Multi Bounding Volume Hierarchies

Manfred Ernst*
Bytes+Lights GmbH

Günther Greiner†
Universität Erlangen-Nürnberg

Figure 1: Example images (500^2 pixels) rendered with a Monte Carlo ray tracer using 256 primary rays per pixel and a maximum trace depth of 20. Absolute rendering times are shown for the multi bounding volume hierarchy. The speed-up over a regular bounding volume hierarchy is given in parentheses. Scenes with complex geometry benefit more, because ray shooting dominates the total rendering time. *Bunny*: 69,451 triangles, 128 million rays, 11 min (1.19×). *Exterior mirror*: 334,080 triangles, 167 million rays, 15 min (1.32×). *Headlight*: 770,534 triangles, 1 billion rays, 92 min (1.64×). *Car*: 4,304,391 triangles, 200 million rays, 55 min (2.28×).

Abstract

Efficient tracing of single and incoherent rays is still a challenge in computer graphics. Coherent packet tracing has reached real-time performance, but ray packets bring about restrictions for the architecture of the renderer and their suitability for diverging secondary ray bundles is uncertain. The algorithm presented in this paper avoids these problems by not using ray packets at all. Instead, it uses triangle packets and bounding volume packets in a novel acceleration data structure called *multi bounding volume hierarchy* (MBVH). It is designed for SIMD single ray tracing. The hierarchy is built from a binary bounding volume hierarchy by collapsing subtrees of height two into SIMD nodes storing four bounding boxes. A modified cost function for construction guarantees that all but one of the leaf nodes contain exactly four triangles. The MBVH makes good use of data-level parallelism during traversal and triangle intersection, yielding speed-ups of up to 2.8× for random ray shooting. It consumes less memory than a regular bounding volume hierarchy and requires no modifications to the architecture of the rendering engine.

Index Terms: I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types; I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1 Introduction

Despite the remarkable advances in real-time packet tracing, most commercial rendering engines are still based on single ray architectures. This is because most rays in photorealistic rendering are incoherent and cannot be easily arranged into packets. Further, packet-based ray tracing has a severe impact on the design of the whole renderer and limits its flexibility. The resulting restrictions on global illumination algorithms and the more difficult source code maintenance make it less attractive for a production renderer. On the other hand, packet traversal is the only known method that exploits data-level parallelism well on current hardware.

Fortunately, rays are not the only entity that can be bundled into SIMD data structures. Though less obvious, it is equally valid to group multiple nodes of the acceleration structure and multiple triangles in the leaf nodes and intersect them with one ray. The resulting algorithm is not as fast as a packet tracer, but it is more flexible and much easier to integrate into existing rendering software.

Ray shooting with our novel multi bounding volume hierarchy consists of three stages:

1. Building a binary bounding volume hierarchy (BVH) with four triangle references in each leaf node.

2. Converting the BVH to a multi bounding volume hierarchy (MBVH) by collapsing subtrees of height two into nodes with four bounding boxes.

3. Traversing and intersecting the hierarchy with SIMD operations, fetching triangle packets from a cache.

An important difference to other acceleration structures is that an MBVH node does not store its own bounding box, but the bounding boxes of its four children. During traversal, the ray is intersected with all of them in parallel. Traversal continues with the closest child and the other nodes are pushed onto a stack in back to front order. If some of the children are leaf nodes, their referenced triangles are intersected immediately. Triangle packets are assembled to SIMD data structures on the fly and cached for reuse.

Our main contribution is a fast and clean ray shooting algorithm, suitable for integration in any rendering engine. It performs well with completely incoherent rays in arbitrary scenes and does not consume more memory than traditional approaches. The algorithm is generally applicable and would benefit interactive ray tracers just as much as the offline ray tracers used in this work.

*e-mail: manfred.ernst@bytes-lights.de
†e-mail: greiner@informatik.uni-erlangen.de

IEEE/EG Symposium on Interactive Ray Tracing 2008
9 - 10 August, Los Angeles, California, USA
978-1-4244-2741-3/08/$25.00 ©2008 IEEE

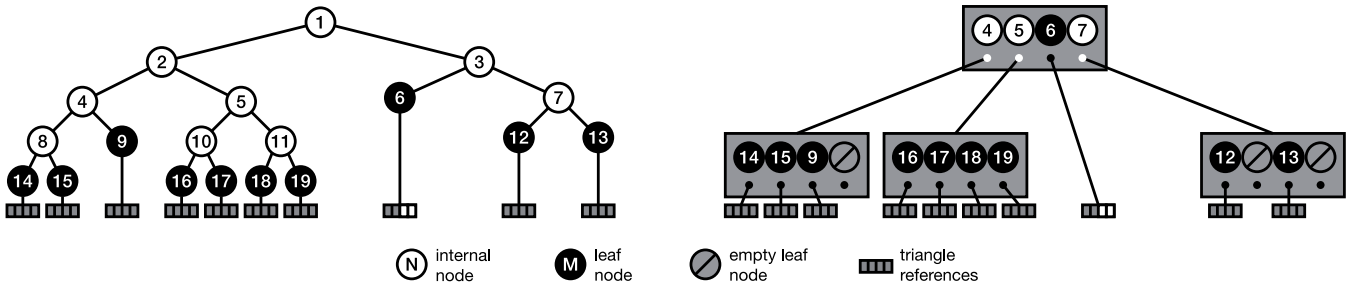

Figure 2: Converting a bounding volume hierarchy (left) to a multi bounding volume hierarchy (right). Subtrees of height two are collapsed into one MBVH node. Incomplete subtrees are filled with empty leaves (gray nodes with a diagonal bar). Note that an MBVH node may contain a mix of leaves and internal nodes.

2 Background

Ray tracing was traditionally used to compute photorealistic images of complex scenes which were impossible to render with graphics hardware. The basic visibility algorithm, as introduced by Appel [1], was later extended with reflections and refractions [45]. Distribution ray tracing [7] made rendering of non-singular phenomena like soft shadows and glossy reflections possible. Kajiya presented path tracing as the first general solution to the rendering equation [19]. Metropolis light transport [38] is another example of an unbiased global illumination algorithm. In the past years, many techniques have been developed that generate incredibly realistic images, including photon mapping [17], image based lighting [9] and subsurface scattering [18]. All of them have one thing in common: they have to trace an enormous amount of incoherent rays.

Efficient acceleration data structures for ray tracing have always been an active area of research. Most of modern rendering engines either use grids [44], kd-trees [2][20], bounding volume hierarchies [5][36], or one of their derivatives [39]. Havran's PhD thesis contains a comparison of the various methods [14]. An automatic building algorithm for bounding volume hierarchies was presented in [21]. Goldsmith and Salmon used a cost model to compute the best split for each node [12]. It is now known as the surface area heuristic (SAH) and was also adopted for kd-trees [25]. More recently, fast construction algorithms were described for kd-trees [43][15][32], bounding volume hierarchies [41][16] and bounding interval hierarchies [39]. Early split clipping [10] was developed to build better BVHS for scenes containing large diagonal triangles.

The basic traversal algorithm for bounding volume hierarchies has also received many improvements over the past years. Early exits for shadow rays were proposed by Smits [37]. Haines described a series of optimizations [13], most notably the culling of boxes that are further away than the current closest intersection. Fast ray-box intersection tests [26] further improved traversal speed. Kay and Kajiya use a heap for depth-ordered traversal of the child nodes [21]. An alternative method is applied by Mahovsky [27]. He stores the splitting axis of internal nodes and decides the traversal order according to the sign of the ray direction.

Packet-based ray tracing on kd-trees [40] was the first method using the SIMD units of modern processors efficiently. It was extended by frustum traversal [35] and applied to dynamic scenes with other acceleration data structures like grids [44] and bounding volume hierarchies [24][42]. All of these techniques work very well for primary rays, but performance quickly degrades for diverging ray bundles. Reshetov improved the traversal speed for secondary rays [33] and he also demonstrated how to cull triangles efficiently in complex scenes [34]. Ray scheduling strategies similar to [31] were recently applied to packet-based ray tracing [28][30] and distribution ray tracing [3] in order to increase coherence.

Wald mentioned the possibility to intersect one ray with multiple triangles using SIMD operations [40], but he did not further investigate this topic. Lauterbach's ray-strips [23] allow for SIMD ray triangle intersection with a single ray. Pixar's RenderMan ray tracing implementation [4] also intersects one ray with all triangles of a grid in parallel. For internal nodes of the bounding volume hierarchy, it applies SIMD to test multiple slabs at once. Details and performance measurements are not given.

Cline et al. used bounding volume hierarchies with a higher arity to reduce the memory footprint [6]. Shallow bounding volume hierarchies for efficient SIMD traversal of single rays were recently presented by Dammertz et al. [8]. Their algorithms and data structures are very similar to our MBVH but were developed independently.

The performance gap between coherent packet tracing and incoherent single ray tracing is widening. This is problematic, because many applications generate more secondary rays than primary rays. Further, packets are difficult to integrate into existing software. The multi bounding volume hierarchy, presented in this paper, is well suited for those applications.

3 Multi Bounding Volume Hierarchies

The MBVH is a hierarchy of axis aligned bounding boxes with a fanout of four. In contrast to a regular BVH, the nodes store information about their children and not about themselves. An internal node of our hierarchy contains the following information:

- The bounding boxes of its four child nodes in one SIMD data structure (96 bytes).
- References to its four children (16 bytes)
- Four boolean flags, indicating whether the respective child is a leaf node (4 bytes)
- The precomputed traversal order for the eight possible octants of ray directions in a compact bitfield (8 bytes)

Some information could be stored in a more compact way using bitwise operations. This is not required because the nodes will be padded to 128 bytes for cache alignment anyway. Leaf nodes are stored differently, because they do not have children. A leaf is simply an array of references to triangles in the scene. The child pointers are type-cast according to the leaf flags. We restrict the number of referenced triangles to a maximum of four. The construction algorithm will ensure that all but one of the leaves are filled completely, yielding optimal SIMD utilization.

3.1 Construction

The most convenient way to build an MBVH is to first construct a regular binary BVH and convert it afterwards. Any building algorithm can be applied in the first stage. Our implementation uses early split clipping and a modified surface area heuristic cost function to account for the SIMD nature of the resulting data structure.

A single pass construction method is an interesting direction for future work.

Good data-level parallelism for ray triangle intersection on current hardware platforms with 4-wide SIMD units can only be achieved when most leaf nodes contain four triangles. The classic SAH cost function C cannot enforce this property. It is defined as:

$$C = C_T + |P_L|\frac{SA(B_L)}{SA(B_P)} + |P_R|\frac{SA(B_R)}{SA(B_P)},$$

where $|P_L|$ and $|P_R|$ is the number of primitives in the left and right child, respectively, $SA(\cdot)$ is the surface area of a box and B_L, B_R and B_P are the bounding boxes of the left child, right child and parent node. C_T is the relative cost of traversing a node to the cost of computing a ray-triangle intersection.

Figure 3: Naming convention for subtree nodes.

Our modified cost function returns ∞ if neither $|P_R|$ nor $|P_L|$ is a multiple of four. In this way, at most one leaf in the whole hierarchy will contain less than four triangles. Splits are forced until each leaf node stores four references or less. This is not mandatory for the MBVH but allows for computing the intersection of a ray with a leaf in a single SIMD test without a loop. Further, the number of references in a leaf need not be stored. The forced splits do not degrade rendering performance for our test scenes.

Each node of the MBVH will correspond to a subtree of height two of the BVH. This mapping is shown in Figure 2. The BVH subtrees contain up to seven nodes, that will be called P, L, R, LL, LR, RL and RR according to Figure 3. If the subtree is complete, the MBVH node stores the bounding boxes in one SIMD data structure in this order: B_{LL}, B_{LR}, B_{RL} and B_{RR}. Incomplete subtrees are simply filled up with empty leaves, represented by empty (inverted) bounding boxes as depicted in Figure 2.

The approximate traversal order is precomputed during construction for each node according to [27]. For about 30 percent of all rays the order is not exact, but it saves a costly sorting operation during traversal. Dammertz et al. [8] use the same order but do not precompute it. The approximate order depends only on the signs of the ray direction vector and the split axes of the original BVH nodes P, L and R. The pseudocode in Algorithm 1 computes the order for a given direction vector.

Algorithm 1: traversalOrder(dir, P, L, R)

```
// dir:  ray direction vector
// P,R,L: nodes of the temporary BVH
Array order = [0, 1, 2, 3];
if L.isInternal() and dir[L.splitAxis()] <= 0 then
   swap(order[0],order[1]);
if R.isInternal() and dir[R.splitAxis()] <= 0 then
   swap(order[2],order[3]);
if P.isInternal() and dir[P.splitAxis()] <= 0 then
   swap(order[0],order[2]);
   swap(order[1],order[3]);
return order;
```

The algorithm swaps LL and LR when L is an internal node and the direction vector has a negative sign in the component along the split axis of L. RL and RR are treated accordingly. Finally, the split axis of P determines if the subtrees below L and R have to be swapped completely. For each of the eight possible octants of ray directions, the order is computed and packed into one byte of the MBVH with bitwise operations.

The hierarchy conversion is performed by a top down traversal of the BVH, starting with the root node. After processing a subtree, the nodes LL, LR, RL and RR are traversed recursively if they exist and if they are internal nodes.

3.2 Ray Traversal

Algorithm 2: closestIntersection(ray, isect)

```
// ray:  scalar ray (input)
// isect:  closest intersection (output)
1  initialize(ray4, isect, stack);
2  push root node onto stack with t = 0;
3  while not stack.empty() do
4     if stack.top().t > isect.t then
5        stack.pop();
6        continue ;
7     node = stack.top().node;
8     stack.pop();
9     hitBox4 = intersectBox4(ray4, node, tBox4);
10    order[] = getTraversalOrder(ray4, node);
11    for c = 0 to 3 do
12       o = order[c];
13       if hitBox4[o] and tBox4[o] < isect.t then
14          if node.isLeaf(o) then
15             intersectTriangle4(ray4, node.leaf[o], isect);
16          else
17             mark child o for pushing;
18    end
19    push marked children and t-values onto stack;
20 end
21 return isect;
```

Traversal is performed by Algorithm 2. Iterative traversal is used because it is more efficient than its recursive equivalent. SIMD data types and functions are marked with a trailing *4*. At the beginning of the routine a temporary SIMD ray is initialized from the scalar ray by duplicating all values four times. It is used for the SIMD computations during traversal and intersection. The intersection data and the stack are also initialized. The stack stores pointers to MBVH nodes that have already been tested for intersection but have not yet been processed. Corresponding parameter values t of the intersections are also stored for a culling test. The root node of the MBVH is pushed onto the stack with $t = 0$ because it may not be culled.

The while loop iterates until the stack is empty. Nodes that are further away than the current closest intersection are culled away in line 6. The ray is then intersected with all four bounding boxes of the next node in SIMD. In line 10, the precomputed traversal order for the child nodes is fetched. The for loop in line 11 iterates over the children in this order. If the bounding box of the child was hit and if it is closer than the closest intersection so far, it will be examined. In case of a leaf node, the intersection with all referenced triangles is computed with one SIMD test. The intersection data structure is updated if any valid intersection was found. Internal child nodes are just marked for pushing onto the stack. In line 19, they are finally pushed in back to front order together with their intersection distance. The algorithm returns the intersection struct.

For shadow rays, child nodes are processed in arbitrary order and traversal is stopped on the first intersection. Entry point caching and backtracking as described in [8] are not used.

Ray-box intersections are computed with a SIMD version of the slabs test [11]. Möller and Trumbore's algorithm [29] is used in

SIMD for ray-triangle intersections. Existing code from a packet-based ray tracer could also be used for this test, e.g. [22].

3.3 Triangle Cache

Packets of four triangles have to be stored together for SIMD intersection. This would be easy if each leaf stored the geometry of its triangles directly. Unfortunately, this requires a lot of extra memory, because the vertices cannot be shared among the triangles. Early split clipping can generate multiple references to one triangle and would further increase the memory consumption. Consequently, the leaves will only store references to the triangles.

Assembling the SIMD friendly data structures is an expensive operation. Additionally, our triangles are stored in a scenegraph and have to be transformed to world space on the fly. Efficiency is regained by a triangle cache. The triangles are transformed and packed into 4-element SIMD vectors on demand and stored in the cache, as the ray traversal executes. This avoids the need to preprocess and store all of the triangle data beforehand. The cache is direct-mapped with the IDs of the leaf nodes.

4 Results

We have integrated multi bounding volume hierarchies into a photorealistic rendering engine. The system is fairly well optimized, but it is primarily designed for flexibility, quality and robustness. Example renderings are shown in Figure 1. The speed-up of 1.19× to 2.28× is comparable to the results reported by Dammertz et al. [8].

In addition to the rendering tests, a simple benchmark was implemented to measure the performance of the MBVH. It reads a triangulated scene and generates 10 million random rays. The start and end points of each ray are chosen randomly inside the bounding box of the scene. In this way, the rays are as incoherent as possible. The C++ code was compiled with ICC 10.1 and optimized for the test system, a Core2 Duo with 2GHz and 4GB of main memory. SSE intrinsics were wrapped into classes with inline functions. Only one core was used for all tests.

The scenes shown in Figure 1 were also chosen for the benchmark: the Stanford bunny (69,451 triangles), a car mirror (334,080 triangles), the headlight of a car (770,534 triangles) and a complete car (4,304,391 triangles). The latter three CAD models are particularly hard for any ray tracer. The geometry is highly tesselated and has an enormous depth complexity. Additionally, there are almost no large axis aligned triangles like in architectural scenes. This results in low traversal and intersection coherence even for primary rays. Multi bounding volume hierarchies also perform well with simpler geometry, but this is not our primary focus.

We compared the MBVH to a regular BVH. The statistics for hierarchy construction are presented in Table 1. The regular bounding volume hierarchies in the lines labeled with BVH were built with a maximum leaf size of four and the classic SAH cost function. For the MBVH, the modified cost function was used. The MBVH is shallower and consumes less memory, because one of its 128-byte nodes represents seven 32-byte nodes of a regular BVH. It usually stores, however, some empty leaves. The time to convert the temporary BVH to the MBVH is negligible. A future implementation will skip this step completely and build the MBVH directly.

The hierarchies from Table 1 were then used to trace random rays as described above. Results are shown in Table 2. All tests were performed with shadow rays and regular rays. Shadow rays can be terminated as soon as any intersection occurs, while regular rays must continue traversal until the closest intersection is found. Performance for regular rays is consistently faster with the MBVH by a factor of 1.67 to 2.61. Shadow rays benefit even more from SIMD traversal and intersection: the speed-up ranges from 2.25 to 2.84. The number of ray-box intersections is reduced to one fourth by the MBVH, because one test intersects a ray with four boxes in SIMD. Ray-triangle intersections are also reduced, but not so significantly. Considering the complexity of our test scenes, the performance of multi bounding volume hierarchies is also comparable to the results reported by Boulos [3].

	T_B	T_C	N_L	N_E	$\bar{L}$	$\bar{D}$	M
Bunny							
BVH	0.4	-	21	-	3.3	14.6	1.6
MBVH	0.4	0.01	17	8	4.0	6.5	1.3
Mirror							
BVH	2.7	-	116	-	3.0	18.4	8.4
MBVH	2.4	0.04	87	43	4.0	8.3	6.6
Headlight							
BVH	6.3	-	250	-	3.1	18.9	18.2
MBVH	5.8	0.09	193	96	4.0	8.5	14.7
Car							
BVH	43.4	-	1,401	-	3.1	22.4	102.0
MBVH	39.2	0.48	1,076	539	4.0	10.3	82.1

Table 1: BVH and MBVH statistics. T_B: build time for regular BVH in seconds. T_C: time for conversion to MBVH in seconds. N_L: number of non-empty leaf nodes (in thousands). N_E: number of empty leaf nodes (in thousands). $\bar{L}$: mean leaf size. $\bar{D}$: average depth. M: memory consumption in mega bytes.

	$N_\triangle$	N_{box}	H	$krps$	S
Bunny					
BVH*	3.6	35.9	-	364	-
MBVH*	1.5	9.7	99.9	1034	2.84
BVH	5.0	42.9	-	295	-
MBVH	1.7	11.0	99.9	769	2.61
Mirror					
BVH*	5.4	44.8	-	286	-
MBVH*	2.2	12.6	99.7	721	2.52
BVH	8.4	58.4	-	205	-
MBVH	3.6	16.0	99.8	423	2.06
Headlight					
BVH*	5.4	48.5	-	251	-
MBVH*	2.4	14.5	99.4	564	2.25
BVH	8.1	64.4	-	183	-
MBVH	3.8	19.2	99.6	306	1.67
Car					
BVH*	5.2	48.8	-	239	-
MBVH*	2.0	13.3	96.8	571	2.39
BVH	7.9	64.5	-	170	-
MBVH	3.3	18.2	97.3	333	1.96

Table 2: Random ray shooting statistics. Shadow ray tests are marked with an * after the hierarchy type. $N_\triangle$: average number of ray-triangle intersections (4-way SIMD for the MBVH). N_{box}: average number of ray-box intersections (4-way SIMD for the MBVH). H: cache hit rate in percent. $krps$: kilo rays per second. S: speed-up over regular BVH.

A triangle cache of 100 MB was used. The smaller scenes fit completely into the cache and even the largest model yields a hit rate of 97 percent. The size of the cache influences, however, the performance as seen in Table 3. A cache for world space vertices was used for the regular BVH, but it has almost no impact on performance.

We have also compared multi bounding volume hierarchies to ray packet tracing. The plots in Figure 4 show ray shooting performance for an increasing degree of incoherence. For this test, random start and end points were generated inside the scene bounding box for the ray packets. Both points were jittered for the individual rays of each packet. The jittering region was set to the scene bounding box, scaled down by the incoherence factor $I \in [0,1]$. Identical

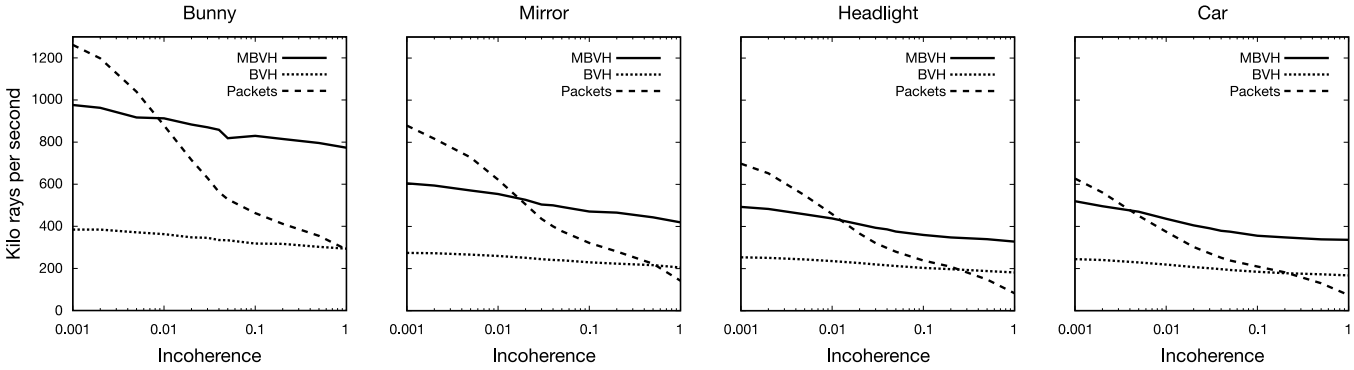

Figure 4: Comparison of regular bounding volume hierarchies (BVH), multi bounding volume hierarchies (MBVH) and ray packet tracing (Packets). Performance of ray packets quickly degrades below MBVH and even below the regular BVH for random rays. Logarithmic scale was used for the x-axis because the performance drop-off is very steep in a narrow region.

	200 MB	100 MB	50 MB	25 MB	10 MB
H	98.3	97.3	90.2	80.2	64.6
$krps$	338	333	283	246	216

Table 3: Hit rate and rays per second for different sizes of the triangle cache, measured with MBVH and regular rays in the car scene. H: cache hit rate in percent. $krps$: kilo rays per second.

bounding volume hierarchies were used for single-ray tracing and packet tracing. As expected, performance of ray packets is excellent for coherent rays. However, it quickly degrades below the MBVH and even below the single-ray BVH for random rays.

In order to estimate the suitability for future hardware architectures with 8-wide and 16-wide SIMD units, we implemented a scalar simulation of multi bounding volume hierarchies with fanouts of 4, 8 and 16. For this purpose, the data structure and the algorithm were generalized to arbitrary fanouts and implemented as a C++ template *without* SSE instructions. The results are shown in Table 4. It can be estimated that hardware architectures with 4-, 8- and 16-wide SIMD units will execute the corresponding multi bounding volume hierarchy traversal up to 3.0, 4.5 and 6.7 times faster than a regular BVH traversal. Results of our native 4-wide SIMD implementation with SSE confirm this theoretical result. Although the performance gain is sublinear, wider SIMD units will clearly be beneficial for multi bounding volume hierarchies.

	binary	fanout-4	fanout-8	fanout-16
Bunny	295	0.75 (3.00)	0.57 (4.56)	0.43 (6.88)
Mirror	205	0.75 (3.00)	0.57 (4.56)	0.43 (6.88)
Headlight	183	0.73 (2.92)	0.56 (4.48)	0.40 (6.40)

Table 4: Performance of scalar simulations of multi bounding volume hierarchies (without SSE operations). The column *binary* gives the performance of a regular BVH in kilo rays per second. The remaining columns show the relative performance of simulated multi bounding volume hierarchies with a fanout of 4, 8 and 16. Potential speed-ups for native n-wide SIMD implementations are given in parentheses.

5 Conclusion

We have presented the multi bounding volume hierarchy, a highly efficient acceleration data structure for ray tracing. It is a SIMD representation of a regular bounding volume hierarchy. A modified cost function guarantees that all but one of the leaves are filled with four triangles for optimal SIMD utilization. In combination with a triangle packet cache, the MBVH makes efficient tracing of single and incoherent rays possible. It is consistently faster than traditional single ray tracing and consumes less memory. The algorithm is easily integrated into existing rendering software and its superior performance has been demonstrated with real world data sets. Multi bounding volume hierarchies may be extended to support ray packet traversal for coherent rays as well.

6 Acknowledgements

We would like to thank the Stanford Scanning Repository for the Bunny model and BMW for the three CAD datasets. Also, we would like to thank the anonymous reviewers for their constructive feedbacks and suggestions.

References

[1] A. Appel. Some techniques for shading machine renderings of solids. In *Proceedings of the Spring Joint Computer Conference*, pages 37–45, 1968.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, 1975.

[3] S. Boulos, D. Edwards, J. D. Lacewell, J. Kniss, J. Kautz, P. Shirley, and I. Wald. Packet-based Whitted and Distribution Ray Tracing. In *Proc. Graphics Interface*, May 2007.

[4] P. H. Christensen, J. Fong, D. M. Laur, and D. Batali. Ray Tracing for the Movie 'Cars'. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 1–6, 2006.

[5] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, 1976.

[6] D. Cline, K. Steele, and P. Egbert. Lightweight bounding volumes for ray tracing. *journal of graphics tools*, 11(4):61–71, 2006.

[7] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, 1984.

[8] H. Dammertz, J. Hanika, and A. Keller. Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. In *Rendering Techniques 2008, Proceedings of the Eurographics Symposium on Rendering*, 2008. to appear.

[9] P. Debevec. Rendering synthetic objects into real scenes. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 189–198, New York, NY, USA, 1998. ACM.

[10] M. Ernst and G. Greiner. Early split clipping for bounding volume hierarchies. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 73–78, Ulm, Germany, September 2007.

[11] M. Geimer and S. Müller. A cross-platform framework for interactive ray tracing. In *Graphiktag im Rahmen der GI Jahrestagung*, Frankfurt am Main, Germany, 2003.

[12] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

[13] E. Haines. *Graphics Gems II*, chapter Efficiency Improvements for Hierarchy Traversal, pages 267–273. Academic Press, San Diego, CA, USA, 1991.

[14] V. Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[15] W. Hunt, W. R. Mark, and G. Stoll. Fast kd-tree construction with an adaptive error-bounded heuristic. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 81–88, 2006.

[16] T. Ize, I. Wald, and S. G. Parker. Asynchronous BVH Construction for Ray Tracing Dynamic Scenes on Parallel Multi-Core Architectures. In *Proceedings of the 2007 Eurographics Symposium on Parallel Graphics and Visualization*, 2007.

[17] H. W. Jensen. *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[18] H. W. Jensen, S. R. Marschner, M. Levoy, and P. Hanrahan. A practical model for subsurface light transport. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 511–518, New York, NY, USA, 2001. ACM.

[19] J. T. Kajiya. The rendering equation. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 143–150, New York, NY, USA, 1986. ACM.

[20] M. Kaplan. The use of spatial coherence in ray tracing. In *ACM SIGGRAPH '85 Course Notes 11*, pages 22–26, July 1985.

[21] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques*, pages 269–278, New York, NY, USA, 1986. ACM Press.

[22] A. Kensler and P. Shirley. Optimizing ray-triangle intersection via automated search. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 33–38, 2006.

[23] C. Lauterbach, S.-E. Yoon, and D. Manocha. Ray-strips: A compact mesh representation for interactive ray tracing. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 19–26, Ulm, Germany, September 2007.

[24] C. Lauterbach, S.-E. Yoon, D. Tuft, and D. Manocha. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 39–46, 2006.

[25] D. J. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 6(3):153–166, 1990.

[26] J. Mahovsky and B. Wyvill. Fast Ray-Axis Aligned Bounding Box Overlap Tests With Plucker Coordinates . *The Journal of Graphics Tools*, 9(1):37–48, 2004.

[27] J. A. Mahovsky. *Ray tracing with reduced-precision bounding volume hierarchies*. PhD thesis, Calgary, Canada, 2005.

[28] E. Mansson, J. Munkberg, and T. Akenine-Möller. Deep coherent ray tracing. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 79–85, Ulm, Germany, September 2007.

[29] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *journal of graphics tools*, 2(1):21–28, 1997.

[30] P. Navrátil, D. Fussella, C. Lin, and W. Mark. Dynamic ray scheduling to improve ray coherence and bandwidth utilization. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 95–104, Ulm, Germany, September 2007.

[31] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 101–108, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[32] S. Popov, J. Günther, H.-P. Seidel, and P. Slusalek. Experiences with streaming construction of sah kd-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 89–94, 2006.

[33] A. Reshetov. Omnidirectional ray tracing traversal algorithm for kd-trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 57–60, 2006.

[34] A. Reshetov. Faster ray packets - triangle intersection through vertex culling. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 105–112, Ulm, Germany, September 2007.

[35] A. Reshetov, A. Soupikov, and J. Hurley. Multi-level ray tracing algorithm. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 1176–1185, New York, NY, USA, 2005. ACM.

[36] S. M. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pages 110–116, New York, NY, USA, 1980. ACM Press.

[37] B. Smits. Efficiency issues for ray tracing. *Journal of Graphics Tools: JGT*, 3(2):1–14, 1998.

[38] E. Veach and L. J. Guibas. Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[39] C. Wächter and A. Keller. Instant ray tracing: The bounding interval hierarchy. In *Rendering Techniques 2006, Proceedings of the Eurographics Symposium on Rendering*, pages 139–149, Aire-la-ille, Swizerland, June 2006. The Eurographics Association.

[40] I. Wald. *Realtime Ray Tracing and Interactive Global Illumination*. PhD thesis, Computer Graphics Group, Saarland University, 2004.

[41] I. Wald. On fast Construction of SAH based Bounding Volume Hierarchies. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*, pages 33–40, Ulm, Germany, September 2007.

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

[43] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 61–69, 2006.

[44] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray tracing animated scenes using coherent grid traversal. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 485–493, New York, NY, USA, 2006. ACM.

[45] T. Whitted. An improved illumination model for shaded display. *Commun. ACM*, 23(6):343–349, 1980.