Parallel BVH Generation via GPU - Jai Madisetty, Denise Yang

Github: https://github.com/Denise-Yang/BVHGPU

SUMMARY:

We are going to implement bounding volume hierarchy creation via a GPU. We will perform a detailed analysis compared to the multi-core CPU platform from last semester.

BACKGROUND:

For our project we will work on accelerating the sequential bounding volume hierarchy implementation in our Scotty3D package on Nvidia's GPUs. The algorithm is used to improve the computational cost of calculating intersections during ray tracing by eliminating the number of primitives that we need to check if there's an intersection by about a factor of two. It is similar to a binary search algorithm in that if the ray intersects the box then we can check if the ray intersects any of the two children of the box else we can discard all of the primitives bound within the box. We will keep navigating through the BVH tree until we hit a lead which represents a list of scene primitives that we can then check for a ray intersection. Creating a bounding volume hierarchy however can often be an expensive process for arbitrarily complex meshes. The serial algorithm involves iteratively dividing the set of scene primitives into bounding boxes, where the best partition is dependent on a cost function known as the surface area heuristic (SAH). This cost function compares the surface area of the primitives within the box to the surface area of the box that bounds those primitives.

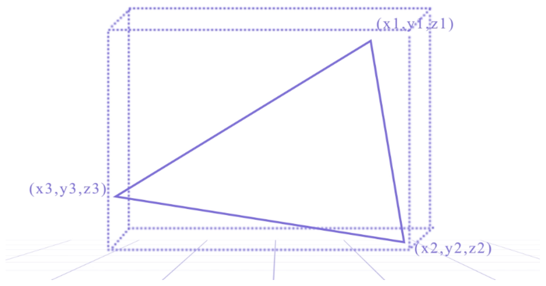

Figure 1. Sample bounding box around a single primitive

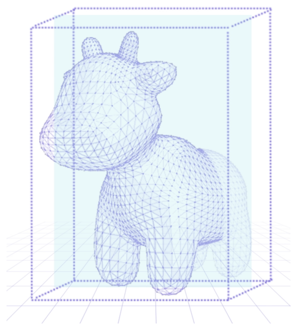

Figure 2. Potential partition of bounding box (blue) to create child nodes

The process of creating a BVH can be split into many sub problems. Viewing each node of the BVH tree as a sub problem already puts into view the potential for parallelism as well as identifying the best partition to split the bounding box. Since sequential creation of BVHs can be expensive, it involves at least O(nlogn) since sorting can be reduced to the problem of BVH creation (at least when the BVH can have arbitrary depth and leaf nodes contain only one primitive). Therefore a lot of room is left for improvement when parallelizing this process.

THE CHALLENGE:

The process to generate a BVH can be separated into sub-problems, where each sub-problem is represented by a node in the BVH tree. Due to the recursive nature of this process, the parent node must be resolved before the current nodes can be computed, such that we can represent the task tree as a directed acyclic graph (DAG) where parent nodes point to their corresponding child nodes. These are the dependencies that we must take into account when parallelizing our program.

The memory access characteristics are rather scattered because we will be jumping over several nodes as we navigate through our BVH. Although when checking for the best partition we will need to access all of the primitives within a bounding box, which could provide some opportunity to optimize for spatial locality, we would need to ensure that the primitives are sorted by location when being processed in the pipeline. The sorting could add some additional overhead, which we would need to consider further.

Throughout this process we hope to better understand the thought process to parallelize computer graphics code since a lot of the data within computer graphics tends to undergo the same transformations; and, especially in the gaming industry, it is important to be able to quickly output high quality scenes for a good user experience. Furthermore this project will hopefully give us more insight as to why a lot of computer graphics related industries such as gaming and animation use GPUs.

RESOURCES:

We will start from the Scotty 3D package which is a 3D graphics package that contains many features; the ones we will be using specifically for this project are ray tracing for rendering, BVH tree visualizer for debugging, and a rendering window to display the rendered image. We will also need to use the GHC machines and Nvidia GPUs in order to parallelize our program.

In order to compare our solution with the multi-core CPU, we will analyze the results of the previous group that implemented a bounding volume hierarchy from Fall 21 via OpenMP.

GOALS AND DELIVERABLES:

Our Goals:

1. Implement a fully functional parallel BVH creation pipeline that can be used for ray-tracing in Scotty3D. We hope to achieve around 8x speedup since we know that speedup is achievable with multi-core as seen from the previous semester.
2. Provide speed-up graphs for BVH creation with on GPU platforms
3. Provide speed-up graphs relative to BVH quality - BVH quality can be measured by SAH or rendering speeds

Stretch Goals:

1. Investigate other methods to speedup BVH generation.

For our demo, we will show the speedup graphs as well as a video comparing the BVH generation of the sequential version versus the parallelized version. From this project we hope to gain more insight into the tradeoffs between using GPU's vs multi core especially from a computer graphics perspective, since a lot of games and animation nowadays rely heavily on parallelism in order to speedup rendering frames. Some questions we plan on answering are:

- What is the behavior of our speedup graphs and why?
- How do our graphs compare to the speedup graphs of BVH on multi-core?

- Under what circumstances would it be better to use multi-core vs GPU?

PLATFORM CHOICE:

We will be using C++, because the Scotty 3D package like many other graphics software uses C++; furthermore CUDA is also compatible with C/C++. We will also be accessing the GHC machines since they are already equipped with GPUs.

SCHEDULE:

Week 1	Revisit Scotty3D code and familiarize ourselves with how BVH creation works and plan out how our parallelization strategy will work
Week 2	Implement parallelism for basic BVH creation for arbitrarily large scenes
Week 3	Debug parallel implementation for BVH creation.
Week 4	Project checkpoint: Investigate other methods for BVH creation speed-up
Week 5	Add benchmarks for testing for later analysis. Prepare speed-up and BVH quality graphs. Answer analysis questions. If there's extra time, implement new methods.
Week 6	Prepare speed-up and BVH quality graphs. Answer analysis questions. Handin + Demo.