Vision, Modeling, and Visualization (2013)
Michael Bronstein, Jean Favre, and Kai Hormann (Eds.)

Poster

Towards Multi-Kernel Ray Tracing for GPUs

T. Schiffer[1] and D. W. Fellner[1,2]

[1]Institut für Computergraphik & Wissensvisualisierung, TU Graz, Austria [2]TU Darmstadt & Fraunhofer IGD, Germany

Abstract

Ray tracing is a widely used algorithm to compute images with high visual quality. Mapping ray tracing computations to massively parallel hardware architectures in an efficient manner is a difficult task.
Based on an analysis of current ray tracing algorithms on GPUs, a new ray traversal scheme called batch tracing is proposed. It decomposes the task into multiple kernels, each of which is designed for efficient execution. Our algorithm achieves comparable performance to state-of-the-art approaches and represents a promising avenue for future research.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing I.3.1 [Computer Graphics]: Hardware Architecture—Graphics Processors G.1.0 [Mathematics of Computing]: General—Parallel algorithms

1. Introduction

Ray tracing is a popular algorithm to compute high-quality renderings of complex scenes. Its huge computational requirements make massively parallel hardware architectures like modern graphics processing units (GPUs) attractive target platforms for implementations. We investigate high performance ray tracing on NVidia GPUs and we focus on the task of intersecting a ray with a scene containing geometric primitives only and omit shading and other operations. In this paper, we use bounding volume hierarchies (BVHs) as acceleration structure for ray traversal and evaluate the presented algorithms on ray loads generated by a path tracer with a fixed maximum path length of three bounces for different test scenes shown in Figure 1.

Figure 1: *Ray tracing-based renderings of our test scenes. From left to right: Sibenik (80K triangles), Conference (282K triangles) and Museumhall (1470K triangles).*

2. Analysis of Current Approaches

Aila et al. [AL09] presented efficient depth-first BVH traversal methods, which use a single kernel containing traversal and intersection, that is run for each input ray in parallel. More recently, they presented some improvements to increase the SIMT efficiency for NVidia's Kepler architecture in [ALK12]. Given two different rays contained in the same warp, potential inefficiencies in their approach now stem from the fact that the rays either require different operations (e.g. one ray needs to execute a traversal step, while the other ray needs to perform primitive intersection) or the sequences of required operations have a different length (e.g. one ray misses the root node of the BVH, while the other ray does not). These two fundamental problems have a negative impact on SIMT efficiency and lead to an uneven distribution of work among the active threads, especially for incoherent ray loads (see also [TS]).

Garanzha et al. presented a novel traversal algorithm for BVHs in [GL10] implemented using multiple kernels. The input rays are partitioned into coherent groups bounded by frusta, which are then used in the breadth-first traversal of the BVH. This stage yields lists of intersected leaves for each frustum. After traversal, the rays are tested for intersection with the primitives contained in each leaf to obtain the final results. Large parts of their implementation are designed to exhibit high SIMT efficiency, since each kernel is carefully designed to perform a single task (e.g. frustum intersec-

DOI: 10.2312/PE.VMV.VMV13.227-228

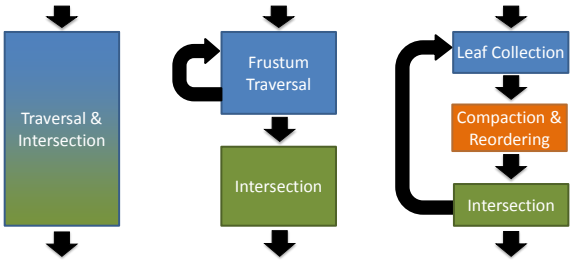

Figure 2: *High-level system designs for monolithic depth-first (left), breadth-first (middle) and our batch tracing approach (right). Blocks denote logical parts of the algorithms, black arrows denote the control flow.*

tion) and complex operations are broken down into smaller tasks. However, the approach performs a complete traversal of the acceleration structure for each frustum regardless of the number of the actually necessary operations. So a lot of potentially redundant work per ray is carried out, which decreases overall performance, especially for incoherent rays.

3. Multi-Kernel Batch Traversal

Based on these observations, we propose a novel approach called **batch traversal** that is designed to achieve the following objectives:

- Given the low SIMT efficiency for incoherent rays of depth-first traversals, our algorithm notably increases SIMT efficiency.
- Unlike breadth-first traversal, the algorithm does not exhibit substantial inefficiencies in handling ray loads of varying coherency.

The components of our algorithm and their interplay are shown in Figure 2 (right) together with depth-first (left) and breadth-first (middle) traversals. Like in breadth-first traversal, our approach splits the ray tracing task into several algorithmic stages implemented in multiple kernels to increase SIMT efficiency of the single code parts. During leaf collection stage, the ray traverses the BVH similar to depth-first traversal and collects a number of intersected leaf nodes called intersection candidates. In the subsequent phase, the active rays and the collected intersection candidates are reorganized to maintain efficient execution of the following stages. In the intersection phase the primitives contained in the intersection candidates are tested for intersection with the ray and the results are used to update the rays. These stages are executed in a loop until all input rays have terminated. Contrary to breadth-first traversal, our approach performs partial BVH traversals and intersection testing alternately in order to avoid collecting a large number of redundant intersection candidates.

Ray Type	Mono	BT (Base)	BT (Opt.)
1. Bounce	66.8 / 47.7	66.5 / 58.3	69.3 / 58.3
2. Bounce	40.3 / 33.2	36.7 / 47.1	56.8 / 47.1
3. Bounce	36.3 / 30.3	32.6 / 45.4	54.6 / 45.4

Table 1: *SIMT efficiency percentages for traversal and intersection of monolithic depth-first kernels (Mono) and batch tracing kernels (BT), higher values are better.*

4. Results and Discussion

For the practical evaluation, we provide a baseline implementation of our algorithm and an optimized variant, which dynamically fetches new rays in the leaf collection stage, if the SIMT efficiency drops below a certain threshold. As shown in Table 1, our algorithms can substantially improve the SIMT efficiency of the traversal and the intersection stage compared to state-of-the-art monolithic depth-first traversals. However, these SIMT efficiency gains do not result in consistent performance wins as reported in Table 2, where a slight improvement can only be measured on a low-end GT 540M chip. Although our current implementation cannot quite compete with mature and heavily optimized monolithic traversal in the other test scenarios, our multi-kernel method possesses appealing characteristics, which makes it an attractive direction for further research.

GPU	Ray Type	BT (Base)	BT (Opt.)
GT 540M	1. Bounce	0.86	0.91
	2. Bounce	0.97	1.05
	3. Bounce	0.91	1.00
GTX 590	1. Bounce	0.72	0.72
	2. Bounce	0.77	0.82
	3. Bounce	0.74	0.80

Table 2: *Ray tracing performance for the batch tracing kernels relative to monolithic kernels, higher values are better.*

References

[AL09] AILA T., LAINE S.: Understanding the efficiency of ray traversal on gpus. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 145–149. 1

[ALK12] AILA T., LAINE S., KARRAS T.: *Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum*. NVIDIA Technical Report NVR-2012-02, NVIDIA Corporation, June 2012. 1

[GL10] GARANZHA K., LOOP C.: Fast ray sorting and breadth-first packet traversal for gpu ray tracing. In *Proceedings of the Eurographics* (2010), EG '10, Eurographics Association, pp. 289–298. 1

[TS] T. SCHIFFER D. W. F.: Ray tracing: State of the art and challenges. *Accepted for publication in IEEE Potentials*. 1