Project Checkpoint

Progress Update

Currently we have successfully implemented a method to parse an `.obj` file and store the mesh of triangles into the GPU memory. After the triangles are copied to the device memory, we are able to correctly render the object using ray casting to determine the correct triangle intersection and interpolate the vertex normal to compute the normal map of each pixel being rendered. Furthermore, we have implemented a basic "lambertian" shading model to compute the color of the intersected triangle point based on the light source position.

This matches the timeline of our project plan which also gave us extra time to read up on literature and useful libraries that will be paramount for the next steps of our project.

Implementation Details

Implementation:

- Our lighting model assumes that we have a fixed-direction light ray at every point in space. This is highly unrealistic as there is no 'source' of light, however we have gotten a base lighting model to work. We implemented the Möller-Trombore triangle intersection algorithm, which is performed $O(N)$ times, where N is proportional to the number of triangles (which is proportional to the number of threads).

Performance Measurements:

Runtime Measurements:

- 960 triangle mesh runtime: 13.685 ms
- 4753 triangle mesh runtime: 66.592 ms
- 18192 triangle mesh runtime: 248.574 ms

NCU Measurements:

- SOL Compute (SM) Throughput: 96.41%
- Achieved Occupancy: 96.20%

https://github.com/rajsugavanam/CSE4059-final-project

Project Plan Revisions

Revisions: Clearer Objectives have been set for the next steps of our project:

- Cornell Box Scene
- Monte Carlo Path tracing and Lambertian Diffuse Reflection

We also plan to put a ground surface for the floor, and plan on making a triangle-based light source. We may use multiple objects to test multiple bounces of a ray, compared to our original decision in the project plan to use only one model. To this effect, we will need to make a new `class` for light sources. We also change the way our memory layout of our vertex or normal list works to allow for better coalescing.

Justification:

- The Cornell Box Scene will allow use to test the path tracing algorithm as it requires multiple bounces of a ray to be able to compute the color of a pixel after it hits a light source.
- This will require us to to use the cuRAND to uniformly sample the reflected ray at a bounce point which must point in the hemisphere of the triangle normal. The lambertian diffuse reflection can be optimized by a cosine weighted sampling so we don't have to reject a uniform ray sample that does not fit in the hemisphere.
- Using triangle meshes to represent both the light source and ground/walls of the scene will allow us to use the same intersection algorithm which will simplify the control flow of the program.
- Changing the memory layout to using structure of arrays (SoA) is necessary for parallel computing which should in theory speed up our program.