Assignment 0: Describing a Parallel Application

(Woody) Ki Fung Chow

February 5, 2013

1 Bio

I am a CS senior student. My main field of interest is Computer Graphics. I am especially engaged in Ray Tracing and other new rendering algorithms. You can see my previous work at http://woodychow.com.

By taking this class, I would like to learn all the available tricks and tools to parallelize my CG programs. The first step would be parallelize my ray tracer as proposed below. On the other hand, I am planning on doing fluid simulation in CG for my final project in CS283. I hope that the knowledge I am acquiring in this class can immediately apply to that project.

2 An parallel application - Real Time Ray Tracing with CUDA

2.1 Introduction

Ray tracing is a rendering algorithm that trace light rays per pixel on an image plane, as opposed to rasterization, where it maps objects in a scene to the screen directly. This algorithm simulates light and ray physics to a certain extent that depends on the implementation. Therefore, it is capable of delivering very high degree of realism when done properly.

However, Ray Tracing has one fatal disadvantage – speed. Since its first appearance in Computer Graphics community in 1979[1], Ray Tracing had only been an algorithm for redering standstill images only because of the massive computation required to simulate ray physics; not until 2005 did the first real-time ray tracer was successfully implemented[2]. The most successful ray tracers nowadays are written in CUDA or OpenCL, running on GPGPU.

Real time ray tracing is still a very hot topic in the field of CG. I wish I can find a good partner, and together contribute to the CG community.

2.2 Parallelism in Ray Tracing

In the sense of operations, ray tracing is embarassingly parallel. The set of operations running for each pixel is identical. In simple recursive ray tracing for example, it involves casting a light ray from a pixel on the image screen into the scene, calculate its reflections and refractions, and sum up the color contribution at different intersections. Ray tracing should be easily parallelized.

However, ray tracing involves a huge dataset. A 3D model would consist at least thousands of faces, and obviously a scene would consist of hundreds of models; not to mention

the textures that are being pasted onto the modles. It logically follows that memory bandwidth is the bottleneck of ray tracing algorithm. The best example would be finding an intersection of the light way with the screen. For each light ray casted from the image plane, intersection is found by going through all the triangles in the scene. Even with space partitioning trees like KD-Tree or Bounding Volume Hierarchy(BVH), this process is still very slow.

GPGPU, which is good at repetitive instructions, having a bigger memory size than CPU cache and lower memory latency than DRAM, is a perfect fit for this problem.

2.3 Performance

The "vendor code" – Nvidia OptiX Ray Tracing Engine renders the Whitted's sphere scene, can render the canonical ray tracing scene with over 30fps on a GeForce GTX480 at 1000 by 1000 resolution. But for more complex scenes, like something one would expect in a video game or movie, is only rendering at a few fps[3].

References

[1] Whitted, Turner. "An Improved Illumination Model for Shaded Display". Proceedings of the 6th annual conference on Computer graphics and interactive techniques. 1979.

[2] "Ray tracing (graphics)". Wikipedia. 5 Feburary, 2013.

[3] Parker, Steven. "OptiX: A General Purpose Ray Tracing Engine". ACM Transactions on Graphics. August, 2010.