Nickolas Hessler (014780129)

Thongsavik Sirivong (013359424)

Prof. Markus Eger

CS 4450-01

13 December, 2022

Raytracer Project: Final Submission Report

For our project, we implemented basic ray tracing, various primitives (spheres, planes, triangles, and cylinders), constructive solid geometry (union, difference, intersection), camera placement/rotation, Phong lighting, reflection, and textured surfaces. And here is a list of Milestones and team member contributions:

Milestone 1: Basic Raytracing and primitives

- Ray intersections: Thongsavik
- Spheres: Thongsavik
- Planes: Nickolas
- Triangles: Thongsavik
- Cylinders: Thongsavik

Milestone 2: Constructive Solid Geometry (CSG) and Camera Placement/Rotation

- (True) Union: Nickolas
- Intersection: Nickolas
- Difference: Thongsavik
- Viewing direction: Thongsavik
- Field of view: Thongsavik

<u>Milestone 3</u>: Phong Lighting and Reflection

- The Phong Lighting Model: Thongsavik
- Shadow Rays: Thongsavik
- Reflection: Nickolas

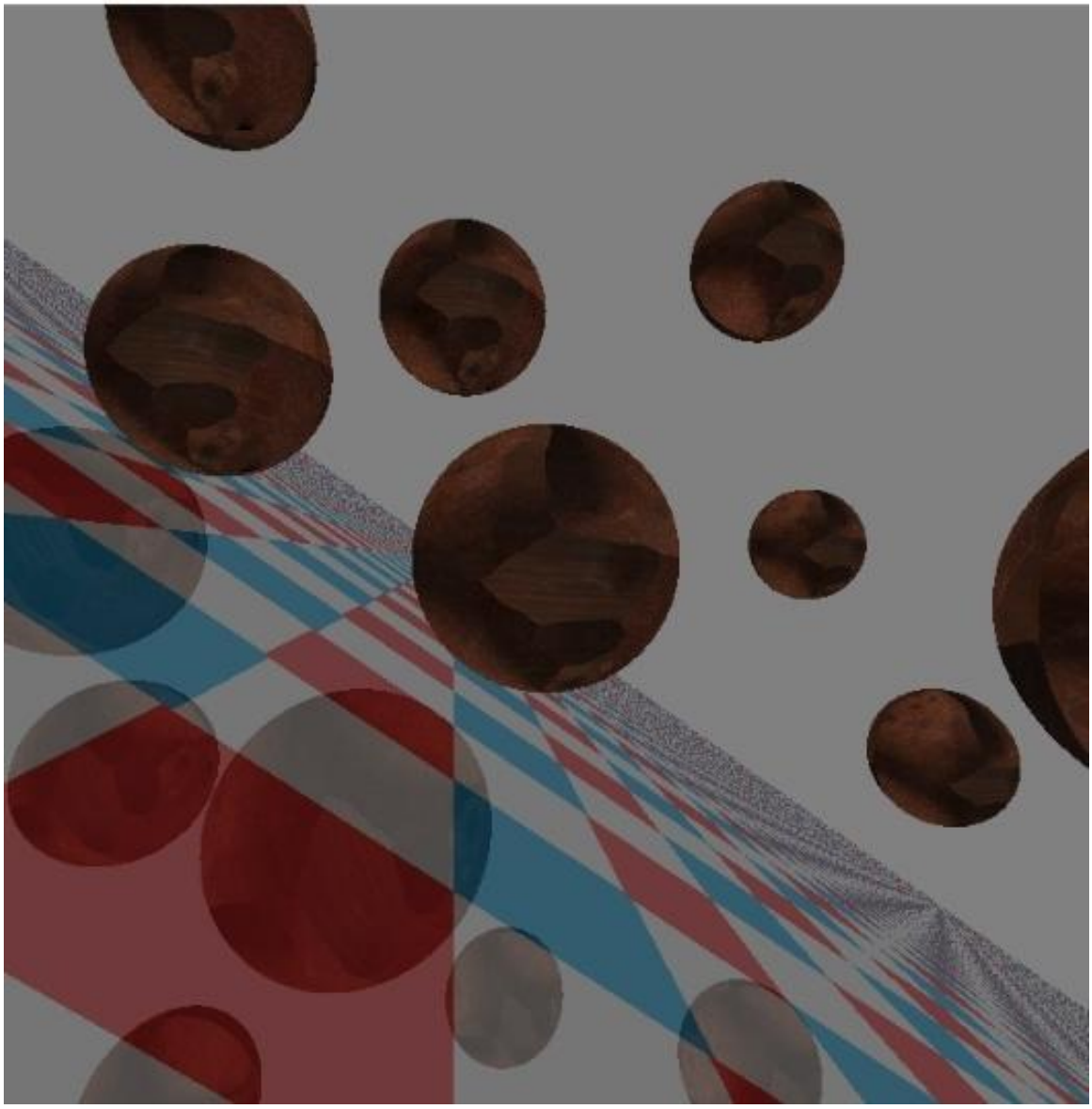
<u>Milestone 4:</u> Textures

- UV-Coordinates on Spheres: Nickolas
- UV-Coordinates on Planes: Nickolas
- UV-Coordinates on Triangles: Thongsavik
- UV-Coordinates Interpolation on Triangles: Thongsavik

We ran into several difficulties along the way, specifically with the implementation of the sphere primitives, the union function, reflections, and textures for spheres and planes. We have taken steps to address each of these bugs along the way. Initially, it was quite difficult to grasp the framework and understand the inner-workings of the api. The json files were quite helpful as well, to see what properties are included in the test cases, which lead to understanding of how some of the methods work. The sphere primitive was a simple one and a good starting point. Though, the corner cases can be tricky, such as for when camera origin is inside the sphere, where the first ray hit will always be an exit. Tackling this issue, we used the dot product of impact point normal and ray direction vector, where if the dot product is greater than 0, then the hit is an exit. As for CSG, it took us a while to figure out the way to check for if the ray origin is inside the object. Thanks to the professor's suggestion, we created a method inside the SceneObject interface that will check based on the primitive type. For special cases like the plane, it is possible a ray may never hit the plane, but the ray origin is inside/behind the plane.

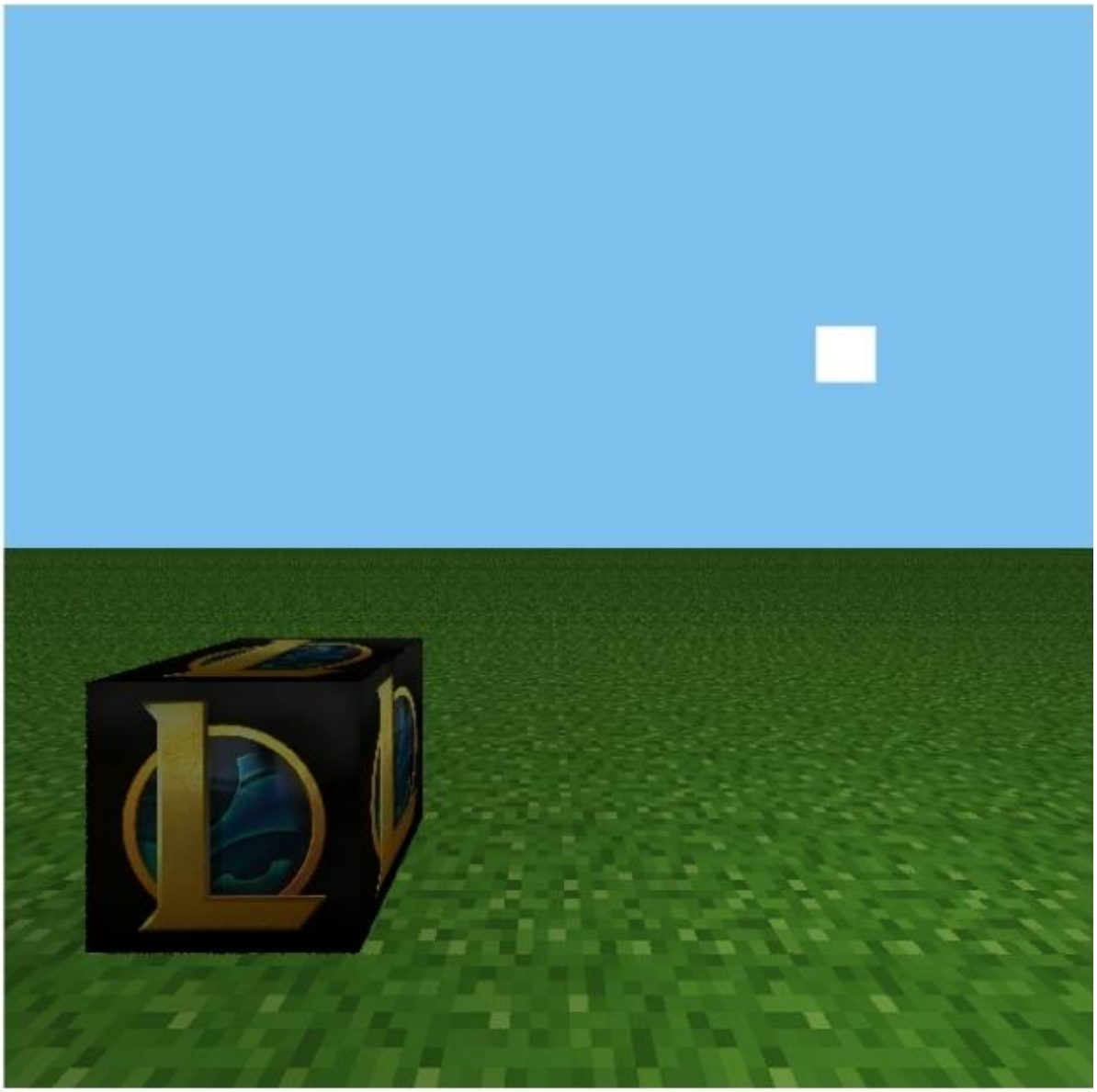
We tackled this issue by overriding the interface method to use the ray origin and shoot another ray in the direction of the plane's normal vector. This will guarantee the ray to intersect the plane if the origin is behind the plane or miss it completely if it is in front. Another speed bump of this project is the reflection. It required the understanding of multiple components like when to use “lerpColor” or scene background color, the differences between the scene reflection and reflectiveness of an object, or how the reflection ray is composed. This implementation took us some time to finally get the correct result with trial and error. Additionally, we attempted to implement an object's transformation, but due to time constraints, we were not able to complete the implementation in time. As for the rest of the implementations, we found they were just matters of understanding the steps, whether the notes from the lecturer slides or the online materials, to implement the necessary functions. Overall, we really enjoyed working on the project. Without the framework, implementing raytracing would have been a miserable experience, so thank you to professor’s Edger framework for simplifying the implementation and creating an enjoyable experience.

For our images, all of the images were created and designed by Nickolas Hessler. As they are shown below:



Original:

Revised: with fixed in spheres primitive and changing the lights location in json file




You've just been
Rickrolled!