Vladimir Maksimovski, Yanghui Wu

CSC 292

Yuhao Zhu

Dec. 18, 2021

# Final Project – Ray Tracing Acceleration

## Introduction

We are trying to implement multiple methods to accelerate the Ray Tracing process for the final project. We use the base code introduced by Professor Peter Shirley in his book "Ray Tracing in One Weekend", and we are constructing the acceleration on the top of the base code. Our original goals are to achieve the acceleration by the following methods: parallelizing the code to have it running in concurrency if using CPU, implementing bounding volume hierarchy, using GPU power to accelerate, and training an AI model to facilitate the speed up Process.

We went to try parallelizing the given CPP code for Ray Tracing. However, unexpectedly, there was a high dependency on most rays, requiring us to put multiple locks on threads for several rays. The number of locks would quickly be challenging to keep on track. The image started to show tearing effects without preventing data dependency, even though the speed is much faster than before. The path seemed unreliable, so we changed gear to study the code in-depth and implement the bounding volume hierarchy.

First, we tried to set up the bounding volume as a sphere since the sphere class was already created and all objects were spheres in the code for the first week. Using sphere meant that we could reuse the intersection algorithm when a ray hit the sphere. However, there were many drawbacks to using a sphere as our bounding volume shape; some bounding spheres were larger than the canvas we had. Even though the ray was inside the sphere, the existing intersection algorithm still counted it as a miss and thus missed all the other spheres in the bounding sphere. And the sphere bounding shape took a massive amount of volume and had big overlapping spacing with another bounding sphere. We then proceed to go with the traditional bounding shape as a rectangular. Since we changed the bounding volume, we also needed to develop another intersection algorithm between a ray and a box. We referenced the box intersection algorithm from Amy Williams with her study on An Efficient and Robust Ray–Box Intersection Algorithm.

For the mainframe of the bounding volume hierarchy, we followed the instructions from the note of lecture 20 on page 45. All sphere objects were placed into a BVH tree, where all non-leaf nodes would have two branches, and all leaf nodes would contain three-sphere objects. While traversing the BVH tree, we would return immediately if the non-lead node was missed. Then we try to find the minimum hit out of the three objects if we were traversing the leaf node. If not, we would run the traversing algorithm recursively by passing in the two branches.

We got a lot of hands-on experience on CPP language during the project that we usually would not have had since most of our college classes were focusing on other languages. We, unfortunately, did not have the chance to work on acceleration for GPU and AI. However, the project was so cool that we might keep it as a side project we could keep on working.

## Coding

We used the source code available at https://github.com/RayTracing/raytracing.github.io. We deleted the “theNextWeek” and “theRestofYourLife” directories, and did our changes on the "inAWeekend” folder. The unmodified version is stored in "src/original”, while the version of the code with Bounding Volume Hierarchies implemented is in "src/bvh".

The source code stores objects in a “hittable_list” structure. There are two different versions of such a node: binary internal and leaf nodes. Leaf nodes store the physical objects directly. Meanwhile, binary internal nodes only store references to two other “hittable_list” nodes. When constructing the hierarchy, we choose a dimension, choose a splitting line, and divide the objects into a left and right half.

If an object passes through the splitting line, we resolve ambiguities by only storing it in the left half. If more than three quarters of the objects are in one half, then we try another splitting axis-aligned splitting line. If none work, we convert the node to a leaf node.

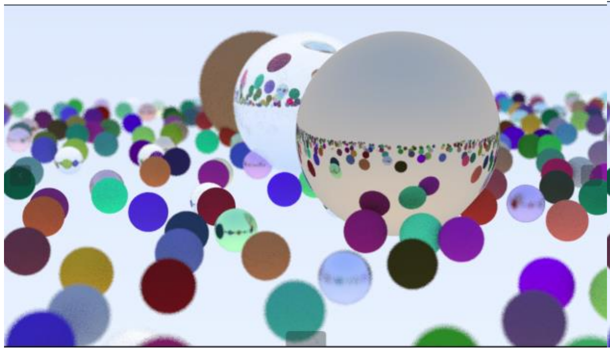
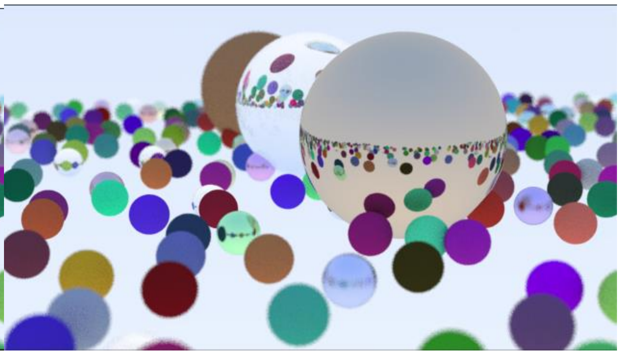
The splitting line is chosen by first choosing a random axis-aligned dimension, then selecting the median object along that dimension, and returning an axis-aligned splitting line passing through the center of the object.

## Methodology

Experiments are done by compiling both original and bvh  with optimizations enabled (by setting CMake to the “Release” setting). Tests were done on the cycle1.csug.rochester.edu machine. We ensured that no background processes were running, and that no one else was logged in to the machine to reduce measurement noise. The same scene was rendered with both executables, and time measurements were made. The experiment was repeated twice to reduce variance. The scene contains 500 uniformly distributed spheres.

## Results

The output images of both executables are shown below. The one on the left is for the unmodified version, while the one on the right is for the modified version. They are indistinguishable, which means that our modifications preserve correctness.

The average runtime for the non-modified code is 41.87 seconds, while the average runtime for the modified code is 11.39 seconds. This indicates a 3.6x change, which makes the modification worthwhile. The authors believe that for larger scene sizes, the speedup will be even more significant, as the asymptotic complexity of the bounding volume-hierarchy approach is better. The BVH version has only a logarithmic expected search time, while the unmodified version has a linear search time for ray object intersection.

## How to replicate results

Run:

- mkdir build

- cd build
- cmake .. (Then, press “c”, then modify CMAKE_BUILD_TYPE to “Release”, then press “c”, then “g”)
- make
- time ./render_bvh > 2.ppm; time ./render_bvh > 2.ppm; time ./render_original > 1.ppm; time ./render_original > 1.ppm

You can observe measurement times by comparing the “user” times for each execution. You can ensure correctness by viewing and comparing 1.ppm and 2.ppm.