15-418 Project Milestone: Parallel Ray Tracer

Samuel Schaub and Nader Daruvala

December 15, 2023

1 Updated Schedule

1.1 Dec 3 - 6

1. Finish computing pixel colors in an image in parallel. (Nader)
2. Implement tiling of image and processing of tiles on multiple cores in parallel. (Sam)
3. Implement explicit tracing of multiple rays in one vector (ray packeting). (Nader + Sam)

1.2 Dec 6 - 9

1. Implement Bounding Volume Hierarchy (BVH) for logarithmic complexity intersections. (Sam)
2. Implement a mesh data structure for modeling complex objects. (Nader)
3. Implement simple textures for objects. (Sam)
4. Implement lights and tracing of rays back to light sources. (Nader)
5. Implement a Cornell Box and test performance against sequential code. (Sam)

1.3 Dec 9 - 12

1. Implement ray recording heuristic to increase SIMD utilization rate. (Nader + Sam)
2. (Reach Goal) Implement cache-optimized BVH treelets. (Nader + Sam)

1.4 Dec 12 - 15

1. Finalize graphs, write-up, and poster board. (Nader + Sam)
2. Print out poster board. (Nader)
3. Practice presenting poster board a few times. (Nader + Sam)

2 Current Progress

So far, we have built a sequential ray tracer from scratch that is capable of tracing rays that reflect off of Lambertian (Matte) objects. We have implemented spheres as our only object type at the moment. We then implemented a separate version of the sequential code in ISPC and obtained correctness for this implementation. Currently, the parallel implementation parallelizes over the pixels in an image. For each pixel in a vector of 8 pixels, we sample some number of rays randomly within that pixel and trace each of these sampled rays sequentially within each pixel. To implement the current version of our algorithms, we have been taking inspiration from Peter Shirley's "Ray Tracing In One Weekend" book series. We have been following his tutorials to implement our sequential version and then implementing our ISPC version based on that sequential version.

3 Goals and Delivarables

We are a little behind our goals because we had a lot of trouble translating C++ templates and classes to ISPC so we had to scrap most of our sequential code written using modern C++ techniques and start from scratch. We believe that we will be able to get ray packeting and reordering finished. However, the cache-optimized BVH tree is now a stretch goal. Another "nice to have" is additional materials such as glass and mirrors. The new list of goals for the project sessions will be ray packeting, ray reordering, BVH trees, meshes, lights, and textures.

4 Poster Session

At the poster session, we plan to show a few things. Firstly, we would like to show off a few of the rendered images we have created using our algorithm. We would like also to show the speedup that our parallel algorithms achieved for these specific images. We plan to create graphs that show the speedup from our sequential implementation, our implementation that parallelizes over pixels, and our implementation that parallelizes over separate rays. If we are able to get ray reordering or cache-optimized BVH trees implemented, we would also like to show off the performance obtained with and without these optimizations.

5 Preliminary Results

Image Dimensions (px)	Speedup Over Sequential
400 x 225	5.32x
1920 x 1080	5.67x
2560 x 1440	5.70x
3840 x 2160	5.70x

Tested with 50 samples per pixel and a maximum recursion depth of 100.

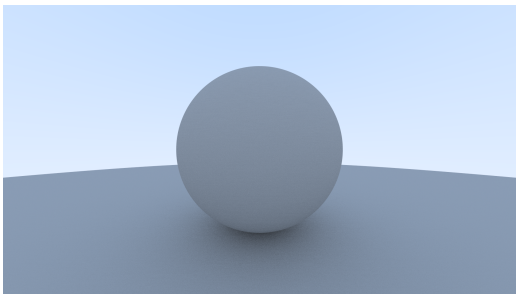

Figure 1: Image used in testing above

6 Concerns and Issues

Our main concern will be writing polymorphic code and getting it to compile properly in ISPC without excessive performance warnings. For example, eventually, we will need to define a generic “hittable object” type that implements a ”hit” function (a function that takes a ray and tests to see if a ray intersects with that object or not). This object type will be useful when we would like to have different types of primitives in our scene, including objects made up of quadrilaterals (meshes) and spheres together.