15418 Milestone Report

Parallel Packet Tracing in Scotty3D

Kenechukwu Echezona

Project Page

https://keche1234.github.io/15418-Final-Project-CPU-Ray-Tracing-via-Packet-Based-BVH-Traversal/

Summary

We attempted to update the **the 15-362 Scotty3D 3D Renderer application** to support Packet Ray Tracing via OpenMP and SPMD. The goal was to run tests on the Gates Machines and comparing the time it took to render provided test scenes with different materials. This would be based off of my implementation of the code, from when I took 15-362: Computer Graphics in Fall 2024.

Background

This project parallelizes **ray tracing** in Scotty3D, a graphics renderer from the CMU course 15-362: Computer Graphics. Pathtracing involves computing frame buffer colors by simulating light ray bounces through a scene. This is done via simulating "backwards scene" ray traversal, by shooting out light rays from the camera and having them bounce around the scene upwards of a fixed amount of time.

The main data structures include a **Bounding Volume Hierarchy (BVH),** which organizes scene geometry hierarchically into a tree-like structure using bounding boxes; and **Ray Packets,** which store groups of rays for coherent traversal. BVHs greatly speed up the expensive **ray-scene intersections queries** involved in rendering a scene, by reducing the amount of scene primitives a ray is checked against for intersection through a tree-like structure. In the best case, the amount of scene objects a ray is tested against goes from $O(n)$ to $O(\log n)$, which can mean cutting out millions of unnecessary ray-object intersection tests. Ray tracing is also **"embarassingly parallel,"** with work that's completely independent across rays. These are the initial motivations for using ray packets to do **Packet Tracing,** something I learned about in 15-473: Visual Computing Systems.

(Disclaimer: earlier in the semester I had asked about double counting this project for 15-473; just letting you know that I made my project for that course separate from this one! For 15-473, I instead worked on expanding the Middle Sort Tile Renderer assignment with Deferred Shading, using Rasterization instead of Raytracing).

Scotty3D already utilizes **SPMD (Single Program Multiple Data)** ray tracing, by having the programmer write code for a single ray trace and letting the hardware handle parallelization through multiple threads. Ray packeting allows the programmer to explicitly write their ray tracer to **trace n rays at a time** ("at a time" in this case meaning "per thread"). This lets us utilize **SIMD (Single Instruction Multiple Data)** execution by mapping each ray's operations onto one vector lane, and amortize the cost of BVH node retrieval.

Effective packet tracing relies on a concept known as **ray packet coherence,** a measure of how close the results of multiple ray-scene intersection queries will be in a single scene. Low ray coherence not only leads to **divergence in BVH traversal** (and therefore low utilization), but to poor cache behavior through our accesses (as any locality/data reuse as a result of being in the same BVH node is gone). Note that ray coherence is a property of both the rays *and* the scene: rays that have completely different points and directions can be highly coherent in a scene with fewer, closer triangles (Figure 1); and rays with similar origins and/or directions can be highly incoherent if there are many triangles in a scene (Figure 2).

Approach

There were two goals: (1) determine where to utilize OpenMP to parallelize ray traversal, and (2) to develop a SIMD-friendly Ray Packet structure, and reimplement some of the functions from the original assignment to support SIMD vectorized operations (particularly the lighting calculations, BVH traversal, and hit intersections for triangles, spheres, and bounding boxes). Notably, we determined Goal 1 before realizing that Scotty3D *already* parallelizes ray traversal via hardware threads.

As mentioned before, this project was an expansion of the Scotty3D codebase from 15-362: Computer Graphics. This software was built in C++, as well as ImGui, a library for implementing GUIs. We also used the OpenMP API, and would have used Intel Intrinsics to implement SIMD, with each ray mapped to a vector lane, and each hardware thread (GPU unit) handling a ray packet.

The primary challenge with this project was determining where to employ which forms of parallelization; as I'll elaborate on in the Results section, attempting to parallelize in the wrong place can actually slow the program down!

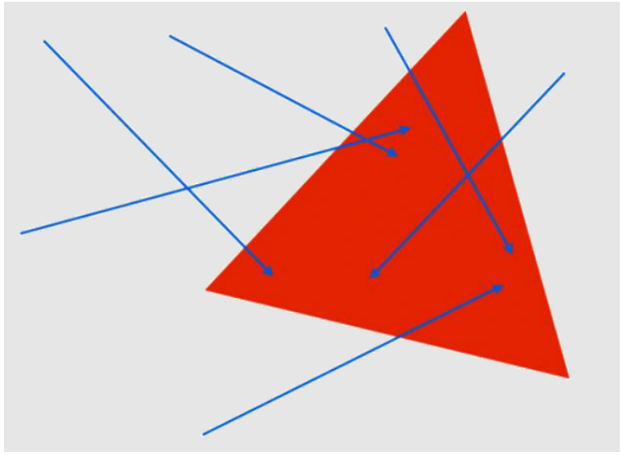


Figure 1: An example of a highly coherent scene despite the seemingly random rays. *Image from Oscar Dadfar's 15-473 lecture slides.*

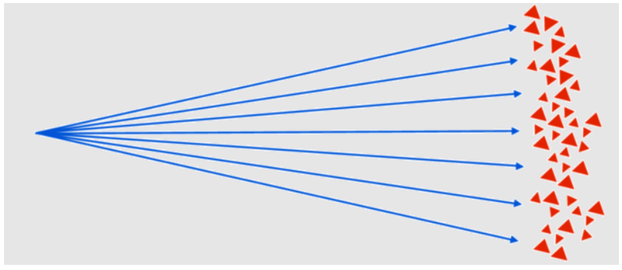


Figure 2: An example of a highly incoherent scene despite the rays starting from the same location with a relatively small angle. *Image from Oscar Dadfar's 15-473 lecture slides.*

I've written Ray Packet versions or overloads of the following functions, using C++ arrays to simulate SIMD structures and code logic:

- `src\lib\bbox.h:hit()`→`hit_packet()`
- `src\pathracer\aggregate.h:hit()`→`hit_packet()`
- `src\pathracer\bvh.*:hit()`→`hit_packet()`
- `src\pathracer\instance.h:Instance::hit()`→`hit_packet()`
- `src\pathracer\tri_mesh.*:Tri_Mesh::hit()`→`hit_packet()`
- `src\pathracer\shape.*:Sphere::hit()`→`hit_packet()`
- `src\pathracer\pathtracer.*:Pathtracer::sample_direct_lighting_task4()`
- `src\pathracer\pathtracer.*:Pathtracer::sample_direct_lighting_task6()`
- `src\pathracer\pathtracer.*:Pathtracer::sample_indirect_lighting())`
- `src\pathracer\pathtracer.*:Pathtracer::trace())`

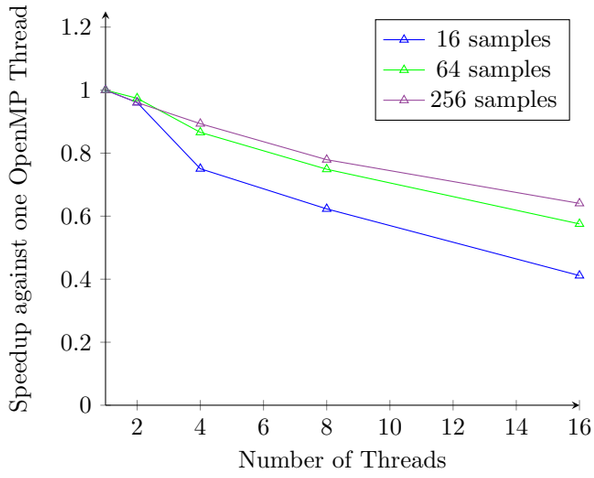
Results

Currently, the GUI now supports our opitons for different rendering modes. The only code that works is using OpenMP threads on the outermost loop that samples and traces individual rays through the scene (using the Packet Option renders an all black scene). Scotty3D does support the option for packets and multiple OpenMP threads!

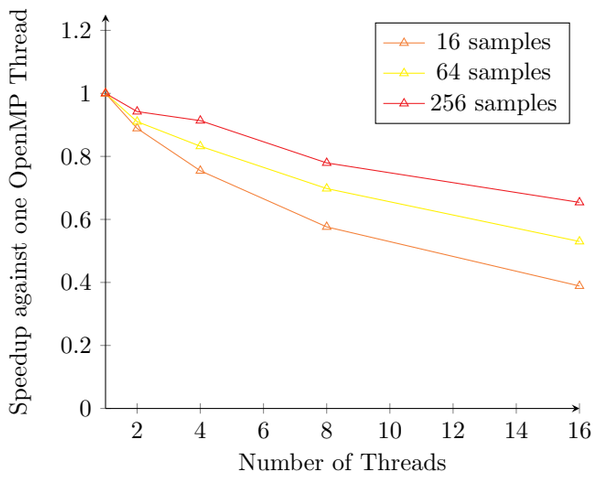
To debug the black screen, I've tried tracking the control flow of different rays, and the way my if statements split my vectors. I was able to patch a bug involving a mistyped function argument for the bounding box, but the bug right now seems to be with the bsdf being assigned to the vector.

As stated before, this actively slowed the work down. The following tests were run with 8 ray bounces, with a film height of 480×480, run on ghc39.

Speedup vs. *Number of OpenMP Threads*, `cbox-spheres`



Speedup vs. *Number of OpenMP Threads*, `cbox-spheres-lambertian`



Analysis

- Notably, introducing Open MP threads *slows down* Scotty3D! This is likely due to my attempt at parallelization getting in the way of the hardware parallelization, messing with locality and caches.
- As the sample count goes up, the slowdown gets less bad, likely because the hardware parallelization has to deal with more cache misses as the sample count goes up, so there's less parallel work that we can mess up.
- One thing I could do is run `perf stat` on the command line version of Scotty3D to determine average per-thread cache misses.

Future Work

I would like to actually implement the Intel Intrinsics over the vectorized code, to truly test if we get any speedup. While I believe my simulation using C++ arrays is accurate to the vectors I'd need, I won't know if it's correct or faster until testing. I'd also like to play around with using Arrays of Structures or Structures of Arrays. The Ray Packet structure utilizes the latter, but I wonder how much of the code I could refactor to support SoA (e.g., Trace or Spectrum), or how much those would even be needed, since the SIMD largely applies to the ray traversal and it probably won't take too long to extract what we need and put it in a vectorized form (though this of course would have to be tested). I would also like to use the command line version of SCotty3D with `perf stat` given more time.

Beyond the current implementation and its goals, something I'd be interested in researching is a heuristic for when to abandon ray packets, *before* traversing a particular incoherent packet. SPMD is broadly the favored approach for ray tracing a scene due to its relative simplicity and versatility. But Packet Tracing does still have some uses, as it's very effective for reflective surfaces (thanks to not degrading coherence too much), point light shadow ray tracing, and for traversing higher levels of a BVH [Wald et al. 2007]; we can dynamically abandon packets at specific thresholds if we *observe* that they become incoherent enough [Benthin et al. 2011]; and we can employ packet reordering to preserve coherence, though this is often very complex to do [Boulos et al. 2008]. But these are often reactive to incoherent ray packets being traversed; is there a way to do this proactively?

References

BENTHIN, C., WALD, I., WOOP, S., ERNST, M., MARK, W. R. 2011. Combining single and packet ray tracing for arbitrary ray distributions on the Intel® MIC architecture.

BOULOS, S., Wald, I., BENTHIN, C. 2008. Adaptive Ray Packet Reordering.

WALD, I., BOULOS, S., SHIRLEY, P. 2007. Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies.

Oscar Dadfar's lecture slides for the course 15-473.

Work Per Student

Ashley wrote the Ray Packet structure in `src\lib\Ray.h` and the initial trace initialization code in `Pathtracer::do_trace()` before having to withdraw from the course. After this, the work was my own.