Report - CG: Bonus Assignment 2

Jeferson Morales Mariciano, Martin Lettry

November 30, 2023

Prerequisites:

Before running the code, please download the meshes from the iCorsi link:
`https://www.icorsi.ch/pluginfile.php/1610541/mod_assign/introattachment/0/spatial_data_structures.zip?forcedownload=1`
and put it inside the *./code* folder and keep it with the name *meshes/* to have the paths like *./meshes/armadillo.obj* for *main.cpp* to run properly.
Such files were not included because to large for iCorsi submissions.

The *run.sh* contains the script use for the benchmarking, which correspond to the script written in the assignment for evaluation.
For benchmarking:

- the mesh load time is not counted, only the image rendering time is
- the resolution is set to 2048×1536
- the assignment command for benchmarking was used inside *run.sh*
- single thread implementation

Summary of Implementation:

We implemented two structures:

- `AABB` (Axis-Aligned Bounding Box) for completing Version A of the assignment.
- `BVHNode` (Bounding Volume Hierarchy Node) for completing Version B of the assignment.

For all benchmarks, including the final *result.ppm* both techniques have been implemented: in last notes on Table 3 show the total time to render the scene using both techniques with and without shading effects, specifically this effect due to time complexity spike.

```
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, z axes. The process continues until each `BVHNode` contains a maximum of n triangles, which for our implementation is chosen to be $n = 20$.

Logic steps:

- 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 bounding box and computes intersections with individual triangles as in assingnment bonus 1.
- With BVH, the algorithm recursively checks intersections with bounding boxes, descending until leaf nodes are reached using the BVH data structure.
- Finally, it computes intersections with triangles and returns the closest one.

We also created a small python file *report.py* 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 special corner case of intersection between the ray and 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 to find the correct intesected one.

Another issue we faced is due to time complexity of shadowing.
We have both stats of the rendered image with and without shadowing and we see that without such effect we get similar stats of what the assignment asked for: around ≈ 15 seconds.
However, when we add shadowing, the time complexity increases a lot, and we get around ≈ 50 seconds.

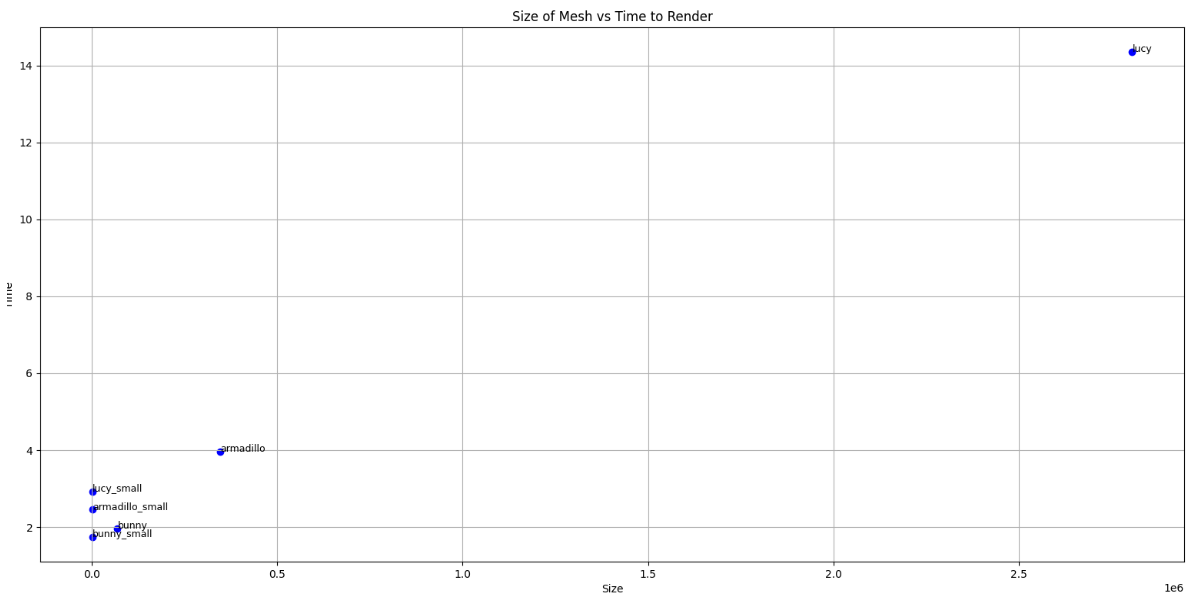
Benchmarking results:

Nevertheless, both Table 1, 2 show from raw data a sublinear relationship between the number of triangles and the render time if taking in consideration the same mesh rendered.

Table 1: Performance Data - no shadowing

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

Figure 1: Render Time by Number of Triangles - no shadowing

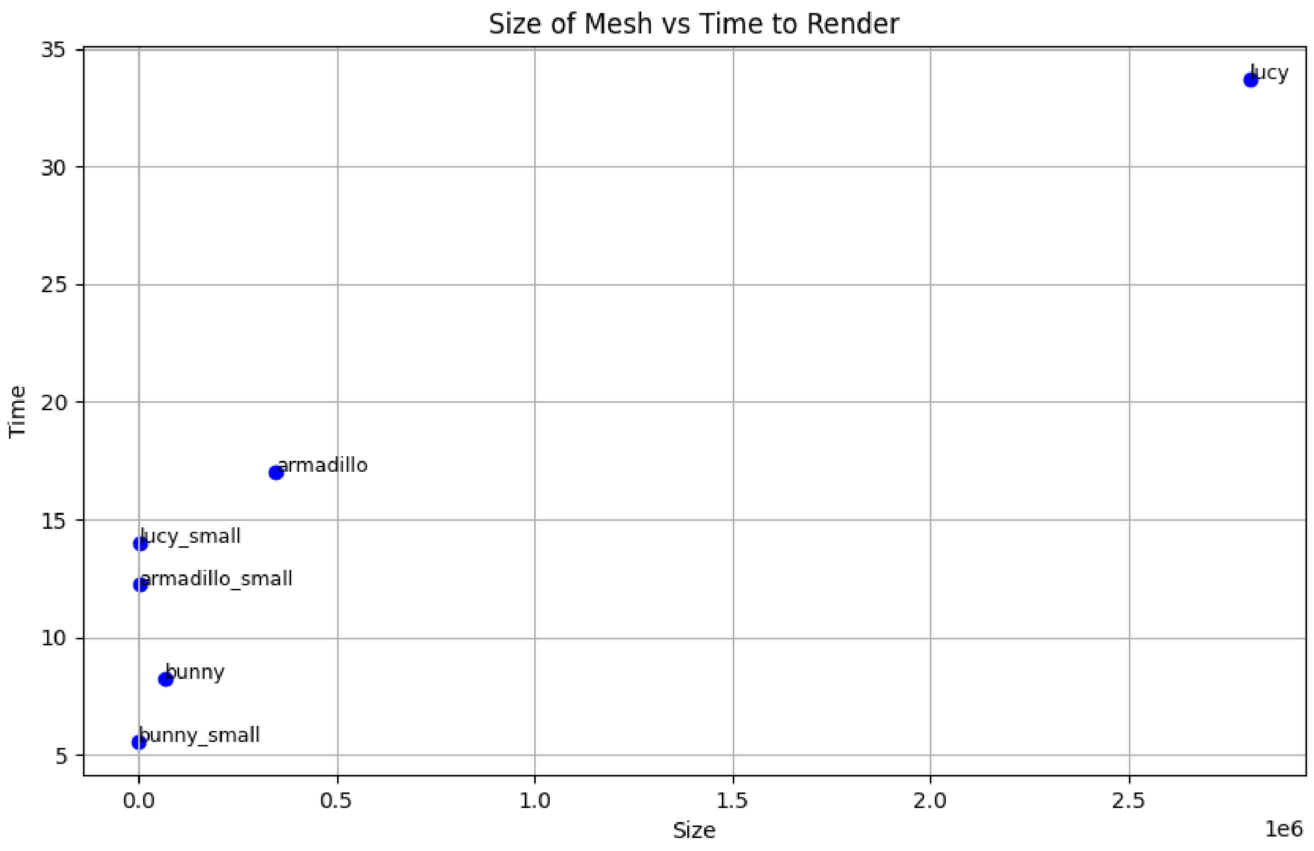


As expected, we can observe a sub-linear relationship between the number of triangles and the render time.
meaning that the slop connecting the same pair of mesh small-large is visibly smaller than 1.

Table 2: Performance Data - with shadowing

Mesh	**Size (in Triangles)**	**Time to render (s)**
armadillo_small	3112	12.2288
lucy_small	2804	13.9946
bunny_small	1392	5.54895
armadillo	345944	17.043
lucy	2805572	33.7117
bunny	69451	8.2359

Figure 2: Render Time by Number of Triangles - with shadowing



The total time to render the assignment scene with large meshes of *armadillo*, *lucy* and *bunny* in 2048×1536 resolution:

Table 3: Scene timing benchmarks

no shadowing	shadowing
≈ 14 seconds	≈ 50 seconds

You can change it and verify it easily in the scene definition function in *main.cpp*.

Further possible improvement analyzation

Implementing Sphere box or even better convex hulls instead of bounding box could greatly improve performance.
Also, implementing a more advanced version of BVH spatial data structure could dramatically improve performance.