An Evaluation of Existing BVH Traversal Algorithms for Efficient Multi-Hit Ray Tracing

Jefferson Amstutz
Applied Technology Operation
SURVICE Engineering Company

Johannes Günther
Intel Corporation

Ingo Wald
Intel Corporation

Christiaan Gribble
Applied Technology Operation
SURVICE Engineering Company

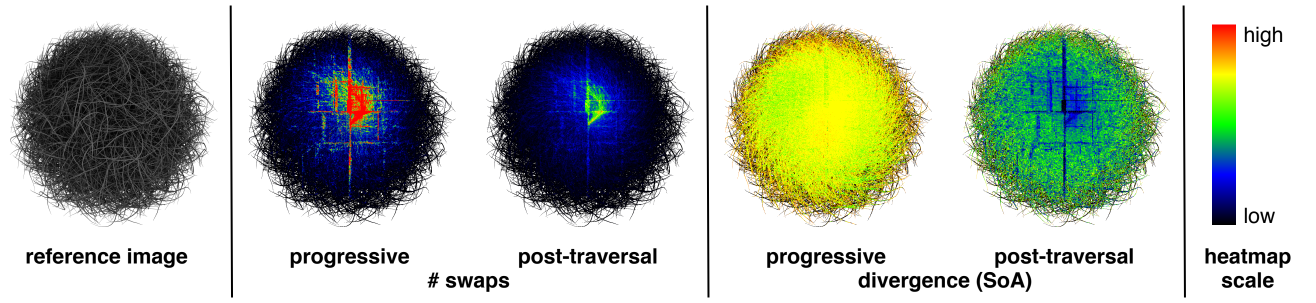

Figure 1: Efficient multi-hit ray tracing using existing BVH traversal algorithms. *Many ray tracing applications in optical and non-optical domains require multiple intersections along each ray, or so-called* multi-hit ray traversal. *Sorting hit points efficiently in light of memory bandwidth limitations has a major impact on multi-hit ray tracing performance in a BVH. Here, heatmap visualizations depicting the number of swaps imposed by sorting, as well as SIMD utilization of the sorting process, reveal the increased efficiency of post-traversal selection sort.*

1 Overview

We explore techniques for multi-hit ray tracing in a bounding volume hierarchy (BVH). BVHs are problematic for implementing multi-hit ray traversal correctly due to the potential for spatially overlapping leaf nodes. We demonstrate that the intersection callback feature of modern, high performance ray tracing APIs enables correct and efficient implementation of multi-hit ray tracing despite this concern. Moreover, the callback-based approach enables multi-hit ray tracing using existing, highly optimized BVH data structures, mitigating maintenance issues imposed by hand-tuned multi-hit traversal kernels across various hardware architectures.

2 Multi-hit traversal in a BVH

Previous work on multi-hit ray traversal [Gribble et al. 2014] assumes an acceleration structure based on spatial subdivision, in which leaf nodes of the structure do not overlap. With these structures, ordered traversal—and therefore generating ordered hit points—is straightforward: sorting is required only within, not across, leaf nodes. However, ordered traversal in a structure based on object partitioning, such as a BVH, is not achieved so easily; in fact, ordered ray traversal is at odds with the BVH structures employed by modern, high performance ray tracing engines. At first glance, then, it is unclear that a BVH can provide reasonable multi-hit performance. Surprisingly, however, we demonstrate several techniques to implement multi-hit traversal in a BVH efficiently.

3 Methodology

Multi-hit ray tracing in a BVH therefore requires careful attention to several factors, including:

- **Efficient traversal.** We use intersection callbacks to implement naive multi-hit ray traversal efficiently in Embree [Wald et al. 2014] and OptiX [Parker et al. 2010] while leveraging their existing, highly optimized BVH acceleration structures.
- **SoA v. AoS memory layout.** Incoherent sorting operations among rays incur scatter/gather operations per-element of a SoA hit point structure, but AoS enables vector memory operations to read or write entire structures more efficiently.
- **Hit point sorting.** Hit points must be sorted to meet the ordering constraint of multi-hit ray traversal, so we explore two sorting methods: progressive insertion sort during traversal and post-traversal selection sort.

We evaluate these techniques using several scenes of varying complexity, including the *hairball* scene depicted in Figure 1.

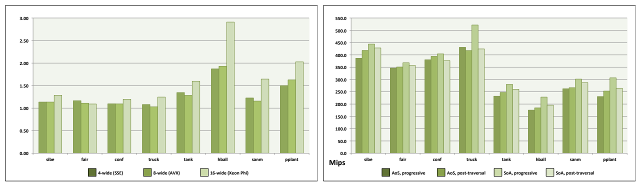

Figure 2: Observed efficiency & performance. *On average, post-traversal sorting improves SIMD utilization by about* $1.41\times$ *for our test scenes (left). Similarly, coherent progressive sort with SoA hit point data performs best for our scenes (right).*

4 Results

For CPU and Xeon Phi results, we build on the OSPRay rendering framework [Intel Corporation 2014] to execute Embree kernels in various hardware and software configurations. For GPU results, we base our implementation on the OptiX SDK examples. We also compare against Rayforce,[1] the open source GPU ray tracing engine used in previous work.

As can be seen in Figure 2, results show that deferring sort until after traversal improves efficiency by reducing memory swaps and increasing SIMD vector utilization. However, post-traversal sort does not necessarily guarantee maximum performance: memory bandwidth constraints on current hardware also impact sorting performance and can be justification alone for preferring an AoS or SoA hit point data layout, regardless of efficiency.

References

GRIBBLE, C., NAVEROS, A., AND KERZNER, E. 2014. Multi-hit ray traversal. *Journal of Computer Graphics Techniques 3*, 1, 1–17.

INTEL CORPORATION, 2014. OSPRay: A ray tracing based rendering engine for high-fidelity visualization. Available via `http://ospray.github.io`. Last accessed 5 June 2015.

PARKER, S. G., BIGLER, J., DIETRICH, A., FRIEDRICH, K., HOBEROCK, J., LUEBKE, D., MCALLISTER, D., MCGRUIRE, M., MORLEY, K., ROBISON, A., AND STICH, M. 2010. OptiX: a general purpose ray tracing engine. *ACM Transactions on Graphics (Proceedings of ACM SIGGRAPH 2010) 29*, 4.

WALD, I., WOOP, S., BENTHIN, C., JOHNSON, G. S., AND ERNST, M. 2014. Embree - a kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics 33*, 4 (July), 143:1–143:8.

[1] *Rayforce*: Exceptional performance through non-traditional means. Source code is available via `http://rayforce.survice.com`.