AVDGR Assignment 2

Saiyang Zhang 4664302
Tim Goedhuys 6231926

1 Introduction

In this assignment we implemented:

- New path tracing with correct random ray generation.
- .obj file reader and data structure to store it.
- Intersection and occlusion.
- A basic BVH, and the BVH with SAH.
- QBVH with SAH.
- BVH and QBVH with bin heuristic.

All BVH handles both ray tracing and path tracing, supporting triangle and primitives from tmpl8rt_2022.

2 Implementation

New path tracer (Tim Goedhuys)

In the previous assignment we didn't apply the render equation properly in our path tracer. In PathTraceNew, You can see our new path tracer. This implementation follows the example from the lecture 4: Path Tracing. On line 207-210, you can see we added Russian roulette to make our path tracer more efficient. We clamped the probability between 0.1 and 0.9 to make sure this doesn't break the code. On line line 227-232, you can see a proper use of a BRDF, which was not the case in our last path tracer. We also already implemented a PDF, But this is still using the standard calculation so it doesn't do anything new at the moment. We started on an implementation of cosine weighted distribution, but didn't finish it because the main focus was really on the implementation of the BVH. The path tracer can be tested by commenting line 313-390 in the Tick function of renderer.cpp. And uncommenting line 391-412, also in the Tick funciton of renderer.cpp.

Adding triangle mesh to the scene (Saiyang Zhang)

We updated triangle class and added triangle mesh class to read from .obj file and add the triangles to the scene.

Basic BVH (Saiyang Zhang)

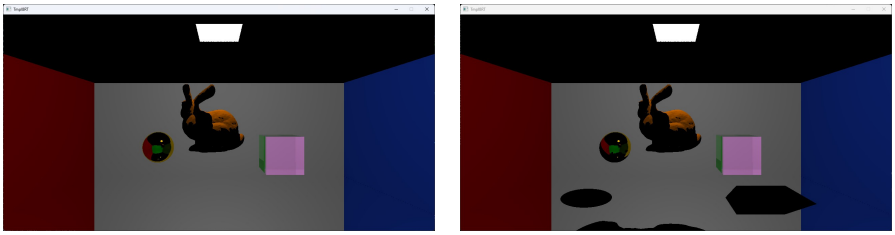
In BuildBVH, the partition method is simple. First compute the bounding box of the current node, then sort the shapes in the node along the longest axis of its bounding box, the split position is the median of the sorted shapes.

After selecting the axis and split position, we sort the shapes and split. Theoretically, moving the shapes left to the split position to the left side of the shape vector is of time complexity $O(n)$, and it should be faster than full sort which is of time complexity $O(n \log n)$. But in practice the standard sort is about 10 times more efficient than swap the shapes one by one, so we stick to the standard library.

Intersection and occlusion (Saiyang Zhang)

In IntersectBVH we implemented the traversal of rays throught the BVH structure.

In IsOccludedBVH we separate the occlusion from intersection. The main difference is that the function will early out if the ray is occluded in any child of the current node, but as our mirror and glass material are implementing dependant on knowing the exact shape that occlude the ray, using this occlusion results in some problem in the rendering result as in figure 1.



(a) Intersection: no shadow on mirror floor (b) Occlusion: shadow on mirror floor

Figure 1: Problem if we ignore the details of occluding shape.

BVH with SAH (Saiyang Zhang)

In BuildBVH_SAH, we take advantage of this heuristic: the larger the shape is, the more likely a ray hits on it. Directly computing the surface area of shapes' projection on camera is expensive, so we define "larger" by comparing the surface area of the bounding box that contain the shapes. Consider the

number of shapes contained, we define the cost of splitting at different position as:

$$C_{SAH} = N_{left}A_{left} + N_{right}A_{right}$$

where N is the number of shapes contained on this side and A is the surface area of bounding box on this side. Our goal is to minimize the cost.

In FindSplit we compare and find the best split position. Along each axis, we first sort the shapes by their centroids, then start with only the leftest shape on the left side, move the shapes one by one from the right side to the left side, compute the cost of each partition, and choose the axis and split position of the least cost.

QBVH (Saiyang Zhang)

Compared to the BVH with 2 children, when implementing QBVH we need to find 3 split positions instead of 1. Still we use SAH heuristic to partition the shapes. Theoretically the height of QBVH structure should be $\log_4 n$, which is half of $\log_2 n$ of BVH with 2 children, but the efficiency is not doubled as the expense of building the structure increases.

Adding bin heuristic (Saiyang Zhang, Tim Goedhuys)

To reduce the expense on building BVH structure, we can first fit the centroids of shapes into evenly distributed bins, and compute the bounding box of binned shapes instead of computing the shapes one by one. The BVH structure is surely not optimal by this method, but the expense of building the structure is reduced. In FindBinSplit we implemented such binning and splitting method, and in BVH_BIN and QBVH_BIN we applied this splitting method in both BVH and QBVH, but the improvement is not obvious.

3 Analysis on different BVHs (Saiyang Zhang)

We added the stanford bunny, 1 shpere, 1 cube and 7 planes (including the light source) in the scene, and use the Whitted-style ray tracer on the same machine to test the performance using different BVHs. The number of shapes in total is 4977, all BVHs have at most 2 shapes in a leaf node, the number of bins in binned BVHs is 16.

	Basic BVH	BVH_SAH	QBVH	BVH_BIN	QBVH_BIN
Number of nodes	5857	5937	5483	5937	5484
Average traversal depth	48.88	7.56	3.62	7.84	3.97
Average construction time	9.69ms	67.27ms	71.34ms	48.53ms	56.83ms
Average traversal time	110.85ms	33.77ms	25.97ms	33.95ms	27.39ms
Average frames per second	9.2	28.7	39.2	25.9	36.3

Table 1: Testing results

By changing the global variables each time we execute the BVH building and traversal function, we can compute the average time for BVH construction and depth for BVH traversal. What we can see from the table 1 is that the traversal depth time positive correlation to traversal depth. Comparing the performance difference between unbinned and binned BVHs, we can see that in our implementation, the traversal time has more impact on the final rendering result (frames per second) than construction time. Although binned BVHs take less time to be built, it is still insufficient to cancel the disadvantage of increased traversal depth.

4 Further development

- We read the instructions on OpenCL but had no time to implement it.
- The animation is incomplete in cube and triangle mesh.
- The project needs to support skydome.

References

https://jacco.ompf2.com/2022/04/13/how-to-build-a-bvh-part-1-basics/
https://jacco.ompf2.com/2022/04/18/how-to-build-a-bvh-part-2-faster-rays/
https://jacco.ompf2.com/2022/04/21/how-to-build-a-bvh-part-3-quick-builds/
https://jacco.ompf2.com/2022/05/27/how-to-build-a-bvh-part-8-whitted-style/