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.

Multi-hit traversal in a BVH

Previous work on multi-hit ray traversal 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.

Results

For CPU and Xeon Phi results, we build on the OSPRay rendering framework, which provides a flexible mechanism to execute Embree kernels in various hardware and software configurations. For GPU results, we use OptiX SDK examples as the basis for our implementation. We also compare against Rayforce, the open source GPU ray tracing engine used in previous work.

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.

Methodology

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

- **Efficient multi-hit traversal.** We use intersection callbacks to implement naïve multi-hit ray traversal efficiently in Embree and OptiX 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 the eight scenes and viewpoints depicted below.