CG Bonus Report

Jeferson Morales Mariciano, Martin Lettry

November 30, 2023

Summary of Implementation:

We implemented two structures: `AABB` (Axis-Aligned Bounding Box) and `BVHNode` (Bounding Volume Hierarchy Node).

```
struct AABB
{
  glm::vec3 min;
  glm::vec3 max;
};

struct BVHNode
{
  AABB bounds;
  vector<Triangle> triangles;
  BVHNode *left;
  BVHNode *right;
};
```

When loading a `Mesh` object, the BVH implementation is optional.

If chosen, the BVH is constructed by recursively splitting triangles along the center of mass, alternating among x, y, and z axes. The process continues until each `BVHNode` contains a maximum of 20 triangles.

When computing the ray-Mesh intersection, the algorithm checks whether the BVH implementation is active. If not, it checks if the ray is within the Mesh's bounding box and computes intersections with individual triangles if applicable. With BVH, the algorithm recursively checks intersections with bounding boxes, descending until leaf nodes are reached. Finally, it computes intersections with triangles and returns the closest one.

We also created a small python file to help with plotting the data. It generated the below plot, which shows the render time as a function of the number of triangles in the mesh.

Implementation Issues:

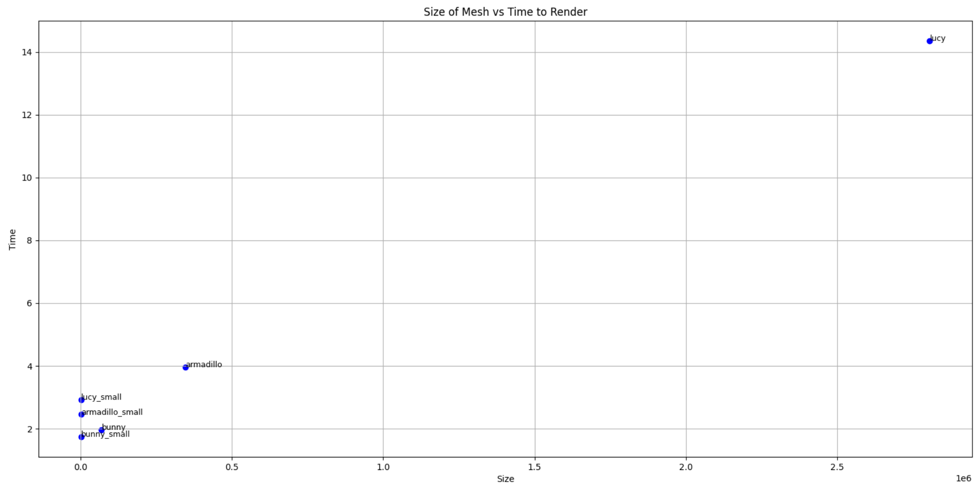
One issue we faced was regarding the intersection of the ray the both the right and left child nodes. This can happen if the triangle overlaps the splitting plane, in which case we need to recursively traverse both the left and right node.

Table 1: Performance Data

Mesh	**Size (in Triangles)**	**Time to render (s)**
armadillo_small	3112	2.45998
lucy_small	2804	2.92673
bunny_small	1392	1.73724
armadillo	345944	3.96717
lucy	2805572	14.3597
bunny	69451	1.95838

Benchmarking results:

Figure 1: Render Time by Number of Triangles.



As expected, we can observe a sub-linear relationship between the number of triangles and the render time.