Improving BVH Ray Tracing Speed Using the AVX Instruction Set

Attila T. Áfra[1,2]

[1]Budapest University of Technology and Economics, Hungary
[2]Babeş-Bolyai University, Cluj-Napoca, Romania

Abstract

High performance ray tracing on the CPU requires the efficient utilization of SIMD instructions. Ray packet and ray stream traversal algorithms achieve this by performing computations on multiple rays, nodes, or primitives at the same time. In this paper, we present our approach to optimizing coherent BVH ray packet tracing for the new AVX instruction set, which enables 8-wide SIMD operations on 32-bit floating-point numbers. We have measured an average speedup of about 50% compared to our SSE4.1 implementation, on an Intel Sandy Bridge processor.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

Ray packet algorithms enable the fast tracing of coherent rays, which is especially important for real-time ray tracing solutions. One major source of performance improvement is the use of SIMD operations provided by the CPU. For example, the SSE instruction set enables the intersection of 4 rays with an acceleration structure node or a primitive approximately at the cost of 1. However, advanced packet-based approaches also have remarkable algorithmic benefits, which are independent of SIMD.

While optimizing ray tracing for multiple cores or processors is trivial and the resulting speedup is almost linear, exploiting instruction level parallelism can be quite challenging and the performance does not scale linearly with the SIMD width. If the rays are not coherent enough, the benefits of SIMD are limited because the SIMD unit utilization is low. Nevertheless, the performance gains can be significant in many cases, particularly for primary rays.

Until recently, popular CPU architectures, like x86, were able to operate on up to 4 single-precision floating-point numbers simultaneously. This has changed with the introduction of the 256-bit AVX (Advanced Vector Extensions) instruction set, which has double the SIMD width of SSE, AltiVec, NEON, etc.

We have optimized two BVH packet traversal algorithms for AVX: ranged traversal [WBS07] and partition traversal [ORM08]. The speed measurements were performed on an actual Intel Sandy Bridge CPU, which is the first processor implementing AVX.

2. AVX Ray Packet Tracing

The smallest ray primitive of both the ranged and partition traversal algorithms is the *SIMD ray*, which consists of multiple rays that are traced together throughout the entire algorithm. This enables efficient parallel intersection of rays with nodes (using the Kay-Kajiya slab test) and triangles. In 4-wide SIMD implementations, a SIMD ray usually contains 2×2 rays, thus, nearby rays are packed together to maximize coherence. For AVX, we employ 4×2 SIMD rays.

Ray packets can be quite large, commonly having a size of 256 or even 1024 rays, therefore, it is important to store the ray data in a cache-friendly way. This can be achieved by using an *array of structures of arrays* (AoSoA) layout, which, by grouping together the data belonging to a SIMD ray, combines the SIMD-friendliness of SoA (structure of arrays) with the locality of AoS (array of structures).

Ranged traversal is a relatively naive algorithm, which is very sensitive to the order of the rays in the packet. Partition traversal is significantly more robust in this area. In order to maximize speed, SIMD rays are stored in Morton-order.

Model	Single	Ranged			Ranged w/o culling			Partition			Partition w/o culling		
		SSE	**AVX**	**+**	**SSE**	**AVX**	**+**	**SSE**	**AVX**	**+**	**SSE**	**AVX**	**+**
	Mray/s	Mray/s	Mray/s	%	Mray/s	Mray/s	%	Mray/s	Mray/s	%	Mray/s	Mray/s	%
Conference	5.6	63.3	**93.2**	47.1	44.6	76.1	70.7	35.3	52.3	48.2	29.8	48.3	62.3
Sibenik	5.4	69.1	**102.1**	47.8	47.4	82.6	74.3	33.0	50.8	53.8	28.8	47.6	64.9
Crytek Sponza	2.7	27.5	**40.2**	46.3	20.2	34.3	70.1	16.1	23.2	44.2	14.0	22.4	60.3
Welsh Dragon	6.7	9.2	13.6	48.4	10.0	15.3	53.3	13.8	15.5	12.4	19.0	**22.9**	20.7

Table 1: *Performance in million rays per second for primary rays. The models were rendered from the viewpoints depicted in Figure 1. The timings do not include ray generation and shading. The highest value for each model is shown in bold, and the speedups caused by AVX over SSE4.1 are also listed. Measurements were executed for single ray traversal, ranged, and partition traversal, with and without frustum culling. The CPU used was an Intel Core i5-2400, and the resolution was* 1024 × 768.

The performance of ray packet tracing can be improved by applying frustum culling in addition to SIMD ray techniques. Our prototype implementation uses interval arithmetic (IA) for culling nodes, and corner rays for culling triangles, as described in [BWS06]. Unfortunately, neither of these take advantage of AVX. The box test is entirely scalar, and the triangle test does not map well to wide SIMD execution because there are only 4 corner rays.

All in all, the operations that can fully exploit the power of the AVX instruction set are: ray-box intersection, ray-triangle intersection, and shading.

3. Results

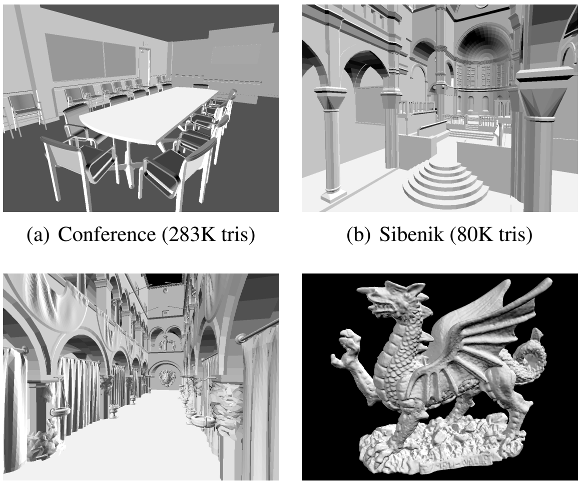
The ray tracing algorithms were implemented in C++ using AVX and SSE4.1 intrinsic functions. The code was compiled for 64 bits with Visual C++ 2010. All tests were run on a system with an Intel Core i5-2400 processor (4 cores, 4 threads, 3.1 GHz) and Windows 7 SP1 64-bit. The rendering resolution was set to 1024 × 768 pixels.

Table 1 shows the performance results for primary rays. It can be seen that for all models except the highly tessellated Welsh Dragon, range traversal with frustum culling is the fastest approach. AVX provides a speedup, compared to SSE, of at least roughly 50% in most cases. This sublinear increase is due to larger SIMD rays with lower utilization and non-SIMD parts of the algorithm.

The Welsh Dragon model contains many small triangles, thus, rays are less coherent. This results in a decreased speedup of about 20%. Partition traversal is better suited for such scenes because it avoids many unnecessary intersections by accurately filtering out dead rays. It is also worth noting that frustum culling has a *negative* impact on the rendering speed for this model.

4. Conclusion and Future Work

We have presented our ideas for accelerating coherent BVH ray tracing using AVX instructions, and have also provided preliminary real-world performance statistics. We will continue our research with a primary focus on incoherent rays.



(a) Conference (283K tris)

(b) Sibenik (80K tris)

(c) Crytek Sponza (279K tris)

(d) Welsh Dragon (2.2M tris)

Figure 1: *Models used for the performance measurements.*

Acknowledgements

This work has been supported by the TeraTomo project of the National Office for Research and Technology, OTKA K-719922, and by TÁMOP-4.2.1/B-09/1/KMR-2010-0002. The Welsh Dragon model was released by Bangor University, UK, for Eurographics 2011.

References

[BWS06] Boulos S., Wald I., Shirley P.: *Geometric and Arithmetic Culling Methods for Entire Ray Packets*. Tech. Rep. UUCS-06-010, School of Computing, University of Utah, 2006. 2

[ORM08] Overbeck R., Ramamoorthi R., Mark W. R.: Large ray packets for real-time whitted ray tracing. In *IEEE/EG Symposium on Interactive Ray Tracing 2008* (2008). 1

[WBS07] Wald I., Boulos S., Shirley P.: Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26*, 1 (2007). 1