Parallel BVH Generation via GPU Milestone Report

Jai Madisetty, Denise Yang

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

Updated 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	Brainstorming different ways of parallelizing the program and researching recursive kernel launches.
Week 3	Implement parallelism for basic BVH creation for arbitrarily large scenes
Week 4.0	Denise: Acquire resources to run project on GHC machines Jai: Research more into CUDA dynamic parallelism documentation: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism
Week 4.5	Denise: Finish implementing parallelism within the nodes approach Jai: Finish implementing dynamic parallelism or switch to iterative implementation
Week 5.0	Denise: Debug parallelism within the nodes approach Jai: Debug parallelism with dynamic parallelism/iterative approach
Week 5.5	Denise: Increase the parallelism within the code for the parallelism within the nodes approach and debug Jai: If dynamic parallelism works, try implementing parallelism over iterative implementation and debug

Week 6.0	Denise: Add benchmarks for testing for parallelism within nodes approach, and sequential code. Prepare speed-up and BVH quality graphs. Answer analysis questions. Jai: Add benchmarks for testing for task based parallelism approach. Prepare speed-up and BVH quality graphs. Answer analysis questions.
Week 6.5	Denise: Answer analysis questions. Prepare poster. Jai: Answer analysis questions. Prepare poster. Demo

Work Completed so Far

- So far, we have a serial version of the BVH implementation working. We start with a vector of primitives which we want split into two partitions, and we want these partitions to yield the optimal cost with respect to the surface-area heuristic. Once we have found the most optimal split, we create children nodes corresponding to each side of the split, and recursively perform this process for both children. We continue this process until the number of primitives left in a node is left than the maximum leaf size. To find the most optimal partition of a node, we transform the node into 8 evenly spaced "buckets" along the x-axis, sort the primitives into these buckets, choose the best split amongst these buckets (split that yields the best surface-area heuristic cost), and then repeat this process for the y and z axes. We then choose the best split amongst the x, y, and z-axes.
- We have two main approaches for parallelizing:
    - Parallelizing within each of the BVH nodes. More specifically, we are working on assigning each CUDA thread a primitive and a bucket for the thread to check if the primitive lies within the bucket. For this method we will also parallelize finding the optimal partition amongst all the bucket partitions by assigning each thread a partition to create and calculate the surface area heuristic of.
        - For this method we plan to implement a basic parallel version that launches a sorting into buckets kernel and an optimizing partitions kernel for each of the 3 axes. This is theoretically implemented however we have been unable to test it due to an unaccounted lack of resources.
        - The next optimization we plan to make is to modify the code and our current data structures to allow us to process all three axes at once.

    - Running each recursive call of our BVH build function in parallel. More specifically, after finding the optimal split of the primitives, there are 2 recursive calls which solve the BVH within each of the child nodes. We want to be able to run these child nodes in parallel with one another.
        - We have done a significant amount of research on dealing with parallel recursion in CUDA, and it seems like it is a more difficult task than we thought. We found that GPU threads have relatively small stacks so we should avoid

recursion in device functions. Additionally, if we wanted to recursively call a kernel from within a kernel, we'd need to deal with dynamic parallelism which requires a compute capability 3.5 device. Since we are using the NVIDIA GeForce RTX 2080, this is not a problem.
    - Documentation for dynamic parallelism: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism
- If dynamic parallelism proves to be impractical to implement within the time frame of this project, we will attempt to make the current recursive code iterative so we can parallelize over different iterations of the code and will not have to deal with dynamic programming.

Goals and Deliverables

Plan to Achieve

- Implement a fully functional parallel BVH creation pipeline with two different parallel approaches that can be used for ray-tracing in Scotty3D.
- Provide speed-up graphs for BVH creation with on GPU platforms
- Provide speed-up graphs relative to BVH quality - BVH quality can be measured by SAH or rendering speeds

Hope to Achieve

- Investigate other methods to speedup BVH generation.

Updated Goals and Deliverables

We will be omitting our stretch goal of exploring other methods of optimizing BVH tree generation due to some setbacks we encountered with testing our project, as well as time spent researching our current methods that we did not account for when planning our original schedule. However we should still be on track to complete our planned goals and deliverables.

Poster Session Deliverables

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?

Concerns

Our biggest concern that we ran into when trying to test a basic implementation was that we cannot build our project on the GHC machines with NVIDIA GeForce RTX 2080 B GPUs since we need the following packages: pkg-config, libgtk-3-dev, and libsdl2-dev. However we do not have permissions to install them on the GHC machines. We have already contacted the CS help desk and hope to hear back from them soon or identify another way we can build our project.

Another concern with one of our approaches is that our original plan to parallelize the tasks across the nodes with recursive kernel calls is likely going to be slower than anticipated along with increased implementation complexity.

Bibliographies

"Cuda C++ Programming Guide." *NVIDIA Documentation Center*, https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#cuda-dynamic-parallelism

.