Early Split Clipping for Bounding Volume Hierarchies

Manfred Ernst* Günther Greiner†

Lehrstuhl für Graphische Datenverarbeitung
Universität Erlangen-Nürnberg, Germany

ABSTRACT

Despite their algorithmic elegance and robustness, bounding volume hierarchies (BVHs) have not reached the performance of kd-trees for ray tracing. BVHs do not adapt well to scenes containing large triangles with overlapping bounding boxes. A node cannot be smaller than the bounding box of the primitives it contains. Consequently, the leafs and internal nodes will overlap substantially. This slows down ray tracing, because the number of traversal steps and ray-primitive intersections is increased. Unfortunately, this kind of geometry is common in architectural scenes and low-poly CAD models.

In this paper, we present a building algorithm for BVHs that handles such scenes more efficiently. The restriction that each primitive must be contained in exactly one leaf node is relaxed. Bounding boxes of large primitives are refined with recursive split clipping *before* constructing the hierarchy. The resulting volumes are used as input for a regular BVH building algorithm. Neither scene geometry nor traversal or building algorithms must be modified in any way. The resulting hierarchies are superior for a wide range of data sets, leading to speed-ups of more than a factor of three.

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

For many years, kd-trees were the prevailing ray shooting acceleration structure. Their performance is still unrivaled for many applications but the price for fast ray shooting is high. Kd-trees take a long time to build and consume huge amounts of memory. Also the implementation of stable construction and traversal algorithms is a challenging task. Dynamic scenes are particularly difficult to render with kd-trees. This is the main reason for a renewed interest in alternative acceleration structures for ray tracing.

Bounding volume hierarchies and skd-trees are attractive because they can be built quickly and their memory consumption is low and predictable. Their ray tracing performance, however, is far below kd-trees for many real-world datasets. We have analyzed this behavior and present a simple solution which makes BVHs competitive to kd-trees for *all* kinds of scenes.

2 BACKGROUND

Ray tracing was used in 1968 by Appel [1] to compute renderings of solid models. The recursive ray tracing algorithm, as it is known today, was introduced twelve years later by Turner Whitted [31]. At that time, hand constructed bounding volume hierarchies [4] were the first acceleration structures for the reduction of intersection calculations [24]. Kay and Kajiya presented an automatic top-down building algorithm for BVHs with spatial median splits [15]. Bottom-up builds, using a cost model to compute the best split for each node, were introduced by Goldsmith and Salmon [6]. Müller [21] and Mahovsky [20] applied this method to recursive top-down construction of bounding volume hierarchies. Randomized builds were investigated by Ng and Trifonov [22].

Goldsmith and Salmon's cost function is now known as the surface area heuristic (SAH). Though originally designed for bounding volume hierarchies, its popularity grew with the advent of kd-trees [2] for ray tracing [14]. MacDonald and Booth applied SAH to the construction of this space subdivision technique [18]. Havran's systematic research of acceleration structures for ray tracing [9] and Wald's work on interactive ray tracing [27] made SAH kd-trees the de facto standard for efficiency schemes. A fast $O(N \log N)$ construction algorithm was presented by Wald and Havran [30]. It sorts the input data only once before construction, not at every level of the tree. Significant speed-ups have also been achieved by using a scanning approach, that evaluates the SAH at only a few sample positions [12], [23]. Wächter and Keller presented a very fast construction algorithm for bounding interval hierarchies (skd-trees) [26]. Their method can also be applied to BVH construction. Ize et al. demonstrated asynchronous BVH construction on parallel architectures [13].

Glassner developed hybrid trees, which mix the concepts of spatial and object hierarchies [5]. A technique known as oversize shelves was used by Günther [7] and later by Havran [11]. They store large objects close to the root of the hierarchy because most rays will have to intersect them anyway.

Efficient traversal of BVHs has also been addressed by many researchers over the past years. A series of optimizations to the basic traversal algorithm was described by Haines [8]. Smits introduced a memory layout that allows stack-free traversal of the hierarchy [25] and leads to more cache-friendly memory access patterns. Faster ray-box overlap and intersection tests were developed by Williams [32] and Mahovsky [19], [20]. Various techniques for ordered traversal of the child nodes were discussed. Kay and Kajiya [15] use a heap to sort child nodes according to their intersection distance along the ray. Mahovsky stores the splitting axis for each internal node and chooses the closer child according to the sign of the ray direction. Culling of bounding boxes that are further away than the current closest primitive intersection was proposed by Haines [8]. Smits suggested early exits for shadow rays [25]. Both authors applied various caching techniques for intersection results.

More recently, the focus in ray tracing research shifted towards interactive rendering of animated scenes. Wald ray traced rigid body transformations at interactive frame rates, using a two level kd-tree [28]. Larsson [16], Lauterbach [17] and Wald [29] addressed the construction and traversal of bounding volume hierarchies for deformable models. Wald also applied packet traversal to bounding volume hierarchies. The frame rates are competitive with the fastest known kd-tree implementations for a wide range of scenes. Boulos applied this approach to Whitted and distribution ray tracing, achieving significant speed-ups over single ray tracing [3].

*e-mail: manfred.ernst@cs.fau.de
†e-mail: greiner@cs.fau.de

IEEE/EG Symposium on Interactive Ray Tracing 2007
10 - 12 September, Ulm, Germany
978-1-4244-1629-5/07/$25.00 ©2007 IEEE

3 Spatial Hierarchies for Ray Tracing

Most acceleration structures are *spatial hierarchies*. Their leaf nodes store references to the geometric primitives (usually triangles) of the scene. The internal nodes build a hierarchy that is used to guide rays quickly towards all relevant primitives for intersection. Spatial hierarchies can be divided into two groups. *Space subdivision* techniques (e.g. kd-trees, octrees and grids) generate disjoint nodes, whereas each primitive can be contained in multiple leafs. *Object subdivision* methods (e.g. bounding volume hierarchies and skd-trees) ensure that each element is stored in exactly one leaf; the nodes may overlap. In this paper, kd-trees represent space subdivision techniques, while BVHs represent object subdivision. The basic construction and traversal algorithms are briefly reviewed here, preceding an explanation for the difference in performance.

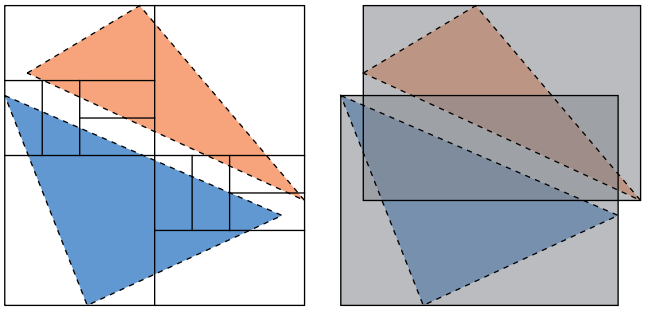

Figure 1: Spatial hierarchies. *Left:* Kd-tree leafs are always disjoint. *Right:* Bounding volume hierarchy with overlapping leaf nodes.

3.1 Kd-trees

An internal node of a kd-tree stores a split axis, a split position and pointers to its two child nodes. The leafs contain lists of references to scene primitives. Kd-trees are built top-down, by recursively splitting nodes until a termination criterion is reached. The recursion is initialized with a leaf, containing all primitives in the scene. Pseudo-code for the subdivision function is given in Algorithm 1. Line one computes the best split axis and position. The function returns in line two if no useful split was found. Bounding boxes B_L and B_R of the child nodes N_L and N_R are defined by the split in line three. They are disjoint ($B_L \cap B_R = \emptyset$) and their union is identical to the parent bounding box. For ray tracing, this implies that tree traversal can stop as soon as an intersection is found, after testing the remaining objects in the current leaf. The for loop sorts primitives into the lists P_L and P_R, overlapping B_L and B_R, respectively. An object may be added to both lists, if its bounding box straddles the splitting plane.

3.2 Bounding Volume Hierarchies

Each node of a BVH stores the axis aligned bounding box of the subtree below itself. Internal nodes have pointers to their left and right children. Leafs store a list of references to scene primitives. BVH construction is performed similar to a kd-tree build. Pseudo-code is given in Algorithm 2. In contrast to spatial subdivision, each primitive is referenced in exactly one leaf node. The primitive lists P_L and P_R are disjoint in each recursion step. The volumes B_L and B_R may overlap and their union is a subset of the parent bounding box. Bounding volume hierarchies can be constructed very quickly with low and predictable memory consumption. Their disadvantage is overlapping leaf nodes. Traversal algorithms for ray tracing are forced to continue, even if an intersection was found, because another leaf might be closer.

Algorithm 1: subdivideKdTreeNode(N, P)

```
   // N:  Leaf node to split
   // P:  List of primitives, stored in N
 1 findBestSplit(N, P, axis, splitPos);
 2 if axis = −1 then return;
 3 splitNode(N, axis, splitPos);
 4 N_L←leftChild(N);
 5 N_R←rightChild(N);
 6 B_L←boundingBox(N_L);
 7 B_R←boundingBox(N_R);
 8 for i ← 1 to size(P) do
 9     B←boundingBox(P[i]);
10     if overlaps(B, B_L) then append(P_L, P[i]);
11     if overlaps(B, B_R) then append(P_R, P[i]);
12 end
13 subdivideKdTreeNode(N_L, P_L);
14 subdivideKdTreeNode(N_R, P_R);
```

Algorithm 2: subdivideBVHNode(N, P)

```
   // N:  Leaf node to split
   // P:  List of primitives, stored in N
 1 findBestSplit(N, P, axis, splitPos);
 2 if axis = −1 then return;
 3 for i ← 1 to size(P) do
 4     B←boundingBox(P[i]);
 5     if center(B, axis) ≤ splitPos then
 6         append(P_L, P[i]);
 7         B_L ← B_L ∪ B;
 8     else
 9         append(P_R, P[i]);
10         B_R ← B_R ∪ B;
11     end
12 end
13 splitNode(N, axis, splitPos);
14 N_L←leftChild(N);
15 N_R←rightChild(N);
16 setBoundingBox(N_L, B_L);
17 setBoundingBox(N_R, B_R);
18 subdivideBVHNode(N_L, P_L);
19 subdivideBVHNode(N_R, P_R);
```

3.3 Surface Area Heuristic

Both kd-trees and BVHs require the computation of an optimal split axis and position. A reasonably good solution is to subdivide a node in the middle along its axis of largest extent. An improvement over this spatial median approach is the *surface area heuristic* (SAH) cost function. It is based on the following theorem: A ray, intersecting box A, will intersect box B, contained in A, with probability $SA(B)/SA(A)$, where $SA(\cdot)$ is the surface area of a box. This can be used to estimate the cost of a particular split for ray tracing:

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

where $|P_L|$ and $|P_R|$ is the number of primitives in the left and right child, respectively. The optimal axis and position are computed by evaluating this function for various candidate positions. SAH is applicable to kd-trees and BVHs.

The cost function implies that good splits generate small leaf nodes, with little overlap and few primitives. Such splits are not possible in regions containing large triangles with overlapping

bounding boxes. Rays passing through these areas will be tested for intersection against all triangles, degrading rendering performance considerably. The problem originates from the approximation of geometry by axis aligned boxes. An elegant solution for kd-trees is split clipping [10].

3.4 Split Clipping

With this extension, the geometry of each primitive is clipped to the boundaries of the child nodes before it is sorted into the lists P_L and P_R. This is also done when the best split axis and position are computed. A modified version of the for loop in Algorithm 1 is given in Algorithm 3. With split clipping, primitives are only referenced in leafs that they actually overlap.

Algorithm 3: Kd-tree subidivision with split clipping

```
1 for i ← 1 to size(P) do
2     P_- ←clipNeg(P[i], axis, splitPos);
3     B_- ←boundingBox(P_-);
4     if overlaps(B_-, B_L) then append(P_L, P[i]);
5     P_+ ←clipPos(P[i], axis, splitPos);
6     B_+ ←boundingBox(P_+);
7     if overlaps(B_+, B_R) then append(P_R, P[i]);
8 end
```

Thus the subdivision function can generate useful splits for any node. The surface area of the leafs can be reduced arbitrarily. This is the reason why kd-trees can adapt to almost any scene. However, split clipping will cut primitives regardless of their size, resulting in many multiple references in the final leaf nodes. This is one of the reasons why kd-trees consume so much memory and take a long time to build.

In a BVH, each primitive must be referenced exactly once. This implies that nodes cannot be subdivided further if they contain only one object. Split clipping is impossible. As a consequence, bounding volume hierarchies are much slower for ray tracing than kd-trees for many scenes. Large triangles spanning big parts of the scene are particularly ill-suited for BVHs.

4 Geometry Refinement

Leaf nodes of a BVH become smaller if the input geometry is refined. The top row in Figure 2 shows the effect of a one-to-four split. Each triangle of the subdivided mesh is represented by one bounding box for BVH construction (1st picture). The edges of subdivided triangles are not axis aligned. Consequently, their bounding boxes overlap and the resulting BVH will not be optimal near the leaf nodes (4th and 5th picture). However, the BVH builder contains information about the refined geometry from the outset, and therefore can compute a slightly better split for the root node (3rd picture). For real scenes, this effect is more pronounced and the upper hierarchy levels will benefit greatly from geometry refinement.

We made some experiments with a CAD dataset, tessellated at two levels of detail. Our tessellator generates perfect triangle meshes, with an extremely regular structure and a user controlled maximum edge length. As expected, a finer tessellation improves rendering performance. The reason is the reduced overlap in the upper hierarchy and the small leaf nodes. When a ray hits a primitive, it very likely needs no more intersections before termination. The results are shown in Table 1.

In practice, this approach is problematic. Input geometry must be modified in order to build a reasonable acceleration structure. In many rendering systems, it is difficult, or even impossible, to modify input geometry. Otherwise, the enormous memory consumption is limiting this approach. Triangle meshes with vertex normals and texture coordinates are heavyweight data structures. In complex scenes, geometry subdivision is not recommended.

5 Split Clipping for BVHs

Instead of refining the geometry, it would be more useful to subdivide it for hierarchy construction only. This is exactly what split clipping does, but it has to reference primitives in multiple leaf nodes. Surprisingly, there is no reason why a BVH should be limited to a single reference. The data structure and the traversal algorithms can handle multiple references without any modifications.

We have investigated two split clipping algorithms for bounding volume hierarchies. The first technique, called *late split clipping*, brings no speed improvements. It is only explained for the sake of completeness. The second approach, *early split clipping* is much easier to implement, does not require more memory and boosts ray tracing performance up to a factor of three.

5.1 Late Split Clipping

This technique builds a bounding volume hierarchy, using standard algorithms. The resulting leaf nodes are subdivided with spatial median split clipping, until their surface area is below a user defined limit $SA_{\max}$. Thus the leafs are shrunken to an arbitrary size. An example is shown in the middle row of Figure 2. Up to the 3rd picture, which shows the original leafs, it is identical to a regular BVH build. The 4th and 5th figures depict the nodes, generated by late split clipping. They are small and disjoint, but the higher levels are not. The problem with late split clipping is that tighter bounding boxes are not available until most of the hierarchy construction is done. Usually, there are many levels between the root and the original leafs and they will overlap heavily if the scene contains large triangles. As a result, the number of primitive intersections during ray tracing can be reduced but more traversal steps are necessary.

A special traversal routine was implemented to alleviate this problem. When the ray visits one of the original leaf nodes, the current stack pointer is saved. The pointer is always restored after a deep leaf is visited, because further traversal of the subtree is useless. Even with this technique, we could not achieve any performance improvement over a regular BVH.

5.2 Early Split Clipping

Geometry refinement leads to little overlap in the upper levels of the BVH, while late split clipping generates disjoint bounding boxes below the original leaf nodes. Both are desirable and can be achieved when split clipping is applied *before* construction of the hierarchy.

A BVH builder only needs a set of bounding boxes with references to the enclosed primitives as input. Usually, each primitive is represented by exactly one bounding box. By relaxing this restriction, the BVH can adapt much better to the input data. Large primitives are approximated by a set of small and disjoint bounding boxes for the hierarchy builder. Pseudo-code is given in Algorithm 4. The function is called for every primitive in the scene. It uses a stack to execute the recursive subdivision in an iterative manner. The stack is initialized with the original scene primitive in line one. Split clipping stops when the stack is empty. In each iteration, a tight bounding box B is computed for the top element. If the surface area of B is below a user defined threshold $SA_{\max}$, the box is added to the list L of bounding boxes for P. Otherwise, the primitive on top of the stack is split in the middle along the axis of largest extent and it is replaced by the resulting primitives P_- and P_+ in lines 12 to 13.

The resulting list of boxes L encloses the primitive more tightly than its original bounding box. In contrast to geometry refinement, the input data is not modified. Furthermore, the resulting boxes are disjoint, because of the axis aligned splits. The advantage over late split clipping is that the BVH can already adapt to the tighter boxes in the upper levels of the hierarchy. The BVH builder itself need not be modified. Our implementation is specialized for triangles as

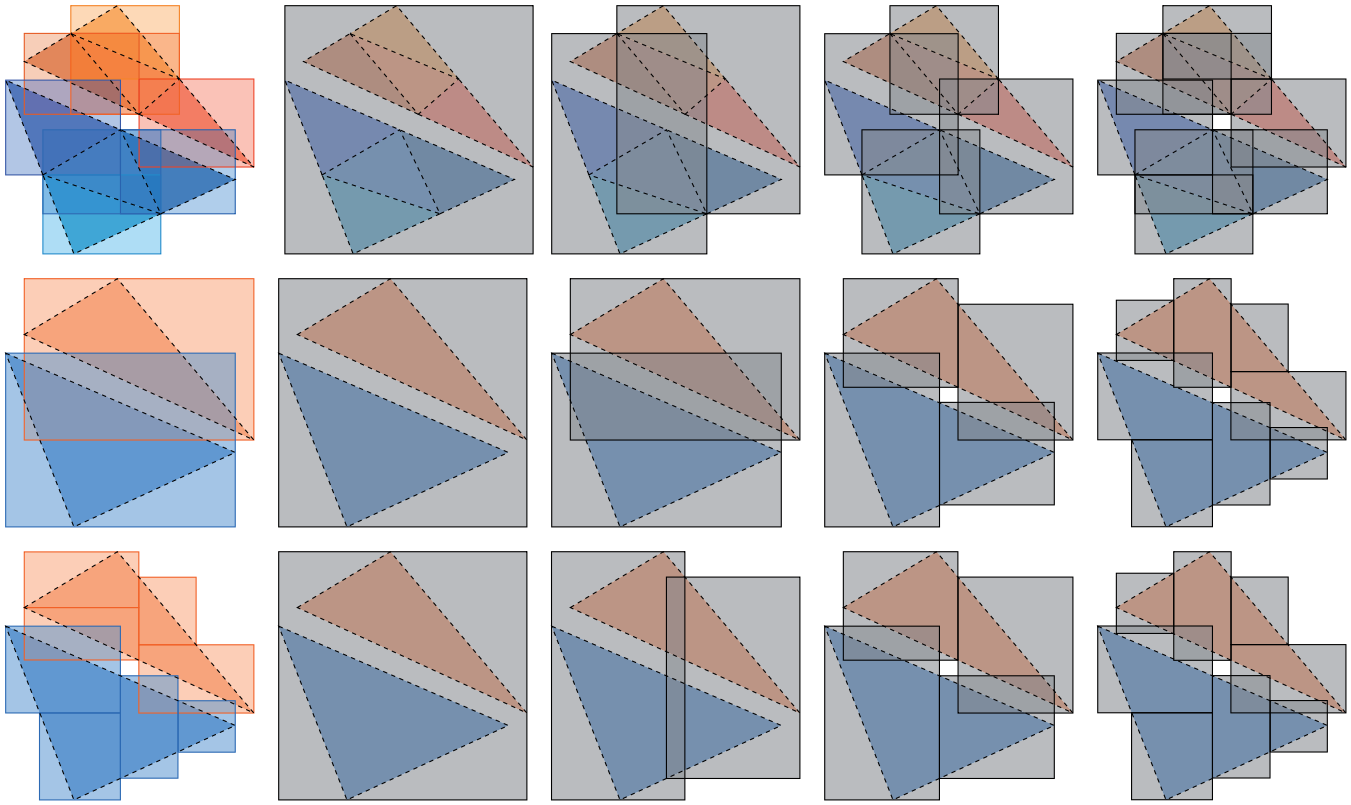

Figure 2: *Top row:* geometry refinement. *Center row:* late split clipping. *Bottom row:* early split clipping. *First column:* Input bounding boxes for BVH builder. Boxes have the same colour as the triangles they represent. *Columns 2-5:* Bounding boxes of all four levels of the BVHs. The root node is shown in the second column and the leafs are depicted in the last column.

input. In general, the algorithm is applicable to any primitive that can be split into two halves at an axis aligned plane.

Algorithm 4: earlySplitClipping(P, $SA_{\max}$, L)

```
// P:  input primitive from the scene
// SA_max:  surface area limit for leafs
// L:  resulting list of boxes
push(stack, P);
while notEmpty(stack) do
    B←boundingBox(top(stack));
    if surfaceArea(B) ≤ SA_max then
        append(L, B);
        pop(stack);
    else
        axis ←axisOfLargestExtent(B);
        splitPos ←center(B, axis);
        P_ ←clipNeg(top(stack), axis, splitPos);
        P+ ←clipPos(top(stack), axis, splitPos);
        pop(stack);
        push(stack, P_);
        push(stack, P+);
    end
end
return L;
```

The results of early split clipping are shown in the bottom row of Figure 2. In the example, our algorithm generates four bounding boxes for each triangle (1st picture). Those boxes are the input for the BVH builder. They are colored according to the triangles they represent. The nodes at the bottom of the hierarchy (4th and 5th picture) are identical to late split clipping. The superior quality of early split clipping is visible in the upper levels of the BVH (3rd picture). A good vertical split could be found because all bounding boxes are available from the beginning. This is the major difference between early and late split clipping.

6 Results

We have implemented early split clipping in our photo-realistic rendering system. It uses pure single ray tracing without any SIMD or assembly optimizations. The software is designed for high rendering quality and flexibility. As our optimization is a pure algorithmic improvement to the hierarchy construction, we propose that the results are beneficial for all kinds of ray tracing.

Many tests were performed with different data sets, including scanned models, tessellated CAD data and architectural models. The four scenes depicted in Figure 3 were chosen as representatives. Results in Table 1 are given for a kd-tree, a standard BVH build ($SA_{\max} = \infty$) and early split clipping BVHs with two different surface area limits. $SA_{\max}$ is given after "bvh" in the first column. All construction algorithms use the SAH cost function. The images were rendered from the viewpoints in Figure 3 with a resolution of 1000×1000 pixels and one primary ray per pixel. Timings were measured on a single core of a 2.16 GHz Core 2 Duo CPU.

A regular BVH is already very close to the kd-tree for homogeneous scenes with low depth complexity (Stanford bunny). Split clipping cannot improve performance for these models, but the complex scenes benefit greatly from it.

The two levels of detail of the mirror reveal that split clipping is better than geometry refinement. The BVH with $SA_{\max} = 10$ for the low poly model consumes less memory than the fine tes-

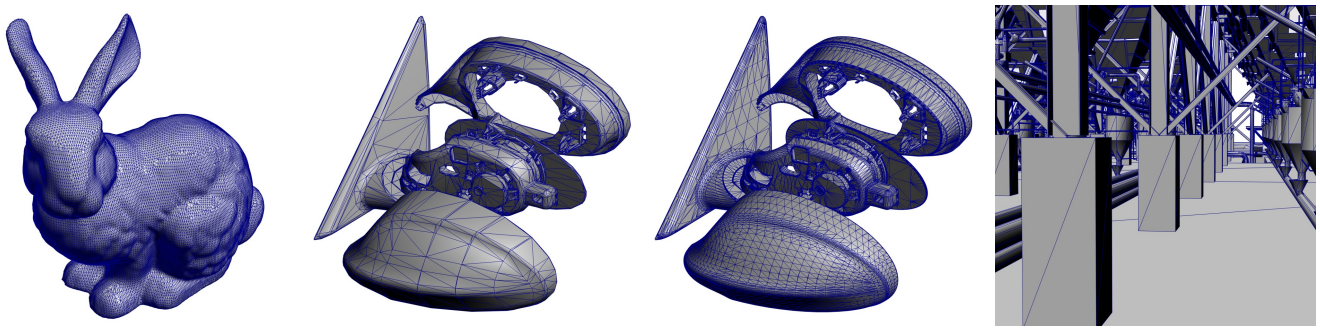

Figure 3: Test scenes rendered in wireframe-on-shaded mode with OpenGL. Images were raytraced with the same camera parameters for the results section. *Bunny*: 69,451 triangles, *Mirror low*: 85,651 triangles, *Mirror high*: 334,080 triangles, *Power plant (part)*: 1,667,578 triangles.

	R	T_{build}	N_{node}	$N_{\triangle}$	T_{ray}	S
Bunny						
bvh ∞	1.00	0.57	28.6	1.33	1.63	1.00×
bvh 3	1.50	1.00	29.6	1.15	1.63	1.00×
bvh 1	3.52	2.80	30.8	0.96	1.68	0.97×
kd-tree	10.60	8.75	47.8	2.00	1.65	0.98×
Mirror low						
bvh ∞	1.00	0.80	60.6	9.73	4.37	1.00×
bvh 100	1.25	1.01	41.0	2.32	2.25	1.94×
bvh 10	3.44	3.43	37.9	1.13	1.98	2.20×
kd-tree	13.39	13.6	30.9	4.73	1.70	2.57×
Mirror high						
bvh ∞	1.00	3.34	48.0	4.99	3.05	1.00×
bvh 100	1.04	3.60	44.5	3.05	2.55	1.19×
bvh 10	1.68	6.46	40.4	1.54	2.16	1.41×
kd-tree	11.53	50.9	34.7	3.53	1.63	1.87×
Power plant						
bvh ∞	1.00	20.6	193	58.2	18.0	1.00×
bvh 10^6	1.49	32.8	160	9.86	7.84	2.29×
bvh 10^5	4.88	123.0	127	3.12	5.39	3.33×
kd-tree	10.01	176.7	96.8	15.6	4.20	4.28×

Table 1: Results. Parameter after bvh: surface area limit $SA_{\max}$. R: Ratio of references in leafs to number of triangles. T_{build}: time in seconds for hierarchy build. N_{node}: mean number of visited nodes per ray. $N_{\triangle}$: mean number of triangle intersections per ray. T_{ray}: time in seconds for ray shooting of primary rays, excluding ray setup and shading. S: rendering speed-up over regular bvh ∞.

sellation: $85,651 \times 3.44 = 294,639$ input bounding boxes versus 334,080 triangles. Construction takes approximately the same time, but rendering is much faster: 1.98 seconds instead of 3.05.

Early split clipping is most effective for scenes like the power plant. Traversal steps are reduced to 65% and triangle intersections are cut down to 5% for this model. This results in a ray tracing speed-up by a factor of 3.33.

The surface area limit $SA_{\max}$ controls the memory consumption and rendering time for bounding volume hierarchies. The total number of references R increases when the surface area limit is decreased. Surprisingly, the mean depth of the BVH is hardly changed (not shown in the table). Obviously, the superior structure of the hierarchy compensates the higher number of input bounding boxes.

7 Conclusion

We have presented a very simple and elegant extension to build better bounding volume hierarchies for static scenes. It allows BVHs to adapt to any geometry and makes them more similar to kd-trees. In fact, the surface area limit $SA_{\max}$ allows us to blend between strict BVH behaviour (less memory, faster build, lower performance) and kd-tree behaviour (more memory, slower build, higher performance). In comparison to a regular BVH, our approach is much faster. In the future, we will try to find some heuristics to choose the surface area limit automatically for a given scene.

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

[5] A. S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, 1988.

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

[7] O. Günther and H. Noltemeier. Spatial database indices for large extended objects. In *ICDE*, pages 520–526. IEEE Computer Society, 1991.

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

[9] 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.

[10] V. Havran and J. Bittner. On improving kd-trees for ray shooting. *Journal of WSCG*, 10(1):209–216, February 2002.

[11] V. Havran, R. Herzog, and H.-P. Seidel. On the fast construction of spatial data structures for ray tracing. pages 71–80, Sept. 2006.

[12] 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.

[13] 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*.

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

[15] 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.

[16] T. Larsson and T. Akenine-Möller. Strategies for bounding volume hierarchy updates for ray tracing of deformable models. tech. rep. mdh-

mrtc-92/2003-1-se. Technical report, Malardalen Real-time Center, February 2003.

[17] 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.

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

[19] 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.

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

[21] G. Müller and D. W. Fellner. Hybrid scene structuring with application to ray tracing. In *Proceedings of International Conference on Visual Computing (ICVC'99)*, pages 19–26, Goa, India, February 1999.

[22] K. Ng and B. Trifonov. Automatic bounding volume hierarchy generation using stochastic search methods. In *CPSC532D Mini-Workshop "Stochastic Search Algorithms"*, April 2003.

[23] 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.

[24] 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.

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

[26] 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.

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

[28] I. Wald, C. Benthin, and P. Slusallek. Distributed interactive ray tracing of dynamic scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*.

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

[30] 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.

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

[32] A. Williams, S. Barrus, R. K. Morley, and P. Shirley. An efficient and robust ray-box intersection algorithm. *Journal of Graphics Tools: JGT*, 10(1):49–54, 2005.