Multi Bounding Volume Hierarchies for Ray Tracing Pipelines

Timo Viitanen * Matias Koskela Pekka Jääskeläinen Jarmo Takala

Tampere University of Technology, Finland

Abstract

High-performance ray tracing on CPU is now largely based on Multi Bounding Volume Hierarchy (MBVH) trees. We apply MBVH to a fixed-function ray tracing accelerator architecture. According to cycle-level simulations and power analysis, MBVH reduces energy per frame by an average of 24% and improves performance per area by 19% in scenes with incoherent rays, due to its compact memory layout which reduces DRAM traffic. With primary rays, energy efficiency improves by 15% and performance per area by 20%.

Keywords: ray tracing, ray tracing hardware

Concepts: •**Computing methodologies** → **Ray tracing;** *Graphics processors;* Computer graphics;

1 Introduction

Ray tracing is a fundamental rendering technique which is widely used in offline rendering to model the physical transport of light. Rendering interactive scenes with ray tracing is a longstanding research challenge in computer graphics. In recent years, there has been an influx of research on specialized ray tracing hardware architectures to enable such interactive rendering. Many hardware architectures have been proposed in academic forums, such as RPU [Woop et al. 2005], SGRT [Lee et al. 2013] and RayCore [Nah et al. 2014]. In addition, recently a commercial mobile GPU IP with ray tracing support has been launched. Many of these works are aimed at mobile devices partly since the ray tracing algorithm is well suited for smaller displays, and also because it is likely commercially easier to incorporate a ray tracing feature into a mobile SoC than to sell a stand-alone product. The focus on mobile systems, and recent trends in CMOS process technology place restrictions on ray tracing hardware architectures. Handsets and tablets operate under strict power constraints since they are battery-powered and passively cooled; it is therefore crucial to optimize the hardware accelerator for low-energy operation. Due to logic scaling, the energy cost of computational logic is falling relative to long-range communication. Especially accesses to off-chip SDRAM main memory are expensive.

The basic operation in ray tracing is *ray traversal*, i.e., finding the closest point of the scene geometry which intersects a given half line. The basis of a modern high-performance implementation is to organize the scene geometry into an acceleration datastructure such as a *Bounding Volume Hierarchy* (BVH), which reduces this into a logarithmic-time operation in the typical case. Most of the computational effort in ray tracing goes to traversing this acceleration tree and performing intersection tests against the scene geometry in the leaf nodes. Consequently, ray tracing hardware architectures tend to include fixed-function hardware pipelines for these two tasks.

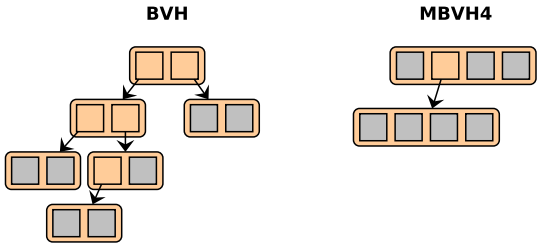

Figure 1: *Left side: BVH organizing 7 leafs (grey) with six 64B nodes (each storing AABBs and pointers of its two children). Right side: 4-wide MBVH organizing the same number of leafs with two 128B nodes ($\frac{2}{3}$ memory footprint).*

In this work, we investigate using *Multi-Bounding Volume Hierarchies* (MBVH) [Ernst and Greiner 2008] [Wald et al. 2008] [Dammertz et al. 2008] in a ray tracing accelerator: this is a variant of BVH with a higher branching factor, typically 4. MBVH was originally intended to take advantage of SIMD instruction sets such as SSE in CPUs, but the technique also has general benefits:

- MBVH has a more compact memory layout than BVH, as noted by Dammertz et al. [Dammertz et al. 2008] and illustrated in Figure 1. Consequently, it improves the hit rate of caches and reduces external memory traffic.
- MBVH ray traversal with a 4-wide MBVH (MBVH4) performs roughly the same amount of computation and memory requests as a BVH, but organized into larger consecutive units. $\frac{n}{2}$ random accesses of $2m$ bytes can be served by a simpler memory hierarchy than n accesses of.m bytes. Likewise, organizing computation into larger units, with fewer branches, reduces the overhead of control logic in the architecture.

Recently, Guthe [2014] found MBVHs advantageous in GPU ray tracing, demonstrating the above advantages. In this paper, we show with simulations and power analysis that introducing MBVH4 to a ray tracing unit significantly improves area- and energy-efficiency over a BVH baseline, especially with incoherent rays.

2 Related Work

Few works on ray tracing hardware architecture have variations on the applied data structures; most of the recent approaches focus on plain BVHs and k-d trees. There is recent interest in quantized BVHs with, e.g., 5 bits per coordinate, most recently by Vaidyanathan [2016]. This structure achieves a very high simulated performance. However, it appears nontrivial to keep updated when rendering dynamic scenes, whereas the present work can use similar update and construction methods as a conventional BVH, with minor modifications.

*e-mail:timo.2.viitanen@tut.fi

SA '16 Technical Briefs, December 05 - 08, 2016, , Macao
ISBN: 978-1-4503-4541-5/16/12
DOI: http://dx.doi.org/10.1145/3005358.3005384

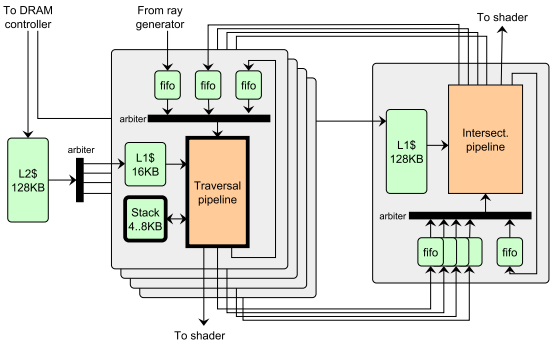

Figure 2: *Ray tracing accelerator architecture modeled in this paper, based on [Lee et al. 2014]. This paper makes changes to the bolded components: the traversal pipeline and the traversal stack.*

Lee et al. [Lee et al. 2014] optimize the BVH data layout in the SGRT system [Lee et al. 2013] for a large performance gain. Hwang et al. [Hwang et al. 2015], optimize the number representations in their data structures to mostly fixed-point-arithmetic, but resort to floating-point numbers in cases where they are more efficient in terms of hardware cost. Our work introduces a hardware architecture based on the MBVH datastructure for major further improvements in area- and energy-efficiency compared to [Lee et al. 2014]. These gains are orthogonal to the approach of [Hwang et al. 2015] and could be combined with their hybrid representation.

3 System Architecture

As a baseline for evaluation, we consider a model architecture based largely on [Lee et al. 2014], which in turn builds on [Lee et al. 2013], shown in Figure 2. The main components of this architecture are *traversal units* (TRV) which handle processing for BVH inner nodes and *intersection units* (IST) which perform ray-triangle intersection tests using Wald's [2004] method. Ray data records enter the system from a shader processor, are assigned a free ray slot by a scheduler hardware, and then pass between TRV and IST units through FIFOs until traversal completes. Each ray is assigned a traversal stack from a specific TRV unit, and all its stack operations are performed by this TRV. This organization is inefficient in some ray states, e.g., upon finishing processing a leaf node, the ray has to return from IST to TRV and pass through the TRV pipeline to perform a stack pop, which may send the ray back to IST. However, the common case of repeated TRV operations is very fast.

The main focus of this work is on the TRV unit. The baseline TRV unit shown in Figure 3 is designed to process BVH nodes in a layout that, for each node, stores the AABBs and pointers of its two children. When a ray enters the TRV, it first attempts to fetch into memory the target node of the ray. It then performs intersection tests against the two child AABBs in parallel using the slabs test. Finally, depending on the results of the tests, the unit performs stack operations, e.g., pushing the pointers of hit children to the stack, and determines the new state of the ray. If both children are hit, they are traversed in a front-to-back order based on the distance values output from the slabs tests.

The rest of the baseline system is configured as follows. 4 TRVs are allocated per IST. Each TRV has storage for 32 stacks, therefore, up to $4 \times 32 = 128$ rays may be active simultaneously. There are two 128KB caches: a node cache shared by the TRVs and a primitive cache serving the IST. In addition, each TRV has a small 16KB L1 node cache. All caches are set as non-blocking,

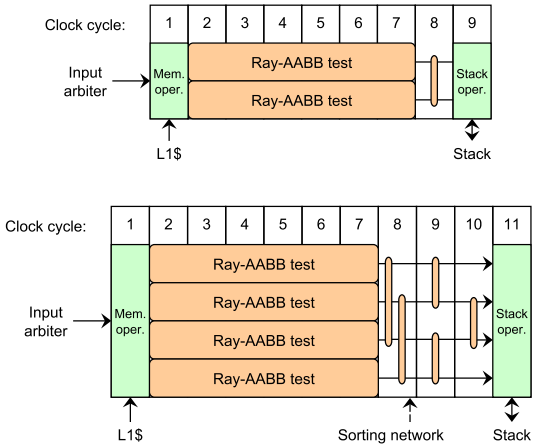

Figure 3: *Top: Baseline TRV unit with two parallel ray-AABB tests. Bottom: Proposed MBVH TRV unit with four ray-AABB tests and sorting network. Pipeline stages are numbered on top.*

4-way set-associative. The node caches use node-sized lines (64B for BVH, 128B for MBVH2), while the primitive L1 is a two-bank interleaved cache [Alvarez et al. 2007] with 64B lines per bank, to accommodate fully pipelined unaligned accesses to 48B primitives. The sizes of FIFOs are determined empirically so as to prevent deadlocks: they are first sized at 32 elements and simulated, and underutilized FIFOs are shrunk to the first power-of-two above the maximum population encountered in simulation. As a latency hiding mechanism, we use "looping for next chance" from RayCore [Nah et al. 2014]: If a memory access misses, the corresponding ray continues through the pipeline, but its subsequent computations are invalidated, and the ray is rescheduled for later execution.

We also diverge from [Lee et al. 2013] by storing full stacks on-chip instead of short stacks. As a 64-entry stack was sufficient to render all evaluated scenes by a large margin - at worst path tracing in *Hairball* used 28 entries - and only accounts for 5..10% of the area and power of the architecture. It appears that short-stack methods are more interesting for architectures with many GPU-like slow threads, or rendering algorithms that require large stack entries such as [Vaidyanathan et al. 2016].

3.1 Proposed MBVH architecture

MBVHs are structured similarly to the BVH layout in [Lee et al. 2014], except there is space for more than two children per node. In this study, we redesign the TRV unit to handle 4-wide MBVHs as shown in Figure 3. It is straightforward to increase the number of box test units to 4, and double the cache line and read port sizes in the node cache hierarchy, keeping the cache capacities constant.

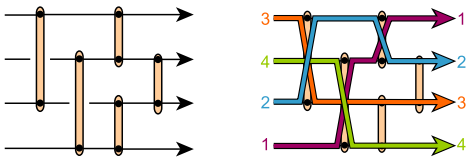

Figure 4: *Left side: Magnification of the sorting network shown in Figure 3 with 5 comparators. Right side: Example sort.*

The main new complication introduced with MBVH is that multiple child nodes may be intersected and pushed to the stack per visit: for a 4-wide MBVH, four children may be hit per TRV processing. As the cache top is kept in the ready state record, up to 3 entries may be inserted into the stack memory. This may be implemented by means of a multi-bank memory, as consecutive entries are guaranteed to be on separate banks. Moreover, it is desirable to traverse BVH nodes in a closest-first order, i.e., the references inserted to the stack should be sorted according to their distance from the camera. In BVH, this is accomplished with a simple compare-swap of the two children, but for high branching factors a sorting operation is necessary. Small sets of numbers can be easily sorted in hardware by means of *sorting networks*, where the array is passed through a series of comparators as shown in Figure 4. Knuth [Knuth 1999] gives depth-optimal networks for up to 16 inputs. For the case of four inputs, an optimal network consists of five comparators, with a *depth* of 3, i.e., the data passes through at most 3 sequential compare-swaps. We pessimistically allocate one pipeline stage per sorting network layer as shown in Figure 3, resulting in 2 extra cycles of latency compared to a BVH TRV. For occlusion rays this step might be bypassed, reducing the TRV latency. A sorting network structure has been used in software by, e.g., Guthe [Guthe 2014] to avoid branches.

Ernst and Greiner [Ernst and Greiner 2008] recommend storing the distance value of each intersected node in the stack, so that after finding a triangle intersection closer than the stored nodes, they can be rejected without testing their children. In initial testing, this technique showed significant performance gains of ca. 10% for MBVH, but only 3% for BVH. The effect is mainly due to triangle intersections avoided by the distance test. As shown in Figure 2, we equip the MBVH TRV with a distance stack, in effect doubling the stack size, while keeping the BVH in the original configuration. MBVH is typically implemented with a power-of-two branching factor to take advantage of SIMD instructions. In custom hardware, it is interesting to use other factors such as 3 or 5. The main complication of odd branching factors is memory hierarchy design: the cache hierarchy should be able to supply one node per cycle to the TRV unit, though they are unaligned in memory. Two approaches are apparent. Firsly, the node L1 may be implemented as a two-bank interleaved cache, as we do with the primitive L1. Secondly, the node cache hierarchy might be addressed with array indices rather than byte addresses, and use a node-sized data word. We experimented with MBVH3, MBVH5 and MBVH6 using both techniques, but these were sightly less efficient than MBVH4.

3.2 Evaluation

In order to evaluate the performance impact of MBVH, we implemented a cycle-level simulator for the baseline and proposed architectures. Though this paper focuses on the TRV pipeline component, the full system including the cache hierarchy and memory needs to be simulated to determine the performance effects. The simulator is split into two parts:A software ray tracer draws scenes and generates, for each ray, logs of node visits and stack operations in a compact binary format, which are then fed to an architecture simulator. The simulator models the cycle-level behavior of the components in Figure 2, including the cache hierarchy, traversal and intersection units, and the interconnection FIFOs and arbiters. The assumed clock rate is 500MHz. The main memory is modeled with Ramulator [Kim et al. 2016]. We assume a LPDDR3-1600 memory with two 32-bit channels, for a peak theoretical data rate of 12.8GB/s.

The area of each architecture is coarsely estimated by counting the number of floating-point units and memory blocks, including caches, stacks and FIFOs: we assume that e.g. control logic, clock trees and pipeline registers add a similar margin to both configurations. The simulator keeps track of activity rates for each components: when idle, they are assumed to be clock gated and contribute only static power. Dynamic and static power figures for SRAMs are obtained from CACTI 6.5 [Muralimanohar et al. 2009], at 45nm. Note that *sequential* caches are used, which first access the tag array before reading data. In the *normal* configuration, the cache reads all words in the target set simultaneously with the tag, but due to the wide read ports in this work, this produces very large caches. For FPUs we use the energy and area per FLOP figures of Galal and Horowitz [2011]. The IST division is implemented with 11 FLOPs as in the algorithm by Markstein [Markstein 2004]. DRAM power figures are produced with DRAMPower [Chandrasekar et al. 2012].

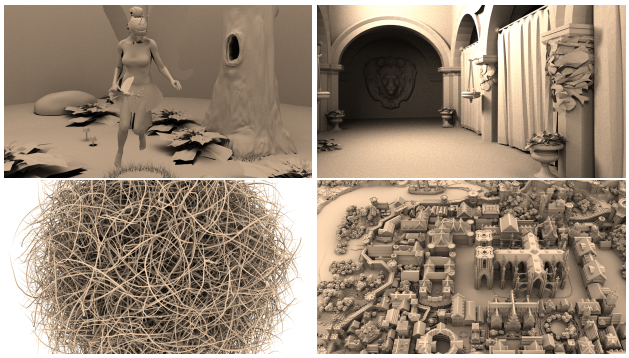

Figure 5: *Test scenes used in simulation: Fairy (179K tri.), Crytek Sponza (262K tri.), Hairball (2.9M tri.) and Rungholt (6.7M tri).*

Four test scenes (Figure 5) were rendered at a 1280x720 resolution with primary rays, as well as diffuse path tracing limited to four bounces and one sample per pixel. Secondary rays are fed to the processor when the preceding ray is complete. BVH trees are constructed with a binned SAH sweep, and MBVHs with Wald's top-down recursive splitting algorithm [Wald et al. 2008]. The latter is chosen since it appears straightforward to implement in hardware by adding extra bookkeeping to the builder unit by Doyle [2013]. Simulation results are shown in Table 2. Since a speedup is expected due to the added hardware resources in MBVH, we use size-independent figures of merit: memory traffic, energy per frame, and performance per area. Area and power breakdown for the *Fairy* scene are shown in Table 1. MBVH4 gives significant improvements in energy and area efficiency across different scenes and ray types. With diffuse rays, the improvement is driven by more efficient use of external memory. DRAM accounts for an average of 70% of energy. In addition to reducing memory traffic, the larger refill increment in MBVH utilizes DRAM better than the baseline. With primary rays, DRAM is insignificant. The efficiency gains from MBVH are smaller and more inconsistent, depending on the ratio of AABB to triangle tests: MBVH performs well in *Rungholt* and poorly in *Hairball*, extreme examples of box-heavy and triangle-heavy scenes, respectively.

3.3 Conclusion

This paper proposed to use MBVH in fixed-function ray tracing accelerators and discussed implementation techniques. 4-wide MBVH improved energy per frame by an average of 24% and performance per area by 19% with incoherent rays: therefore, it appears to be a significant low-hanging fruit in ray tracing hardware design. As future work, we are interested in whether the improvements from MBVH are cumulative with other memory-conserving techniques such as quantized trees and treelet scheduling.

Scene	Ray type	BVH				MBVH4			
		Perf. (MRPS)	Energy (mJ/frame)	Perf. / A (MRPS/mm^2)	DRAM traffic (MB)	Perf. (MRPS)	Energy (mJ/frame)	Perf. / A (MRPS/mm^2)	DRAM traffic (MB)
Fairy	primary	47	9.3	12.2	8	76 (+61%)	8.4 (-10%)	13.8 (+13%)	6 (-19%)
	diffuse	36	84	9.3	644	60 (+67%)	67 (-20%)	11.0 (+17%)	497 (-23%)
Crytek	primary	25	18	6.5	5	47 (+87%)	15 (-15%)	8.5 (+31%)	3 (-42%)
	diffuse	11	417	2.8	3332	19 (+78%)	307 (-26%)	3.5 (+25%)	2345(-30%)
Hairball	primary	14	29	3.7	80	19 (+33%)	24 (-17%)	**3.4 (-6%)**	36 (-55%)
	diffuse	6	490	1.5	4370	10 (+67%)	341 (-31%)	1.8 (+17%))	2933 (-33%)
Rungholt	primary	43	9.1	11.4	3	88 (+100%)	7.5 (-18%)	16.1 (+41%)	3 (-18%)
	diffuse	29	92	7.6	703	48 (+66%)	76 (-17%)	8.8 (+16%)	582 (-17%)
Mean	primary	-	-	-	-	+70%	-15%	+20%	-33%
	diffuse	-	-	-	-	+69%	-24%	+19%	-26%

Table 2: *Comparison of BVH and MBVH4 accelerators. MBVH4 is consistently more efficient except for primary rays in Hairball, where IST is a bottleneck. With incoherent rays, DRAM dominates energy consumption.*

Unit	TRV	IST	stack	cache	fifo	dram	$\sum$
BVH							
Area (mm^2)	0.25	0.20	0.25	1.57	1.25	-	3.51
P. power (mW)	237	74	13	90	23	37	475
D. power (mW)	157	60	10	130	25	608	990
MBVH4							
Area (mm^2)	0.41	0.20	0.51	2.35	1.25	-	5.47
P. power (mW)	413	88	22	101	22	42	689
D. power (mW)	285	71	18	153	24	767	1319

Table 1: *Area estimate for baseline (BVH) and proposed (MBVH), and power breakdown on Fairy, primary and diffuse rays. Caches have the same capacity in MBVH but take up more area and power due to the wider read port. The DRAM is a clear bottleneck for incoherent rays.*

Acknowledgement

The authors would like to thank Finnish Funding Agency for Technology and Innovation (project Parallel Acceleration 3, funding decision 1134/31/2015) and European Commission in the context of ARTEMIS project ALMARVI (ARTEMIS 2013 GA 621439)", as well as the TUT graduate school and the Nokia Foundation. Models used are courtesy of Ingo Wald (Fairy), Frank Meinl (Crytek Sponza), Samuli Laine (Hairball) and kescha (Rungholt).

References

ALVAREZ, M., SALAMI, E., RAMIREZ, A., AND VALERO, M. 2007. Performance impact of unaligned memory operations in SIMD extensions for video codec applications. In *IEEE Int. Symp. Performance Analysis of Systems Software*, 62–71.

CHANDRASEKAR, K., WEIS, C., LI, Y., AKESSON, B., WEHN, N., AND GOOSSENS, K., 2012. DRAMPower: Open-source DRAM power & energy estimation tool. http://www.drampower.info.

DAMMERTZ, H., HANIKA, J., AND KELLER, A. 2008. Shallow bounding volume hierarchies for fast SIMD ray tracing of incoherent rays. *Comput. Graph. Forum 27*, 4, 1225–1233.

DOYLE, M. J., FOWLER, C., AND MANZKE, M. 2013. A hardware unit for fast SAH-optimised BVH construction. *ACM Trans. Graph. 32*, 4, 139.

ERNST, M., AND GREINER, G. 2008. Multi bounding volume hierarchies. In *IEEE Symp. Interactive Ray Tracing*, 35–40.

GALAL, S., AND HOROWITZ, M. 2011. Energy-efficient floating-point unit design. *IEEE Trans. Comp. 60*, 7, 913–922.

GUTHE, M. 2014. Latency considerations of depth-first GPU ray tracing. In *Eurographics (Short Papers)*, 53–56.

HWANG, S. J., LEE, J., SHIN, Y., LEE, W.-J., AND RYU, S. 2015. A mobile ray tracing engine with hybrid number representations. In *SIGGRAPH Asia Mobile Graph. Interact. Appl.*, 3.

KIM, Y., YANG, W., AND MUTLU, O. 2016. Ramulator: A fast and extensible DRAM simulator. *IEEE Comp. Arch. Letters 15*, 1 (Jan), 45–49.

KNUTH, D. E. 1999. *The Art of Computer Programming: Volume 3: Sorting and Searching*, vol. 3.

LEE, W., SHIN, Y., LEE, J., KIM, J., NAH, J., JUNG, S., LEE, S., PARK, H., AND HAN, T. 2013. SGRT: A mobile GPU architecture for real-time ray tracing. In *Proc. High-Performance Graph.*, 109–119.

LEE, J., LEE, W.-J., SHIN, Y., HWANG, S., RYU, S., AND KIM, J. 2014. Two-AABB traversal for mobile real-time ray tracing. In *SIGGRAPH Asia Mobile Graph. Interact. Appl.*, 14.

MARKSTEIN, P. 2004. Software division and square root using Goldschmidt's algorithms. In *Proc. Conf. Real Numbers and Comp.*, vol. 123, 146–157.

MURALIMANOHAR, N., BALASUBRAMONIAN, R., AND JOUPPI, N. P. 2009. CACTI 6.0: A tool to model large caches. *HP Laboratories*, 22–31.

NAH, J.-H., KWON, H.-J., KIM, D.-S., JEONG, C.-H., PARK, J., HAN, T.-D., MANOCHA, D., AND PARK, W.-C. 2014. RayCore: A ray-tracing hardware architecture for mobile devices. *ACM Trans. Graph. 33*, 5, 162.

VAIDYANATHAN, K., AKENINE-MÖLLER, T., AND SALVI, M. 2016. Watertight ray traversal with reduced precision. In *Proc. High-Performance Graph-*, Eurographics Association, 33–40.

WALD, I., BENTHIN, C., AND BOULOS, S. 2008. Getting rid of packets - efficient SIMD single-ray traversal using multi-branching BVHs. In *IEEE Symp. Interact. Ray Tracing*, 49–57.

WALD, I. 2004. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Saarland University, Germany.

WOOP, S., SCHMITTLER, J., AND SLUSALLEK, P. 2005. RPU: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph. 24*, 3, 434–444.