# A straightforward CUDA Implementation for Interactive Ray-Tracing

Brian C. Budge, John C. Anderson, Christoph Garth, Kenneth I. Joy *
University of California, Davis

## ABSTRACT

In recent years, applying the powerful computational resources delivered by modern GPUs to ray tracing has resulted in a number of ray tracing implementations that allow rendering of moderately sized scenes at interactive speeds. In our poster, we present a fast implementation for ray tracing with CUDA. We describe an optimized GPU-based ray tracing approach within the CUDA framework that does not explicitly make use of ray coherency or architectural specifics and is therefore simple to implement, while still exceeding performance of previously presented approaches. Optimal performance is achieved by empirically tuning the ray tracing kernel to the executing hardware. We describe our implementation in detail and provide a performance analysis and comparison to prior work.

**Index Terms:** I.3.6 [Computer Graphics]: Graphics data structures and data types—Methodology and Techniques Realism; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

## 1 MOTIVATION

GPU-based ray tracers have recently achieved a performance equal and surpassing that of CPU implementations. Various techniques have been proposed to adapt ray tracing to GPU acceleration, and such optimizations are typically aimed at addressing both strengths and limitations of the underlying architectural programming model. Typical approaches exploit SIMD hardware characteristics through the use of coherency between neighboring rays (e.g. [1]), and limit the use of per-ray traversal state by employing specialized acceleration data structures (cf. [3, 2]). Both approaches complicate implementation of a GPU ray tracer since the required data structures and algorithms deviate strongly from their CPU equivalents.

Under CUDA, graphics hardware is able to support the execution of general purpose programs with a large number of concurrent threads. Typically, the number of simultaneous threads is much larger than the number of available execution units, and sophisticated hardware scheduling techniques are applied to make maximal use of the available execution resources through techniques such as latency hiding and zero-cost thread switching. Because of limited resources, ray tracing implementations on GPUs are often explicitly implemented to achieve the sweet spot of a specific hardware implementation such as the G80 architecture; to guarantee optimal performance, complex analysis of SIMD mapping, register counts and local memory constraints per thread is required ([1] for example). Resulting algorithms may not perform optimally on future hardware.

We present an optimized GPU-accelerated ray tracing implementation within the CUDA framework based on fast stack-based kd-tree traversal. Each ray is mapped to a thread, and a single kernel is used for the entire ray tracing pipeline including shading. Our ray tracer is not tailored to a specific number of execution units or state size, nor does our kernel address ray coherency; instead, we implicitly make use of hardware scheduling characteristics by selecting execution parameters through experiment. This allows us to both achieve optimal performance and preserve simplicity of implementation. We base our code on generic optimization principles and deviate from an optimized CPU implementation only in the use of a small number of specialized instructions. We provide an in-depth description of our ray tracing kernel and examine optimal execution parameters, and we give performance numbers on typical scenes and compare them to prior work.

*e-mail: {bcbudge|janderson|cgarth|kijoy}@ucdavis.edu

IEEE/EG Symposium on Interactive Ray Tracing 2008
9 - 10 August, Los Angeles, California, USA
978-1-4244-2741-3/08/$25.00 ©2008 IEEE

## 2 IMPLEMENTATION

Our ray tracing kernel is an adaptation of the classic stack-based traversal for the CUDA architecture, augmented by several optimizations. First, the texture cache is heavily exploited, and linear textures are used to store kd-tree as well as triangle information. This yields significant speed-ups for the most commonly traversed nodes near the top of the tree, as well as speeding up computation in regions where rays show strong coherency. Additionally shared memory is used to store ray origin and direction vectors in order to allow fast indexing independent of access pattern. To optimally exploit coherence in hardware scheduling, we emperically determine kernel block sizes by explicit measurement. Because this test can be carried out in real-time, the implementation is robust to changes in hardware specifics, and has the potential to scale to future architectures.

## 3 RESULTS

To judge the general performance of our kernel after optimization, we have benchmarked it on standard scenes and compared our results with recently published data on a standard viewport of $1024 \times 1024$. For most scenes, we strongly outperform available GPU implementations in both fully-shaded and non-shaded scenarios. Most notably, we achieve 30 frames per second for the 12.7 million triangle Power Plant scene for primary rays and 15.5 frames per second with shading and shadows. This surpasses ealier work which reports 6.4/2.9 frames per second [1]. To examine the dependence on ray coherency, we additionally benchmarked random rays generated on a scene-encompassing sphere and achieved nearly 14 Mrays/sec.

## REFERENCES

[1] J. Günther, S. Popov, H.-P. Seidel, and P. Slusallek. Realtime ray tracing on GPU with BVH-based packet traversal. In *Proceedings of the IEEE/Eurographics Symposium on Interactive Ray Tracing 2007*, pages 113–118, Sept. 2007.

[2] D. R. Horn, J. Sugerman, M. Houston, and P. Hanrahan. Interactive k-d tree gpu raytracing. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 167–174, New York, NY, USA, 2007. ACM.

[3] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum*, 26(3):415–424, Sept. 2007. (Proceedings of Eurographics).