore.ieee.org/abstract/document/1580017

General raytracing optimizations

https://ieeexplore.ieee.org/abstract/document/537389

Distribution of data for complex scenes

https://ieeexplore.ieee.org/abstract/document/291533

KD-tree data divisions

https://arthurflor23.medium.com/ray-tracing-with-kd-tree-from-scratch-9b218823bc00

Creating Rays

https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-generating-camera-rays/generating-camera-rays

Ray Sphere intersection

http://www.lighthouse3d.com/tutorials/maths/ray-sphere-intersection/

triangle-ray intersection

https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/ray-triangle-intersection-geometric-solution

Snell's Law

https://en.wikipedia.org/wiki/Snell%27s_law#Vector_form