Project Proposal - EEC 277

Yuxin Chen and Ahmed H. Mahmoud

21 February 2017

1 Abstract

2 Motivation

1. Why GPU?
   In the last decade, the performance of GPUs has increased more rapidly than that of CPUs and most of current GPUs incorporate floating points units even double float points units, making GPUs a computing beast for parallelable computation. As the GPU evolves to include more instruction set and supports larger data types operations, it becomes a more and more general-purpose streaming processor. The programmable stages of graphics pipeline are enhanced and allow more flexibility and longer program that basic ray tracing components can be loaded as a single program. So the convergence of the three separate trends: sufficient raw performance for single-chip real-time tray tracing; increasing GPU programmability; and faster performance improvements on GPUs than CPUs make GPUs an attractive platform for real-time ray tracing.

2. Why using graphics pipeline?
   Ray tracing can be easily formulated into a streaming of computations which is easily mapped into a streaming processor. When we say streaming computing, it differs from traditional computing in that the system reads a sequential stream of data and execute a kernel on each element of the input stream of data. In this sense, a programmable graphics processor executing a vertex program on a stream of vertices,

and a fragment program on a stream of fragments is a streaming fragment processor.The streaming model of computation leads to efficient implementations on streaming processors for three reasons: 1) process the data in parallel since there is no dependence between data, 2) kernels achieve high arithmetic intensity, 3) streaming hardware can hide the memory latency of texture fetches by using prefetching **??**. So in this experiment, we are going to break the ray tracing into several kernels, chain those kernels with streams of data and map they onto streaming processor.

3. Elimination of control flow and recursion
   Ray tracing is usually used in a recursive manner. To compute the color of primary rays, recursive ray tracing algorithm casts additional, secondary rays creating indirect effects like shadows, reflection or refraction. It is possible to eliminate the need for recursion and to write the ray tracer in an iterative way which runs faster since we don't need function calls and the stack. Note that the graphics pipeline doesn't include data-dependent conditional branching in its instruction set. To overcome this limitation, conditionals can be mapped to the hardware architecture by using the multipass rendering technique by **??**: the conditional predicate is first evaluated using redering passes, and then a stencil but is set to true or false depending on the result. The body of the conditional is then evaluated using additional rendering passes, but values are only written to the framebuffer if the corresponding fragment's stencil bit is true. It is quite efficient to use multipass rendering using large fragment programs under the control of the stencil buffer.

4. Spatial data structure for acceleration
   In ray tracing process, finding the closest object hit by a ray requiring to check a set of objects but only take the closest one. A brute force approach that checks all the object with a ray is too expensive. Therefore, to accelerate this process, we can only check a subset of objects by using spatial data structure. Many spatial data structures are available such as BSP trees, KD-trees, octrees, uniform grids, adaptive grids, etc.

5. OpenGL and CUDA are two programming model on GPU, and they both support streaming computing. Although CUDA support branching and recursion on the hardware level, as we discuss before, control

flow and an alternative of recursion can also be implemented efficiently on rasterization pipeline. So we are going to implement ray tracing on GPU using both OpenGL and CUDA programming model and then compare their performance and try to find out if and why one programming model mapping onto the hardware outperformance the another.

3 Related Work

4 Goal

5 Milestones