Deferred Ray Tracing Using Intersection Binning

Apollo I. Ellis and Eric G. Shaffer and John C. Hart
University of Illinois at Urbana Champaign

1 Introduction & Related Work

We present an effective approach for dealing with incoherent memory accesses and thread divergence in GPU ray tracing of diffuse rays, *deferred ray tracing*. We separate traversal from intersection, working under the guiding principle that the coherence we regain will more than make up for any extra work required. Our method achieves this coherence with minimal overhead by employing a ray binning scheme. Our approach can outperform a forward ray tracer, by up to 20X, in apples to apples comparisons. For preliminary evaluation of deferred ray tracing as a general approach, we have implemented it with a two level world space grid structure, such as that of Kalojanov et al. [2011].

Aila and Karras introduced a queue-based architecture with bounding volume hierarchy modifications which is able to schedule randomly ordered rays to execute more coherently [2010]. Guntury and Narayanan used uniform grids to enforce coherence as well, but by explicit ray reordering [2012]. We also address coherence explicitly through ray reordering, but we employ a new low overhead arrangement technique, and unlike Guntury and Narayanan we observe significant improvement in intersection rates.

2 Design

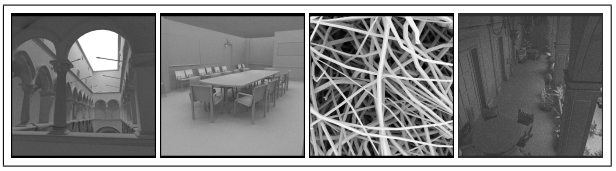
Our traversal is akin to that of Kalojanov et al., except we defer all intersection calculations, so traversal is more thread coherent. We store each ray-cell encounter as a pair, and we arrange these pairs in a buffer, such that pairs which contain the same grid cell are adjacent. This encourages adjacent threads to access the same memory locations.

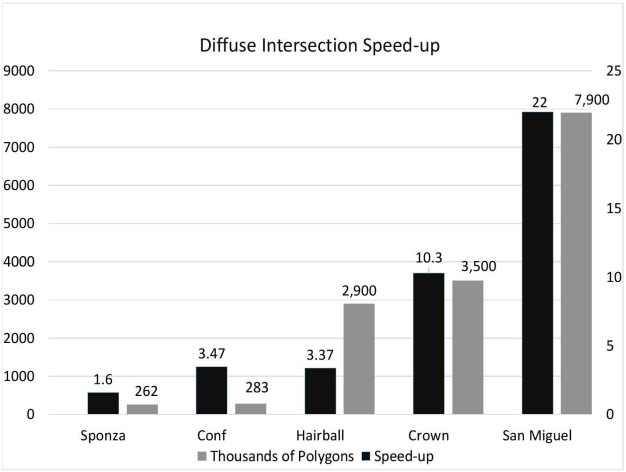
Arranging the cells in the buffer is a sort-like operation, and in fact sort is used in previous work [Guntury and Narayanan 2012]. Unfortunately, sorting the number of pairs encountered in complex scenes is prohibitive, around 1ms per 1M pairs. So we seek an alternative to sorting. We implement a two-pass binning traversal. The first traversal pass counts ray visits to each cell, and the second pass packs ray-cell pairs at array offsets according to the prefix sum of the counts. This packs pairs with a common cell adjacently. So we have obtained the same effective layout as sorting by cell, but our approach is extremely rapid, with negligible overhead since traversal is only an increment and store.

We execute intersection in waves of pairs, in order of encounter. We mark finished rays, and utilize GPU list compaction to remove them between waves. Thus we waste less work during intersection than a brute force all pairs intersection. Our intersector is designed to avoid branching. We transform geometry into ray space and execute a 2D point in polygon test. In total a test is six dot products and one cross product. We observe at least a 2X speed up over Mollër-Trumbore [1997].

3 Results and Conclusion

We set our first and second level grid resolutions at 8^3 and 128^3 respectively. Diffuse traversal in rays-per-second is shown in figure **1** bottom. For an intersection rate comparison with a forward system we compare against synthetic work optimal intersection. For each ray, we record how far into the grid we traversed to the final hit point. Then we run a kernel which spins over the recorded cells, in order, reports the first intersection, and halts. Here, a ray intersects a cell if and only if it would have on the path to the hit point in a forward ray tracer as well. This intersector is thus fair, baring obscure caching effects, and the scheduling may be more optimal than having to interleave traversal operations.





Scene	Sponza	Conf	Hair B.	Crown	S. Mig.
Rate	70M	142M	23M	65M	86M

Figure 1: *Views [top], Diffuse Intersect [mid], and Traversal [bot]*

With respect to traversal, when tracing diffuse rays in a grid, intersection calculations seem to dominate. Figure **1** middle, showcases our speed up over forward intersection relative to the scene size, it is near-linear! This speedup persists even if we reduce the forward ray tracer's hypothetical traversal cost to zero.

Uniform grid structures are not necessarily optimal for GPU ray tracing in comparison with recent work [Pérard-Gayot et al. 2017]. Thus, we need to evaluate deferral with hierarchical data structures. In addition, we believe deferred ray tracing could be significantly accelerated by empty occluders [Reshetov et al. 2005], which would yield less extra traversal steps and intersection tests. Overall, we believe deferred ray tracing has the potential to improve the state of the art in GPU ray tracing.

References

AILA, T., AND KARRAS, T. 2010. Architecture considerations for tracing incoherent rays. In *Proceedings of the Conference on High Performance Graphics*.

GUNTURY, S., AND NARAYANAN, P. J. 2012. Raytracing dynamic scenes on the gpu using grids. *IEEE Transactions on Visualization and Computer Graphics*.

KALOJANOV, J., BILLETER, M., AND SLUSALLEK, P. 2011. Two-level grids for ray tracing on gpus. *Computer Graphics Forum*.

MÖLLER, T., AND TRUMBORE, B. 1997. Fast, minimum storage ray-triangle intersection. *J. Graph. Tools*.

PÉRARD-GAYOT, A., KALOJANOV, J., AND SLUSALLEK, P. 2017. Gpu ray tracing using irregular grids. *Computer Graphics Forum*.

RESHETOV, A., SOUPIKOV, A., AND HURLEY, J. 2005. Multi-level ray tracing algorithm. In *ACM SIGGRAPH 2005 Papers*.