Spatial Splits in Bounding Volume Hierarchies

Martin Stich* Heiko Friedrich† Andreas Dietrich‡

NVIDIA Research

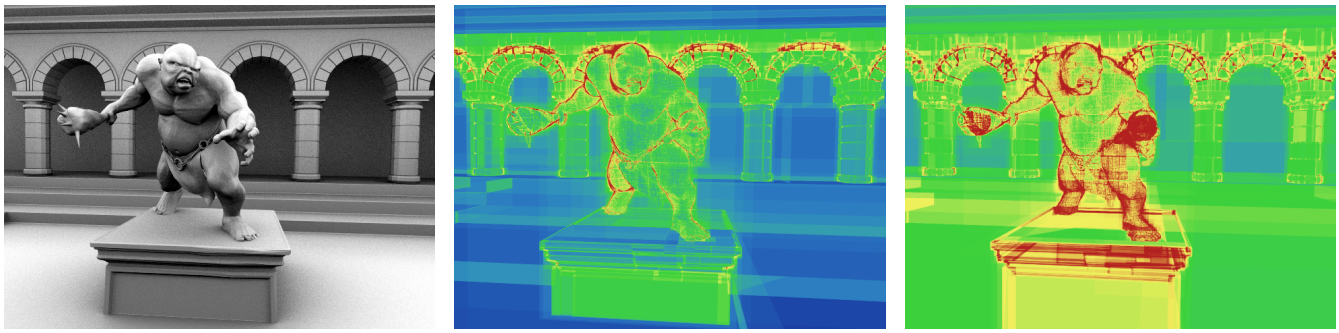


Figure 1: *Sample scene consisting of roughly 1.9 million triangles (left). Our method (middle) results in a significant reduction of ray shooting costs compared to a regular bounding volume hierarchy (right). The heat views visualize the summed number of traversal steps and primitive intersections for primary rays.*

Abstract

Bounding volume hierarchies (BVH) have become a widely used alternative to kD-trees as the acceleration structure of choice in modern ray tracing systems. However, BVHs adapt poorly to non-uniformly tessellated scenes, which leads to increased ray shooting costs. This paper presents a novel and practical BVH construction algorithm, which addresses the issue by utilizing spatial splitting similar to kD-trees. In contrast to previous preprocessing approaches, our method uses the surface area heuristic to control primitive splitting during tree construction. We show that our algorithm produces significantly more efficient hierarchies than other techniques. In addition, user parameters that directly influence splitting are eliminated, making the algorithm easily controllable.

CR Categories: 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

Keywords: ray tracing, bounding volume hierarchy

1 Introduction

Ray tracing is a powerful image synthesis technique, which has been used for offline rendering since decades and is becoming increasingly important for real-time applications. However, ray tracing is compute intensive and has to rely on preprocessed data structures to achieve fast performance. Among the methods to improve ray shooting efficiency, hierarchical data structures are currently the most important.

*e-mail: mstich@nvidia.com
†e-mail: heiko@graphics.cs.uni-sb.de
‡e-mail: dietrich@graphics.cs.uni-sb.de

In general, highest rendering performance is achieved with kD-trees and bounding volume hierarchies (BVH). Especially BVHs have recently been attracting increasing attention for several reasons: BVHs are simple to construct, have a low memory footprint, allow refitting in animations, and work well with packet tracing techniques [Wald et al. 2007]. It has also been found that BVHs tend to outperform kD-trees on GPU architectures, even for single ray implementations [Luebke and Parker 2008].

Over the last few years, ray tracing efficiency with BVHs has been tremendously improved, but this is almost solely due to novel and highly optimized traversal algorithms. Only little research tackling fundamental problems of the BVH, like large and heavily overlapping bounding volumes, has been done. In this paper, we propose a new method to significantly increase the efficiency of BVHs, especially for difficult scenes with highly non-uniform primitive sizes. Our main contribution is an algorithm that greatly reduces overlap of bounding volumes by using spatial splits during BVH construction. The basic idea is to split a given node using either object list partitioning or spatial partitioning by selecting the more cost-effective scheme. We demonstrate that the new method improves several significant hierarchy metrics compared to other techniques: overlap area, SAH cost, traversal steps and primitive intersections are all reduced in the vast majority of cases. This results in consistently improved ray tracing performance compared to traditional methods in all our experiments.

2 Background

Ray tracing acceleration structures exploit spatial coherence by sorting a scene's primitives into spatial groups. That way, objects of a group can be rejected immediately if the group is not relevant to a ray. Typically, the most effective methods employ hierarchical tree data structures, whose leaves reference primitives. Internal nodes contain spatial information to cull the associated part of a scene. Generally, such acceleration data structures can be divided into two categories: spatial partitioning schemes and object partitioning schemes.

Spatial partitioning schemes recursively subdivide a given space and distribute the geometric primitives into the resulting partitions. Each primitive is inserted into all partitions it overlaps, potentially resulting in multiple references for a primitive. This process is re-

peated until some termination criterion is met. kD-trees [Bentley 1975; Sung and Shirley 1992] perform binary spatial partitioning using one axis aligned plane per node. An important property of kD-trees is their ability to adapt to arbitrary scenes with highly varying geometric densities. This leads to excellent culling efficiency even in difficult settings. On the downside, kD-trees suffer from high memory consumption due to deep trees and high reference duplication.

Object partitioning schemes, on the other hand, recursively divide a list of primitives into disjoint sets. The full bounding volumes of those sets are stored in the tree nodes and can thus have arbitrary overlap. Such overlapping regions are expensive during traversal, because rays that intersect these regions must traverse all nodes contributing to the overlap. The actual definition of the bounding volumes is arbitrary, but in practice, axis aligned bounding boxes (AABBs) are most commonly used. They can be handled very efficiently and in most cases enclose primitives reasonably well. Since in a conventional BVH each primitive is referenced exactly once, the hierarchies consist of fewer nodes than kD-trees for virtually all scenes.

While in principle the use of n-ary data structures is possible for both spatial partitioning and object partitioning schemes, the remainder of this work will focus on binary trees.

2.1 BVH Construction

In many implementations, BVHs are constructed as described in [Wald et al. 2007]: in each partitioning step, the primitive list of the current node P is sorted based on the centroids of the primitive AABBs. This ordered list is then split into two subsets P_1 and P_2, for each of which a bounding box is created and assigned to the corresponding node's children. This process is recursively continued.

In order to find a suitable split position in a specific primitive list, sorting is performed for each of the three Cartesian coordinate axes. The split axis and position is chosen based on the lowest estimated ray tracing cost for the new child nodes. In order to estimate the cost of a particular split, the *surface area heuristic* (SAH) cost function [Goldsmith and Salmon 1987; MacDonald and Booth 1989] is used. This heuristic relies on the assumption that rays are uniformly distributed and do not intersect any primitive in the scene. A ray which intersects a parent box B_p with surface area $SA(B_p)$ then intersects a child box $B_c \subseteq B_p$ with surface area $SA(B_c)$ with probability $SA(B_c)/SA(B_p)$. Using the SAH, the cost C of tracing a ray through a node B and its two children B_1 and B_2 is estimated as:

$$C = C_t + \frac{SA(B_1)}{SA(B)}|P_1|C_i + \frac{SA(B_2)}{SA(B)}|P_2|C_i,$$

where C_t is the cost of a traversal step, $|P_1|$ and $|P_2|$ denote the number of primitives in each subset, and C_i is the cost of a single ray-primitive intersection. The SAH not only helps to find a good split position, it can also be used as a termination criterion for node subdivision. In this case, a leaf is created whenever the cost for splitting the node is higher than the cost of sequentially intersecting all primitives.

3 Related Work

[Glassner 1988] introduced a hybrid technique combining adaptive space subdivision with bounding volumes. The method builds an octree over the scene and uses the resulting nodes to guide the bottom-up construction of an overlap-free BVH. This BVH can contain references to the same primitive in multiple leaves.

In order to improve BVH ray tracing performance in scenes with non-uniform tessellation, [Ernst and Greiner 2007] proposed an extended construction method called *Early Split Clipping* (ESC). They build on the observation that it can be beneficial to reference primitives more than once in a BVH and use smaller bounding boxes instead. This fact is exploited in a preprocess, which splits primitive bounding boxes (not the primitives themselves), and creates new primitive references with tightened AABBs. Each AABB is split recursively at the center of its longest axis until the box surface area is below a user defined threshold SA_{max}. The final set of references is then passed on to a regular BVH build process. The resulting hierarchies are often of higher quality than regular BVHs and succeed in improving ray casting performance. However, it is left to the user to find a reasonable value for SA_{max}, which can be a tedious process. In addition, because the tree is simply built over the reference AABBs using an unmodified BVH construction method, multiple (unnecessary) references to the same primitive often occur in the same leaf.

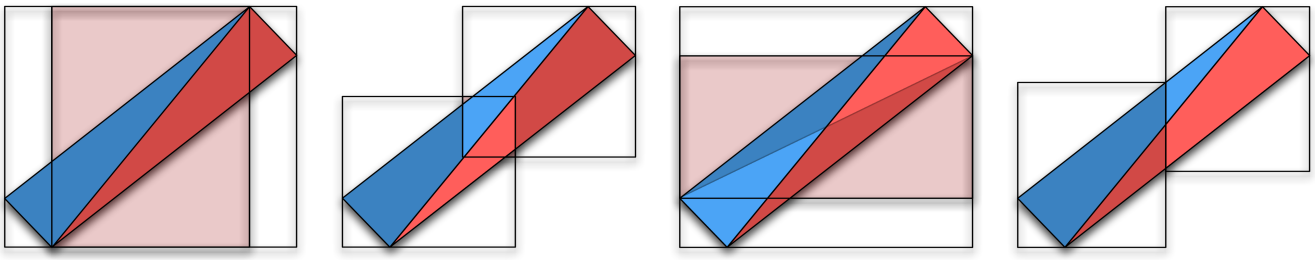
Similar to Early Split Clipping, [Dammertz and Keller 2008] developed the *Edge Volume Heuristic* (EVH) to reduce node overlap. Their algorithm differs mainly in that actual triangles are subdivided, not bounding boxes. Subdivision is done recursively and stops once the largest bounding volume of the triangle edges falls below a certain threshold. Tessellation is performed in a numerically robust fashion by dividing the triangle edge with the largest AABB volume in the middle. The resulting primitive bounding boxes can again be passed to a regular BVH builder. Unlike for ESC however, [Dammertz and Keller 2008] propose to modify the construction method to remove duplicate references from leaves. The user defined subdivision threshold is related to the scene volume, making EVH somewhat more controllable than ESC. Because EVH operates on edge AABB volumes, axis-aligned primitives are never subdivided. This approach is in line with the idea that only extremely expensive (non axis-aligned) triangles should be split. On the other hand, many opportunities are missed to improve the common case of overlap due to axis-aligned primitives.

3.1 Spatial Split Motivation

Both ESC and EVH perform splitting on a per-primitive basis and do not take information about the surrounding geometry into account. Because the split positions for each primitive are independent, the resulting reference AABBs tend to be unaligned, which in turn easily results in unnecessary node overlap. Our method makes informed splitting decisions during hierarchy construction by considering an entire set of primitives in a node. It is thus able to split multiple references at once, and only do so if the estimated cost can be reduced.

Figure 2 shows through a simple example how previous techniques can be improved. Two triangles are partitioned as during hierarchy construction, making the effects of various splitting techniques visible. Using a regular BVH results in two child bounding volumes that almost entirely overlap (a). In the case of ESC (b), the triangle AABBs are pre-split. Because the split positions fail to line up horizontally, the AABBs cannot be separated by child boxes without overlap. Subdividing the triangles according to the EVH (c) shows hardly any improvement over a regular BVH in this case. As can be seen in (d), a single spatial split produces child boxes without any overlap.

Note that although this work focuses on constructing hierarchies over triangles, the presented approach is applicable for any type of primitive that can be clipped against axis-aligned planes.



(a) (b) (c) (d)

Figure 2: Overlap reduction with spatial splits. *In this example, a node containing two triangles is partitioned, showing the effects of various splitting strategies. The black boxes represent the child nodes produced by the partitioning. The pink shaded areas are the regions of overlap. The splitting strategies used are: (a) no splitting (BVH), (b) ESC, (c) EVH, and (d) our method (SBVH). All methods but the SBVH produce overlapping bounding boxes in this case.*

4 The SBVH

To improve the efficiency of BVHs, we introduce a new construction algorithm, combining a traditional BVH build with the idea of spatial splits. We call our approach *Split Bounding Volume Hierarchy* (SBVH). In the spirit of [Glassner 1988], the restriction that each primitive is only referenced once inside a BVH is relaxed. This does not affect BVH traversal – a hierarchy which references a primitive in more than one leaf can be used with an unmodified standard traversal algorithm. We then observe that a node partitioning step during BVH construction is no longer limited to sorting a primitive into either of the two child nodes. Instead, primitives can now be sorted into *both* children, therefore duplicating the primitive reference and potentially decreasing the size of the child bounding boxes. This reference splitting during the construction phase (as opposed to a preprocess) is a fundamental difference to previous approaches.

4.1 SBVH Construction Algorithm

Similar to [Ernst and Greiner 2007], we build the hierarchy over primitive references. A reference is a pointer or index to the associated primitive, combined with an axis-aligned bounding box. Initially, there is one reference for each primitive, and the AABB is set to the primitive's bounding box. During tree construction, a reference can be split, i.e., it is duplicated, and the AABBs of the resulting children are updated to tightly enclose the contained part of the original primitive. The final SBVH is thus equivalent to a regular BVH over all the generated references (not primitives).

Adding the capability of splitting references to a BVH results in much more freedom when deciding how to partition a node. However, the search space of possible splits for a given node also becomes significantly larger, thus making the process of finding a good partitioning more complex.

We address the problem by separately considering splits which do duplicate references (*spatial splits*) and those which do not (*object splits*). The basic partitioning algorithm consists of three steps:

1. **Find an object split candidate.** This step is equivalent to the conventional split search for a BVH node (see Section 2.1). We use a full SAH search in our implementation, but other variants could easily be used.

2. **Find a spatial split candidate.** This step is similar to the partitioning process in kD-tree construction algorithms and will be described in more detail below.

3. **Select the winner candidate.** Based on the SAH cost, the cheaper of the above split candidates is chosen as the winner. The node is partitioned according to that candidate if the criterion for creating a leaf is not met.

Given a standard BVH builder, it is usually straightforward to replace its node partitioning method by the above algorithm.

4.2 Chopped Binning

As already mentioned, step 2 of the partitioning algorithm is somewhat similar to finding a split plane in a kD-tree. A common way to partition a kD-tree node is to consider split plane candidates at the AABB boundaries of all references in the node. Because this full SAH approach is expensive, binning [Hunt et al. 2006; Popov et al. 2006; Shevtsov et al. 2007] is commonly used to speed up kD-tree construction. With binning, split planes are considered only at a fixed number of equidistant positions within the node AABB. The references are projected into the resulting bins in a fast $O(N)$ pass, and counters associated with the bins are incremented for every projected primitive. The SAH cost for each split candidate can then be evaluated directly, without requiring a full sorting step. After selection of the best split plane, a final $O(N)$ pass distributes the references into the child boxes.

We adopt the binning idea for the spatial split search in the SBVH. Because SBVH nodes store full bounding boxes and always tightly enclose their references, node children can adapt their size in all dimensions, not just in the split dimension. Therefore, a simple counter for each bin, like in conventional kD-tree binning methods, is not sufficient. Instead, we need to store one AABB per bin, similar to techniques for BVHs [Havran et al. 2006; Wald 2007]. The actual binning process for a reference considers all bins that the reference AABB overlaps. For each of those bins, the referenced primitive is clipped against the bin boundaries, resulting in an AABB that bounds the portion of the primitive inside the bin. This resulting AABB is used to grow the AABB associated with the bin. Similar to the min-max binning method for kD-trees [Shevtsov et al. 2007], we maintain two additional sets of counters to keep track of the number of reference entries and exits in each bin. The process is illustrated in Figure 3. Because a reference is clipped against multiple bins, we call the method *chopped binning*. Note that this clipping of actual primitives against planes is analog to split clipping [Havran 2000] at each candidate plane. Build performance can be increased by clipping only the reference AABBs. [Soupikov et al. 2008] noted that this approach reduces hierarchy quality slightly for kD-trees, and our SBVH experiments are in line with this observation. Thus, enabling candidate split clipping can be regarded as a quality vs. performance trade-off parameter like

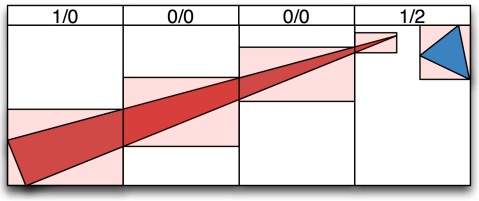


Figure 3: Chopped binning. *Primitives are clipped against each bin boundary, and the resulting boxes (light red) are used to grow the AABBs of the bins. The top row shows the entry and exit counters after both references have been binned.*

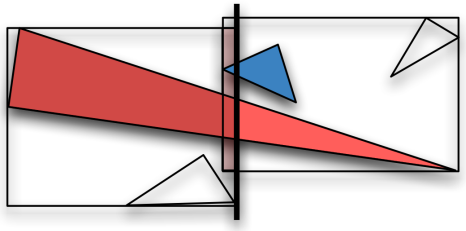


Figure 4: Unsplitting a reference. *The reference to the red triangle is split and inserted into both child nodes. The blue triangle's reference is inserted only into the right child, even though it straddles the split plane as well. This introduces slightly overlapping bounding boxes, but potentially improves the SAH cost of the split.*

for the kD-tree.

When all references have been binned, it is straightforward to evaluate the SAH costs for split planes at the bin boundaries in a linear pass. The left and right child boxes are given by the union of all AABBs stored in the left and right bins, respectively. The number of references on the left is computed by summing the entry counters left of the split position; the number of references on the right is given by the sum of the exit counters on the right of the split position. After computing the SAH costs for all split candidates, the cheapest split is selected for subsequent comparison against the best object split (step 3 of the partitioning algorithm in Section 4.1).

4.3 Spatial SBVH Splits vs. kD-tree Splits

Although spatial splits in the SBVH are similar to kD-tree splits, there are a few important differences. The most obvious one is the previously mentioned fact that in the SBVH, child nodes tightly enclose their references. One consequence of this is the absence of empty nodes in the SBVH. While most kD-tree implementations take special care to minimize empty space, e.g. by creating empty leaves, the concept does not apply for the SBVH. A few methods for cutting off empty space in kD-trees are discussed in [Havran 2000].

Another interesting difference to kD-trees is that we never switch to finding the exact minimum of the SAH cost function, as is commonly done in other binning algorithms once the number of references is below a certain threshold. The reason is that, unlike for a kD-tree, the SAH cost function is not guaranteed to have its minimum at one of the reference bounds. Because the child bounding boxes can adapt its size in all three dimensions, the function is piecewise quadratic[1], as opposed to piecewise linear for kD-trees. Furthermore, there may be C^1-discontinuities which do not lie on a reference bound. This makes finding the exact minimum unreasonably complex, and so we resort to binning at all levels of the hierarchy.

In general, the spatial split search for the SBVH can be kept rather simple compared to the methods commonly used for kD-trees. Nodes which are difficult to split by a single plane are problematic for kD-trees. Simply creating a leaf in such cases can result in inefficient trees, and good alternatives tend to be somewhat complex to find. For instance, [Havran 2000] uses a heuristic to decide whether a particular split could pay off in subsequent levels of the hierarchy, even if its cost is relatively high. Such methods are not necessary in the SBVH build. If the best spatial split candidate is too costly, the object split candidate will simply be chosen instead. Hence, a successful spatial split will never be worse than a regular BVH-like object split (in terms of SAH cost), and not finding a good spatial split will never generate a worse split than a regular BVH would.

[1] For triangles and other linear primitives.

4.4 Reference Unsplitting

While previously we have made a clear distinction between spatial splits and object splits, there is in fact no reason why the two concepts have to be kept separate. Because SBVH nodes hold full bounding boxes (in contrast to a single plane like kD-tree nodes), they can be allowed to overlap even for spatial splits. One can thus create splits which are hybrids between spatial and object splits: some references are duplicated at the split plane, while others are allowed to straddle the plane. The latter are sorted into only one child and therefore cause node overlap. An example of such a split is shown in Figure 4. We exploit the idea of hybrid splits during spatial split search in order to further improve SAH cost.

When considering a spatial split plane, there are three possibilities for each reference intersected by the plane: it can either be split into both children, or put entirely into only one of the two child boxes. Thus, for N straddled references, there are 3^N possible partitionings – generally too many to test exhaustively.

We employ a simple heuristic to select one of the three options for each reference. First, the spatial split is computed as described previously, with all the straddled references being split at the candidate plane. This results in two child boxes, B_1 and B_2, as well as the reference counts for these children, N_1 and N_2. We then test for each split reference whether "unsplitting" it, i.e. moving it entirely to one of the children, decreases the SAH cost of the partitioning. This is a conservative cost estimate, because the target box may grow, while the box from which the reference is removed is not recomputed and thus cannot shrink. In other words, we compare the cost equations

$$\begin{aligned} C_{split} &= SA(B_1) \cdot N_1 + SA(B_2) \cdot N_2 \\ C_1 &= SA(B_1 \cup B_\triangle) \cdot N_1 + SA(B_2) \cdot (N_2 - 1) \\ C_2 &= SA(B_1) \cdot (N_1 - 1) + SA(B_2 \cup B_\triangle) \cdot N_2, \end{aligned}$$

where $B_\triangle$ denotes the bounding box of the (unsplit) reference in question, C_{split} is the cost for splitting the reference, and C_1 and C_2 are the costs of putting it entirely into either one of the child boxes. We then choose the action with the cheapest cost. In our experiments, reference unsplitting resulted in a slightly improved total SAH cost in almost all cases.

4.5 Restricting Spatial Split Attempts

The spatial split approach described so far is very successful in reducing node overlap. However, one of the important advantages of

regular BVHs over other acceleration structures is their low memory consumption. Any method involving reference duplication necessarily conflicts with that goal. To a certain extent, the SBVH inherits the issue from kD-trees, which often imply large memory footprint and deep hierarchies, especially if the desired number of references per leaf is small.

Unlike for kD-trees, solving the problem for the SBVH is rather simple. At no point during construction are we forced to actually use a spatial split – a simple object split is always a valid alternative. Thus, we can carefully choose the nodes for which spatial splits are even considered, and make sure that reference duplication only occurs where the expected benefit is high. Since the main purpose of spatial splits is to reduce node overlap, we use the amount of overlap produced by the best object split as a decision criterion. More precisely, we compute the surface area of the overlap AABB

$$\lambda = SA(B_1 \cap B_2),$$

where B_1 and B_2 are the child bounding boxes of the object split candidate with the lowest cost. We then relate λ to the surface area of the hierarchy's root node and compare it to a user constant α:

$$\frac{\lambda}{SA(B_{root})} > \alpha$$

If the above condition is not met, step 2 of the partitioning algorithm (Section 4.1) is omitted, and the object split candidate is used exclusively. The parameter α is chosen to lie within the interval $[0, 1]$ and blends between a regular BVH without any duplication ($\alpha = 1$) and a full SBVH ($\alpha = 0$). For the full SBVH, a spatial split is attempted whenever the references could not be separated without overlap by the best object split. Intuitively, α denotes the overlap area to root area ratio which is tolerated without attempting a spatial split. It is important to note that the user constant only guides the spatial split *attempts*, not the reference duplication itself. The actual splitting decisions are left to the SAH, eliminating the risk of excessive splitting due to a poor choice of parameters. This is in contrast to previous methods, where the user controls splitting directly by choosing a threshold value, without being able to rely on an automatic mechanism.

Experiments confirmed that applying our heuristic with α close to zero generally results in hierarchies with excellent properties: because we measure the amount of overlap relative to the root node (as opposed to e.g. relative to the node to be partitioned), smaller nodes are unlikely to reach the threshold. Therefore, most spatial splits occur close to the top of the tree, where they are most effective. Further down in the hierarchy, object splits are used almost exclusively. This is acceptable, since potential overlap is small and only affects relatively few rays during ray tracing. Compared to a full SBVH, a non-zero α results in shallower trees requiring significantly fewer references, yet leads to hardly any performance degradation. Figure 5 shows a typical case, where practically all of the attainable SAH cost improvement is reached around $\alpha = 10^{-5}$, while requiring only a fraction of the full duplication rate.

5 Results

We compared the SBVH to Early Split Clipping, the Edge Volume Heuristic, and a regular BVH build. ESC, EVH, and the regular BVH used the full SAH evaluation described in section 2.1 at all levels of the hierarchy, i.e. no binning or other approximations were used. With all methods, nodes were split until the number of references was eight or fewer. In order to fairly compare different construction algorithms under similar memory budgets, we adjusted the parameters of all splitting strategies such that they resulted in the same number of references. The SBVH user constant α was

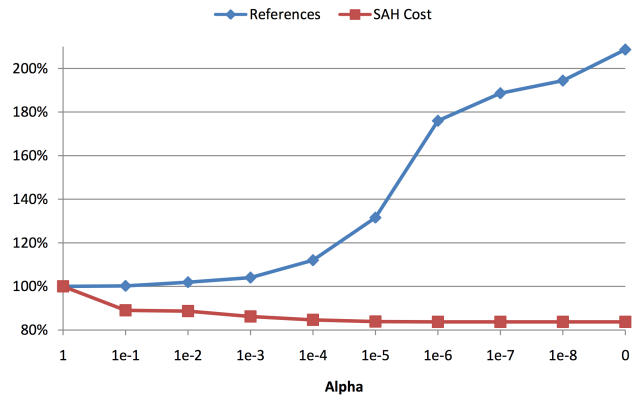


Figure 5: Influence of the α parameter. *The plot shows the SAH cost and reference duplication rate with varying α for the conference room scene. We found that a good performance/duplication ratio is achieved around $\alpha = 10^{-5}$ for a wide range of scenes.*

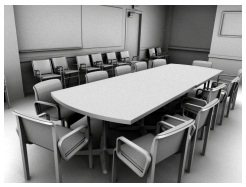
set to 10^{-5} in all reported tests. The number of bins for the SBVH spatial splits was 256 at all levels of the hierarchy. For the timing comparisons, we used a CUDA-based GPU ray tracer, which implements our currently fastest traversal kernel. Performance was measured on a NVIDIA GeForce GTX280 with CUDA 2.2.

Table 1 shows various statistics for the scenes we have used in our experiments. As can be seen, the SBVH outperforms both regular BVHs and the other splitting approaches in nearly every comparison. The highest performance improvement is achieved in the Sponza scene. Similar to [Dammertz and Keller 2008], we rotated the original model around the canonical axes in order to provoke a near worst case scenario for the traditional BVH. As a consequence, the SBVH increases the number of references by about a factor of two. In all other cases, which are less extreme, memory overhead is far lower with its maximum as little as ~30%.

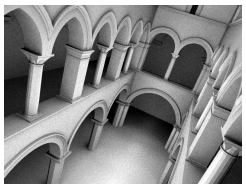
An especially noteworthy case is the "Bubs" scene: it consists of a character with high polygon count (~1.8M triangles), surrounded by a environment with only ~100K triangles. Because the majority of the geometry is very finely and uniformly tessellated, one would expect spatial splits not to improve performance much. Indeed, this is true if the environment is removed from the model, in which case almost no reference duplication (and therefore no speedup) occurs. However, in the full scene, the low environment tessellation causes overlap in the important top levels of a BVH, which in combination with the dense character geometry makes hierarchy traversal expensive. This situation is mitigated with minimal reference duplication by the SBVH, which easily avoids the top level overlap by placing a small number of spatial splits. The effect is visualized in Figure 1.

6 Conclusions

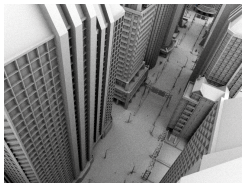
In this paper, we have presented the Split Bounding Volume Hierarchy (SBVH), which offers significantly improved culling efficiency over conventional BVHs, and in our experiments consistently outperforms previously published optimization techniques. The SBVH mostly retains the BVH advantages of low memory footprint and shallow hierarchies. In contrast to previous approaches like [Ernst and Greiner 2007; Dammertz and Keller 2008], we perform splitting during tree construction instead of as a preprocess. This enables splitting decisions on a per-node basis, which in turn allows to generate additional references only when estimated ray tracing costs are improved. Our tree construction method provides a



Build Method	References	Nodes	SAH Cost	Intersections (Avg/Max)	Traversals (Avg/Max)	Overlap	Spatial Splits	Performance (Primary/AO)
Conference, 283K tris								
SBVH	125%	124805	60.6	43.2 / 309	98.7 / 402	73%	3216	122% / 125%
ESC	125%	124633	97.6	35.7 / 649	126.0 / 530	97%	–	87% / 103%
EVH	125%	124129	67.3	48.5 / 690	109.5 / 458	101%	–	104% / 102%
BVH	100%	98703	73.2	63.0 / 935	117.1 / 488	100%	–	100% / 100%



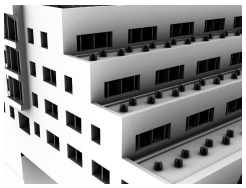
Rotated Sponza, 66K tris								
SBVH	203%	50197	86.9	102.6 / 429	176.2 / 426	57%	6058	215% / 174%
ESC	203%	49041	172.3	150.3 / 678	247.4 / 594	116%	–	139% / 114%
EVH	203%	49187	130.4	92.8 / 632	254.2 / 672	114%	–	141% / 107%
BVH	100%	24291	123.5	224.1 / 969	321.9 / 797	100%	–	100% / 100%



City, 879K tris								
SBVH	117%	363859	57.1	34.6 / 245	120.8 / 336	74%	12320	134% / 139%
ESC	117%	363421	81.5	36.0 / 356	136.9 / 451	113%	–	105% / 112%
EVH	117%	362701	71.3	37.4 / 331	149.9 / 599	118%	–	96% / 93%
BVH	100%	308643	67.1	42.3 / 366	139.9 / 443	100%	–	100% / 100%

Sibenik Cathedral, 80K tris								
SBVH	128%	35079	75.4	38.9 / 267	109.0 / 280	76%	2990	116% / 113%
ESC	128%	35125	94.1	40.9 / 249	127.1 / 330	103%	–	94% / 96%
EVH	128%	35369	93.7	39.9 / 341	126.9 / 459	125%	–	95% / 94%
BVH	100%	26993	86.1	42.3 / 321	122.6 / 311	100%	–	100% / 100%

Bubs, 1888K tris								
SBVH	103%	669113	29.8	33.2 / 374	97.9 / 476	47%	1493	126% / 103%
ESC	103%	669771	53.4	28.3 / 458	135.6 / 509	70%	–	94% / 88%
EVH	103%	668321	42.7	34.6 / 435	143.5 / 545	117%	–	95% / 92%
BVH	100%	646783	44.6	92.2 / 465	126.7 / 501	100%	–	100% / 100%



Soda Hall, 2169K tris								
SBVH	112%	812509	106.7	31.7 / 619	108.5 / 390	76%	15241	124% / 118%
ESC	112%	810869	142.2	29.2 / 616	127.2 / 390	105%	–	104% / 107%
EVH	112%	807029	127.5	33.5 / 756	122.4 / 534	107%	–	93% / 99%
BVH	100%	713499	124.2	34.1 / 636	119.9 / 443	100%	–	100% / 100%

Bar, 234K tris								
SBVH	122%	97053	62.1	39.7 / 289	123.5 / 381	79%	3351	119% / 116%
ESC	122%	96143	86.6	34.7 / 321	145.3 / 379	108%	–	107% / 102%
EVH	122%	97145	69.9	34.9 / 358	154.4 / 465	117%	–	95% / 94%
BVH	100%	77697	67.4	41.1 / 373	152.7 / 414	100%	–	100% / 100%

Table 1: ***Measurements for different build strategies.*** *The applied strategies are listed in the first column. The second column shows the relative increase of references compared to a regular BVH. The user thresholds for ESC and EVH are chosen so that the resulting amount of references is equal to that of the SBVH. This is also reflected in the final tree size (third column), which in all cases is roughly equal among the splitting based approaches. For all but one of the tested scenes, the memory overhead is less than 30%. Only in extreme cases like the Sponza Atrium (rotated around all major axes by ~45°), the number of references, and thus the total memory overhead, is higher. As can be seen in the fourth column, the SAH cost for SBVH trees is well below the other approaches in all cases. The traversal and intersection statistics (fifth and sixth column) are accumulated per pixel with one primary and four cosine-distributed ambient occlusion rays, and are averaged over a number of typical viewpoints. Although the SBVH is not always able to reduce the average number of intersection tests compared to ESC, it requires by far the fewest traversal steps. In addition, the maximum number of SBVH traversal and intersection operations is clearly below the numbers for the other methods in most cases. Similarly, the accumulated overlap surface area of child nodes in the SBVH is greatly reduced compared to the other techniques (seventh column). The eighth column lists the number of successful spatial splits used during SBVH construction. The last column shows that the reduction of ray tracing costs results in a significant performance improvement by up to a factor of two compared to a regular BVH.*

user parameter to blend between regular BVH behavior and a full SBVH, but does not leave the selection of the actual splitting threshold to the user. This makes the SBVH very practical compared to other techniques.

6.1 Future Work

We have shown that using spatial splits in a BVH can greatly improve its efficiency. However, spatial splits also remove some of the simplicity of regular BVHs. In particular, the ability to easily refit bounding volumes in dynamic scenes [Lauterbach et al. 2006; Wald et al. 2007] is lost. Finding a way to combine the two concepts presents an interesting area of future research.

Currently, our system focuses on building high-quality acceleration structures and not on efficient construction. However, a relatively low number of spatial splits is usually executed with $\alpha > 0$ (see Table 1). Thus, the additional work over a regular BVH build is fairly manageable, and SBVH construction times are not dramatically higher than for BVHs even with our unoptimized implementation. To further improve build performance, a promising avenue for future research is to investigate parallel builds, in particular on current high-end GPUs. It is likely that some of the previously published parallelization principles can be applied with relatively little modification [Zhou et al. 2008; Lauterbach et al. 2009]. More generally, there is a number of readily available efficient construction schemes for both BVHs and kD-trees, many of which have the potential to work well in our setting for generating the individual split candidates.

The current SBVH algorithm provides reasonably good control over the reference duplication rate and the resulting memory footprint. More importantly, it is guaranteed that no excessive splitting takes place, since the user parameter is not directly related to a split threshold. However, an interesting area for future work would be to investigate SBVH variants with a hard memory limit. A simple approach would be a breadth-first construction scheme, which disallows the use of spatial splits once a certain memory threshold has been reached. A more advanced solution could attempt an in-place build, e.g. by combining it with techniques discussed in [Wächter and Keller 2007].

Finally, one can reasonably assume that SBVH hierarchies work well not only in ray tracing scenarios. Other applications, such as collision detection, are likely to benefit from improved hierarchy quality as well.

Acknowledgements

The authors would like to thank their colleagues at NVIDIA for discussions and the anonymous reviewers for their comments. Thanks to Ryan Vance for the Bubs scene and Guillermo M. Leal Llaguno as well as Cornell University for the bar scene.

References

BENTLEY, J. L. 1975. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM 18*, 9, 509–517.

DAMMERTZ, H., AND KELLER, A. 2008. The Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance. *IEEE Symposium on Interactive Ray Tracing 2008*, 155–158.

ERNST, M., AND GREINER, G. 2007. Early Split Clipping for Bounding Volume Hierarchies. *Proceedings Eurographics/IEEE Symposium on Interactive Ray Tracing 2007*, 73–78.

GLASSNER, A. S. 1988. Spacetime Ray Tracing for Animation. *IEEE Comput. Graph. Appl. 8*, 2, 60–70.

GOLDSMITH, J., AND SALMON, J. 1987. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications 7*, 5 (May), 14–20.

HAVRAN, V., HERZOG, R., AND SEIDEL, H.-P. 2006. On the Fast Construction of Spatial Data Structures for Ray Tracing. 71–80.

HAVRAN, V. 2000. *Heuristic Ray Shooting Algorithms*. Ph.D. Thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague.

HUNT, W., MARK, W. R., AND STOLL, G. 2006. Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing*, IEEE.

LAUTERBACH, C., EUI YOON, S., AND MANOCHA, D. 2006. RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs. In *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 39–45.

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2009. Fast BVH Construction on GPUs. *Computer Graphics Forum 28*, 2, 375–384.

LUEBKE, D., AND PARKER, S. 2008. Interactive Ray Tracing with CUDA. *NVIDIA Technical Presentation, SIGGRAPH 2008*.

MACDONALD, J. D., AND BOOTH, K. S. 1989. Heuristics for Ray Tracing using Space Subdivision. In *Proceedings of Graphics Interface 1998*, 152–163.

POPOV, S., GÜNTHER, J., SEIDEL, H.-P., AND SLUSALLEK, P. 2006. Experiences with Streaming Construction of SAH KD-Trees. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, 89–94.

SHEVTSOV, M., SOUPIKOV, A., AND KAPUSTIN, A. 2007. Highly Parallel Fast KD-tree Construction for Interactive Ray Tracing of Dynamic Scenes. *Computer Graphics Forum 26*, 3 (September), 395–404.

SOUPIKOV, A., SHEVTSOV, M., AND KAPUSTIN, A. 2008. Improving KD-tree Quality at a Reasonable Construction Cost. In *Proceedings of the 2008 IEEE Symposium on Interactive Ray Tracing*.

SUNG, K., AND SHIRLEY, P. 1992. Ray Tracing with the BSP-Tree. In *Graphics Gems III*. Academic Press, 271–274.

WÄCHTER, C., AND KELLER, A. 2007. Terminating Spatial Hierarchies by A Priori Bounding Memory.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26*, 1.

WALD, I. 2007. On fast Construction of SAH based Bounding Volume Hierarchies. In *Proceedings of the 2007 Eurographics/IEEE Symposium on Interactive Ray Tracing*.

ZHOU, K., HOU, Q., WANG, R., AND GUO, B. 2008. Real-time KD-tree Construction on Graphics Hardware. *ACM Trans. Graph. 27*, 5, 1–11.