ECE408 Final Project Milestone 1

Group 08

Ray Tracing Algorithm

Ray tracing is a rendering technique widely used in computer graphics. In short, the principle of ray tracing is to trace the path of the light through each pixel, and simulate how the light beam encounters an object. Unlike rasterization technique which focuses on the realistic geometry of objects, ray tracing focuses on the simulation of light transport, where each ray is computational independent. This feature makes ray tracing suitable for a basic level of parallelization.

Potential Parallelism

Due to the characteristics of ray tracing algorithms, it is straightforward to convert the serial code into a parallel one, and the given code already implements a CPU-based parallel approach using OpenMP, where each image row runs in parallel. For the GPU parallel approach, we can use two-dimensional thread blocks and grids, in which each thread does computation for one pixel or a block of subpixels. Shared memory can be used to decrease global memory accesses.

However, because of the divergence of ray paths, the computational workload of each pixel is different. Thus it is hard to achieve high utilization and low overhead under parallelization. Therefore, computation uniformity and branch divergences should be taken care of to optimize the performance.

Related Parallel Approaches

Now, there are many codes to generate images using ray tracing algorithms, which can run on CPU or GPU in single or multi-thread methods. Existing CPU-based parallel approaches include dynamic scheduling and shared memory multiprocessing using OpenMP.

For GPU-based parallelism, there are various papers on this topic that targets specific algorithms or data structures that can be used in ray tracing and implement new optimized structures suitable for GPGPU computing. In 2018, the announcements of NVIDIA's new Turing GPUs, RTX technology spurred a renewed interest in ray tracing. NVIDIA gave a thorough walk through of translating C++ code to CUDA that results in a 10x or more speed improvement in their book *Ray Tracing in One Weekend*.

After doing some research online, we find some links/implementations that could be useful for our next step, listed in references.

Environment Set Up

All of our group members have successfully installed and setuped RAI, and are able to build and make a trial run of the provided code on the local machine. The average running time on our local machine (CPU implementation) running `./smallpt` is as follows:

[2.5 GHz Intel Core i7]

Number of samples per pixel	real	user	sys
100	2m8.663s	1m56.878s	0m1.053s
300	10m41.490s	8m8.317s	0m4.614s
500	12m36.452s	11m11.927s	0m5.339s

References

[1] Accelerated Ray Tracing in One Weekend in CUDA

[2] Optimizing Raytracing Algorithm Using CUDA

[3] StackOverflow: Raytracing with CUDA

[4] Allen, R. (2019, June 4). Accelerated Ray Tracing in One Weekend in CUDA. Retrieved April 8, 2020, from https://devblogs.nvidia.com/accelerated-ray-tracing-cuda/

[5] Razian, Sayed & MahvashMohammadi, Hossein. (2017). Optimizing Raytracing Algorithm Using CUDA. Italian Journal of Science & Engineering. 1. 167-178. 10.28991/ijse-01119.

[6] J.-C. Nebel. A New Parallel Algorithm Provided by a Computation Time Model, Eurographics Workshop on Parallel Graphics and Visualisation, 24–25 September 1998, Rennes, France.

[7] A. Chalmers, T. Davis, and E. Reinhard. Practical parallel rendering, ISBN 1-56881-179-9. AK Peters, Ltd., 2002.

[7] Aila, Timo and Samulii Laine, Understanding the Efficiency of Ray Traversal on GPUs, High Performance Graphics 2009, New Orleans, LA.