Implementing Node Culling Multi-Hit BVH Traversal in Embree

Christiaan Gribble
SURVICE Engineering

Ingo Wald
Intel

Jefferson Amstutz
Intel

Abstract

We present an implementation of node culling multi-hit BVH traversal in Embree. Whereas previous work was limited by API restrictions, as of version 2.10.0, Embree respects ray state modifications from within its intersection callbacks (ICBs). This behavior permits an ICB-based implementation of the node-culling algorithm, but requires a trick to induce culling once the necessary conditions are met: candidate intersections are accepted with a (possibly) modified t_{far} value, so that ray traversal continues with an updated ray interval. We highlight a scalar implementation of the Embree ICB and report results for a vector implementation in OSPRay that improves performance by as much as $2\times$ relative to naive multi-hit when users request fewer-than-all hits.

1. Introduction

Multi-hit ray traversal is a class of ray-traversal algorithms that finds one or more, and possibly all, primitives intersected by a ray ordered by point of intersection [Gribble et al. 2014]. Amstutz et al. [2015] show that intersection callbacks (ICBs) in Embree [Wald et al. 2014] and OptiX [Parker et al. 2010] enable correct implementation of naive multi-hit traversal in a BVH while leveraging these engines' existing BVH construction and traversal routines.

To avoid the cost of naive multi-hit in a BVH, Gribble [2016] introduces the node culling multi-hit BVH traversal algorithm, which keeps a running tally of the N closest intersections and, after $N \geq N_{\text{query}}$ intersections have been found, uses the distance to the farthest intersection among the first N_{query} to cull nodes during traversal. This approach offers potentially significant improvement in multi-hit performance for cases in which users request fewer-than-all hits, but requires modifications either to internal traversal routines—making any such implementation incompatible with mainstream Embree developments—or to ray state from within the nodeculling ICB, which was not permitted by the Embree API at the time.

However, as of version 2.10.0, Embree permits updates to t_{far} from within its ICBs, or *intersection filters*, which in turn allows correct implementation of node culling multi-hit BVH traversal: after identifying N_{query} intersections, we set t_{far} to be the farthest of these and begin accepting candidate intersections, so that subsequent traversal operations cull nodes beyond the closest N_{query} intersections. In this way, we implement the node-culling algorithm without engine-level code specific to multi-hit traversal, resulting in potentially better performance while maintaining the benefits of an ICB-based approach.

We highlight a scalar implementation of the corresponding Embree intersection filter and report results for vector implementations of both naive and node culling multi-hit algorithms in OSPRay, a high-fidelity rendering framework based on Embree [Wald et al. 2016]. We also provide source code for both algorithms implemented as modules for OSPRay in the supplemental materials (`http://jcgt.org/published/0005/04/01/code.zip`) that accompany this work.

2. Implementation

At the time Gribble [2016] introduced the node-culling algorithm, Embree intersection filters could only accept or reject candidate intersections but could not modify hit data or otherwise influence subsequent traversal operations. As of version 2.10.0, however, Embree permits updates to t_{far} from within its intersection filters, behavior which permits correct implementation of node culling multi-hit BVH traversal.

In particular, once culling conditions are met, we *accept* candidate intersections with a t_{far} value possibly other than the incoming value—that is, we deliberately cause Embree to store hit data that does not (necessarily) correspond to the surface intersection for which the filter is actually invoked. In this case, traversal continues with the updated ray interval and thus culls nodes beyond the closest N_{query} intersections.

The scalar (single-ray) node-culling intersection filter is shown in Listing 1. Intersection data is collected in a preallocated per-ray buffer with exactly $N_{\text{query}} + 1$ entries. This approach allows us to always write (even potentially ignored) intersection data at index `idx` following the insertion-sort loop, which we empirically determine to be faster than conditional branching to handle hits beyond range. Additionally, each entry's `tval` member is initialized to some value beyond the range of valid intersections (e.g., `FLT_MAX`) to facilitate the insertion sort.

For each candidate intersection, the filter determines the index at which to store the corresponding data, actually stores that data, and updates the number of intersections collected so far. When this value is less than `Nquery`, the candidate intersection is rejected, which causes Embree to continue traversal with the incoming ray interval, $[\epsilon, t_{\max})$, as in previous work [Amstutz et al. 2015].

```
static void collectIntersectionsFilter(void* /* unused */,
                                       RTCRay& _ray)
{
  mhRay&   ray  = reinterpret_cast<mhRay&>(_ray);
  HitData* hits = ray.hits;

  // Find index at which to store candidate intersection
  uint idx = Nquery;
  while (idx > 0 && ray.tfar < hits[idx-1].tval)
  {
    hits[idx] = hits[idx-1];
    --idx;
  }

  // Store intersection, possibly beyond index of the
  //   N ≤ N_query closest intersections (i.e., at
  //   idx = Nquery)
  HitData& hit = hits[idx];
  hit.geomID   = ray.geomID;
  hit.primID   = ray.primID;
  hit.tval     = ray.tfar;
  hit.Ng       = ray.Ng;

  // Update number of intersections identified so far
  ray.nhits += (ray.nhits < Nquery ? 1 : 0);

  if (ray.nhits < Nquery)
  {
    // Reject intersection to continue traversal with
    //   incoming ray interval, as in previous work
    ray.geomID = RTC_INVALID_GEOMETRY_ID;
    return;
  }

  // Induce node culling
  //   Trick:  set ray.tfar to farthest value among the
  //           N = N_query intersections identified so far
  //           and (implicitly) accept intersection with
  //           modified ray interval
  ray.tfar = hits[Nquery-1].tval;
}
```

Listing 1. Node culling multi-hit intersection filter. Once culling conditions are met, candidate intersections are accepted with a t_{far} value possibly other than the incoming value so that Embree's ray traversal operations continue with the updated interval and thus cull nodes that the ray encounters beyond the farthest valid hit point.

However, once the number of intersections reaches `Nquery`, the filter assigns the value of the farthest valid intersection point along the ray, or `hits[Nquery-1]`, to the incoming ray's `tfar` member. Importantly, this value might be (and often is) *different* from the incoming t_{far} value. This trick is valid, as it meets the criteria for updating hit data from within an intersection filter in Embree version 2.10.0:

> The filter function is not allowed to modify the ray input data ... but can modify the hit data of the ray.... Updating the `tfar` distance to a smaller value is possible without limitation.[1]

The filter then (implicitly) accepts the intersection, so that the ray state (including `tfar`) is updated to values set in the filter rather than restored to its pre-filter values (as in the *reject-intersection* case), before Embree's traversal operations continue.

Using `ispc` [Pharr and Mark 2012], a vector implementation is nearly identical to the scalar implementation in Listing 1, with only minor modifications to accommodate the *single program, multiple data* programming model. The source code for both naive and node culling multi-hit BVH traversal algorithms, implemented as modules for OSPRay using `ispc`, is provided in the supplemental materials that accompany this work. Performance results for these modules are reported in the next section.

3. Results

We implement both naive and node-culling multi-hit BVH traversal algorithms as modules for OSPRay using `ispc`. We report performance for these vectorized modules using eight scenes of varying geometric and depth complexity rendered from the viewpoints depicted in Figure 1.

In particular, we measure multi-hit performance using the values of N_{query} considered by Gribble [2016]. For each test, we render a series of 10 warm-up frames followed by 100 benchmark frames at 1024×768 pixel resolution using visibility rays from a pinhole camera and a single sample per pixel. In these tests, the naive multi-hit implementation uses coherent post-traversal hit point sorting [Amstutz et al. 2015], which provides the best performance on our test platform. Results are obtained using two Intel Xeon E5-2650 v3 processors (20 cores, 40 hardware threads) averaged over the 100 benchmark frames.

For brevity, the graph in Figure 2 depicts results for only the *truck* scene; however, the supplemental data accompanying this paper includes multi-hit performance for all of our test scenes. Generally speaking, we observe trends present in results for the *truck* scene in results for the other scenes as well.

As in previous work [Gribble 2016], the impact of node culling varies directly with the number of requested intersections. In particular, when averaged across all

[1]Embree Documentation, *Embree API > Filter Functions > Normal Mode*, `http://embree.github.io/api.html`.

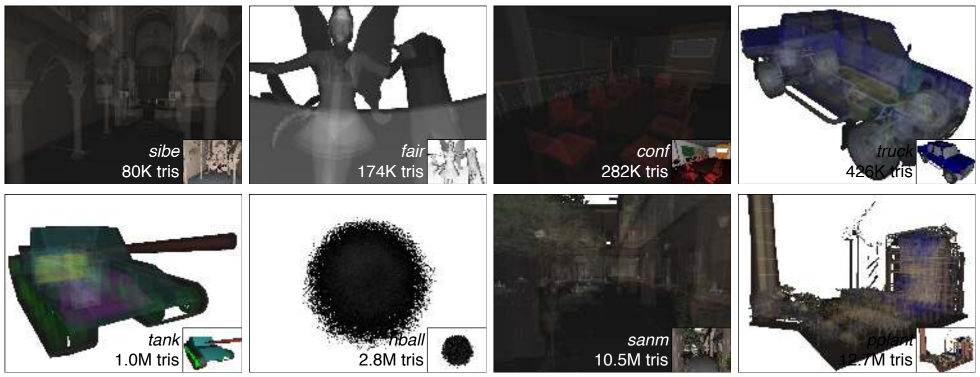

Figure 1. Scenes used for performance evaluation. Eight scenes of varying geometric and depth complexity are used to evaluate the performance of node culling multi-hit BVH traversal in Embree. First-hit visible surfaces hide significant internal complexity in many of these scenes, making them particularly useful in tests of multi-hit traversal performance.

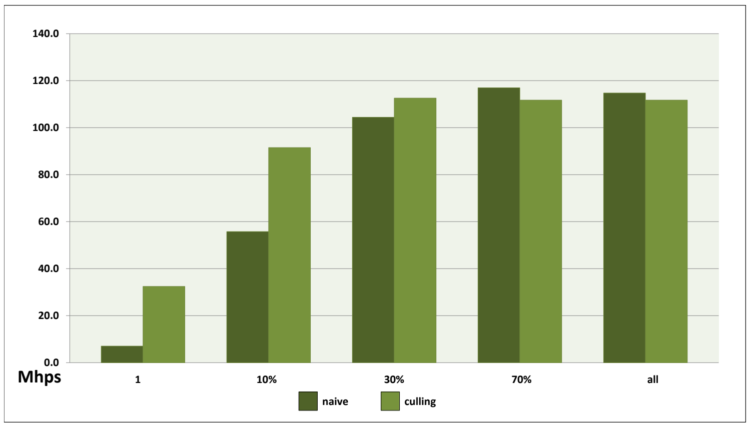

Figure 2. Performance of 8-wide vector implementations of the multi-hit variants in the *truck* scene. Here, the graph compares performance in millions of hits per second (Mhps) between naive and node culling multi-hit BVH traversal for various values of N_{query}.

of the test scenes, the impact of node culling on performance improvement relative to naive multi-hit decreases from a factor of 2 when $N_{\text{query}} = 1$ to effectively zero (or worse) when $N_{\text{query}} = \infty$. In fact, as $N_{\text{query}} \to 70\%$, naive multi-hit begins to outperform the node-culling approach, a result of the progressive insertion sorting operations invoked for each candidate intersection during node-culling traversal (and in contrast to the single coherent post-traversal selection sort employed by naive multi-hit traversal). Nevertheless, our node-culling intersection-filter implementation provides better performance for cases in which N_{query} is less than 70% of maximum.

4. Conclusions

We explore an implementation of ICB-based node culling multi-hit BVH traversal in Embree. As of version 2.10.0, Embree respects ray-state modifications from within its intersection-filter functions, which permits a correct implementation of the node-culling algorithm. We highlight an implementation trick to induce node culling from within the filter: once appropriate, candidate intersections are accepted with a t_{far} value possibly other than the incoming value, so that ray traversal continues with an updated ray interval. This approach implements node culling multi-hit BVH traversal without engine-level code and enables potentially better performance while maintaining the benefits of an ICB-based approach. Results obtained using eight scenes of varying depth complexity show that a vector implementation of node culling multi-hit BVH traversal in OSPRay improves performance by as much as $2\times$ relative to vectorized naive multi-hit when N_{query} is less than 70% of the maximum number of valid ray/primitive intersections.

References

AMSTUTZ, J., GRIBBLE, C., GÜNTHER, J., AND WALD, I. 2015. An evaluation of multi-hit ray traversal in a BVH using existing first-hit/any-hit kernels. *Journal of Computer Graphics Techniques 4*, 4, 72–90. http://jcgt.org/published/0004/04/04/. 1, 2, 4

GRIBBLE, C., NAVEROS, A., AND KERZNER, E. 2014. Multi-hit ray traversal. *Journal of Computer Graphics Techniques 3*, 1, 1–17. http://jcgt.org/published/0003/01/01/. 1

GRIBBLE, C. 2016. Node culling multi-hit BVH traversal. In *Eurographics Symposium on Rendering*, Eurographics Association, Aire-la-Ville, Switzerland. http://dx.doi.org/10.2312/sre.20161213. 1, 2, 4

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. http://dl.acm.org/citation.cfm?id=1778803. 1

PHARR, M., AND MARK, W. R. 2012. ispc: A SPMD compiler for high-performance CPU programming. In *Innovative Parallel Computing (InPar)*, IEEE, Los Alamitos, CA, 1–13. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6339601. 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. http://dl.acm.org/citation.cfm?id=2601199. 1

WALD, I., JOHNSON, G. P., AMSTUTZ, J., BROWNLEE, C., KNOLL, A., JEFFERS, J., GÜNTHER, J., AND NAVRATIL, P. 2016. OSPRay - a CPU ray tracing framework for scientific visualization. *IEEE Transactions on Visualization and Computer Graphics*. To appear. 2

Index of Supplemental Materials

We provide multi-hit performance data for all eight test scenes and highlight the key elements of the source distribution of our OSPRay multi-hit modules in `http://jcgt.org/published/0005/04/01/supplemental.pdf`.

Author Contact Information

Christiaan Gribble
SURVICE Engineering
6101 Penn Avenue, Suite 301
Pittsburgh, PA 15206
christiaan.gribble@survice.com
http://www.rtvtk.org/˜cgribble/

Ingo Wald
Intel Corporation
1300 South Mopac Expressway
Austin, TX 78746
ingo.wald@intel.com
http://www.sci.utah.edu/˜wald/

Jefferson Amstutz
Intel Corporation
1300 South Mopac Expressway
Austin, TX 78746
jefferson.d.amstutz@intel.com
https://www.linkedin.com/in/jeffersonamstutz

Christiaan Gribble, Ingo Wald, Jefferson Amstutz, Implementing Node Culling Multi-Hit BVH Traversal in Embree, *Journal of Computer Graphics Techniques (JCGT)*, vol. 5, no. 4, 1–7, 2016
http://jcgt.org/published/0005/04/01/

Received: 2016-08-06
Recommended: 2016-09-27 Corresponding Editor: Ulf Assarsson
Published: 2016-11-15 Editor-in-Chief: Marc Olano