DH 2323 - Project Specification (for A grade)

Forrest Timour, Poorya Ghavamian

May 2, 2018

Background

Ray tracing is a rendering technique that is becoming increasingly more popular with the emergence of powerful hardware and algorithms that allow the very expensive photorealistic visual imageries to be produced in real time. A large amount of research and effort has been funneled down the path of real time raytracing and companies like Nvidia have recently announced products like *NVIDIA RTX^{TM} Technology* that show great promise. In this project, we will try to focus mainly on the software side of the problem, and therefore, will focus on an algorithmic solution to the problem of real time ray tracing. We will apply a KD-Tree, a type of spatial raytracing acceleration structure, to optimize intersection finding by subdividing the bounding box into smaller boxes recursively, effectively partitioning the large triangle set into much smaller sets that can be skipped during intersection finding by selective branch pruning for nodes whose bounding boxes do not intersect with a ray.

Problem

It should come as no surprise that tracing the path of a large number of rays in an image plane and simulating their interaction with the surrounding on its own comes with a great computational cost, and coupling that with real-time rendering takes a massive toll on a system. Our goal is to try to utilize a raytracing acceleration structure in the context of a massively parallelized GPU pipeline to improve the performance of a basic raytracer as much as possible, with the goal of being able to simulate real-time rendering. This will be done through literature review and referencing contemporary research done in the field. Furthermore, we will be able to run trials on our implementation and gather sufficient evidence to back our solution's effectiveness in the form of preprocessing and rendering time.

Implementation

We have created a list of optimizations that we will consider applying:

- Implement raytracing on a GPU for parallel performance
- Optimize the intersection finding algorithm
  - pre-calculation for faster ray-triangle intersections [1]
  - preprocessing with *kd*-trees to avoid unnecessary tests[2][3]
    - surface area heuristic for construction in $\mathcal{O}(n \log n)$
  - first-hit optimization for early termination of shadow ray checks
  - implement intersection tests for other graphical primitives

Evaluation

Because we plan to build up this project iteratively, we will compare the preprocessing and rendering times to previous instances of the project, for example both before and after utilizing KD-trees for preprocessing. This data will be gathered on multiple computing environments and compared in order to avoid bias from any particular system configuration, and then compared to the expected improvements found in the literature.

Risks & Priorities

Task	Priority	Risk
Loading in Test Model	1	None
Naïve GPU Raytracer	2	Highest
Construct KD-Tree	3	Low
Integrate KD-Tree into Raytracer	4	High
Optimize KD-Tree Construction	5	Moderate
Ray-Triangle precalculation optimimzation	Low	Low
More Graphical Primitives	Low	None

* "Low" priority tasks are things that we will implement only if we have ample extra time, and will not prioritize.

References

[1] Doug Baldwin and Michael Weber, *Fast Ray-Triangle Intersections by Coordinate Transformation*, Journal of Computer Graphics Techniques (JCGT), vol. 5, no. 3, 39-49, 2016 http://jcgt.org/published/0005/03/03/

[2] Ingo Wald and Vlastimil Havran RT06 Conference, Sept. 2006 http://dcgi.felk.cvut.cz/home/havran/ARTICLES/ingo06rtKdtree.pdf

[3] M. Vinkler, V. Havran, J. Bittner *Performance Comparison of Bounding Volume Hierarchies and Kd-trees for GPU Ray Tracing", in Computer Graphics Forum* DOI: 10.1111/cgf.12776, Volume 8, Issue 35, pages 68-79 https://onlinelibrary.wiley.com/doi/epdf/10.1111/cgf.12776