Appears in the High Performance Graphics 2009 conference proceedings

Object Partitioning Considered Harmful: Space Subdivision for BVHs

Stefan Popov* Saarland University
Iliyan Georgiev† Saarland University
Rossen Dimov‡ Saarland University
Philipp Slusallek§ DFKI Saarbrücken, Saarland University

Abstract

A major factor for the efficiency of ray tracing is the use of good acceleration structures. Recently, bounding volume hierarchies (BVHs) have become the preferred acceleration structures, due to their competitive performance and greater flexibility compared to KD trees. In this paper, we present a study on algorithms for the construction of optimal BVHs.

Due to the exponential nature of the problem, constructing optimal BVHs for ray tracing remains an open topic. By exploiting the linearity of the surface area heuristic (SAH), we develop an algorithm that can find optimal partitions in polynomial time. We further generalize this algorithm and show that every SAH-based KD tree or BVH construction algorithm is a special case of the generic algorithm.

Based on a number of experiments with the generic algorithm, we conclude that the assumption of non-terminating rays in the surface area cost model becomes a major obstacle for using the full potential of BVHs. We also observe that enforcing space subdivision helps to improve BVH performance. Finally, we develop a simple space partitioning algorithm for building efficient BVHs.

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, acceleration structures, bounding volume hierarchies, construction, surface area heuristic

1 Introduction

In the recent years, the demand for interactive ray tracing of dynamic scenes has lead to a wave of research on fast construction of acceleration structures. Various algorithms have been proposed, most of which try to trade some rendering performance for faster construction. This research has also lead to a shift in the general understanding of acceleration structure efficiency: BVHs have become the preferred structure for numerous applications. Even though generally slower in rendering than KD trees, BVHs have proved to be well suited for dynamic scenes, due to their ability to handle changes in geometry while maintaining their topology. BVHs also generally have predictable size, they are smaller and shallower than KD trees, and can be built efficiently. Recent research on packet based traversal has also shown that BVHs can have competitive rendering performance to KD trees [Wald et al. 2007]. However, KD trees still remain optimal for small ray packets and static scenes.

With a few exceptions, e.g. triangle pre-splitting [Ernst and Greiner 2007; Dammertz and Keller 2008] and tree post-processing [Kensler 2008], most research on BVH construction in the last years has concentrated on lower build times. In contrast, in this paper we try to exploit the full potential of BVHs for high-performance ray tracing by improving the construction algorithm, while tolerating longer construction times.

The classic top-down construction algorithm for BVHs, which is also the one that currently gives the best rendering performance, uses sweep plane partitioning. It has been adopted from KD trees and approximates primitives by the centroids of their bounding boxes. This approach considers only a fraction of all possible partitions for a given set of objects, since BVHs allow for simultaneous subdivision along multiple dimensions, including spatial nesting of sibling nodes.

In this paper, we present a theoretical framework for BVH construction that explores all possible ways of partitioning a set geometrically and runs in polynomial time. The framework can perform triangle splitting during construction in a way consistent with the cost function, as opposed to previous heuristic-based pre-splitting methods. We explore different building strategies based on this framework and evaluate their impact on rendering performance. Finally, based on experimental results, we develop a simple and relatively fast SAH-based algorithm, similar to KD tree construction, that produces trees that perform best on most of the tested scenes.

We also analyze and discuss the surface area cost model and the SAH. Even though considered good and working well in practice, we experimentally show that the SAH is not optimal for fully exploiting the capabilities of BVHs, as it does not explicitly account for early ray termination.

2 Background and Previous Work

In this chapter we give a brief overview of acceleration structures, cost models, and construction methods often used in ray tracing, and summarize relevant previous work.

2.1 Acceleration Structures

For the needs of this paper, we consider two commonly used types of acceleration structures for ray tracing: the KD tree [Bentley 1975] and the BVH [Arvo and Kirk 1989]. A study of other acceleration structures can be found in [Havran 2000]. These, however, fall beyond the scope of this paper.

The KD tree is a binary space partitioning tree, with splitting planes aligned with the three major axes. KD trees are always constructed in a top-down manner, and construction algorithms differ mainly in the way they choose the splitting plane that separates the children of a node. Popular split choices include the spatial and object medians, as well as planes chosen according to the surface area heuristic (SAH) [MacDonald and Booth 1990].

*e-mail:popov@cs.uni-saarland.de
†e-mail:georgiev@cs.uni-saarland.de
‡e-mail:rdimov@graphics.cs.uni-sb.de
§e-mail:slusallek@dfki.de

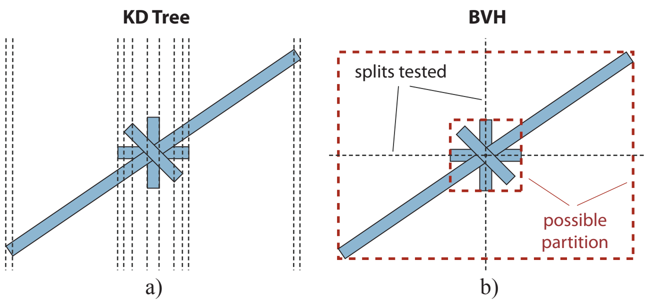

Figure 1: *An extreme case where the centroids of all objects in a node coincide. While KD tree construction will be able to partition the set, classic BVH construction will have to create a leaf node, even though it would be more optimal to create two children nested in each other.*

A typical bounding volume hierarchy (BVH) is a binary tree, each node of which contains an axis-aligned bounding box (AABB), enclosing the AABBs of its children. The AABB of a leaf encloses the primitives it contains. In general, BVHs can have arbitrary arity, e.g. 4 or 16 [Dammertz et al. 2008; Wald et al. 2008], and can use any convex volume instead of AABB, but such BVHs are less often used in practice. BVHs usually do not share primitives among their leaves, which makes their size predictable in advance. However, it has been recently shown that sharing primitives [Ernst and Greiner 2007] or splitting them in advance [Dammertz and Keller 2008] can be beneficial for rendering performance. BVHs are usually constructed in a top-down or bottom up manner, but other methods exist as well [Lauterbach et al. 2008; Walter et al. 2008].

The traversal algorithms for BVHs and KD trees, even though different, share a common pattern. A node is traversed by ordering its children along the ray and labeling them as near and far. An intersection is recursively searched for in the near child first. Upon return, if the best found intersection so far occurs after the entry of the ray in the far child, the latter is also processed recursively. If a child is not hit by the ray at all, it is not traversed. A leaf is traversed by intersecting all contained primitives with the ray. A good reference on traversal algorithms can be found in [Havran 2000].

2.2 The Surface Area Cost Model

The ray tracing performance of a tree can be estimated using the surface area cost model [MacDonald and Booth 1990]. The expected cost of a tree T is

$$Exp(T) = C_T \sum_{n \in N} P(n|T) + C_I \sum_{l \in L} P(l|T)|l| \qquad (1)$$

where N is the set of all inner nodes of the tree, L is the set of all leaves, $P(.|T)$ is the geometric probability of a random ray hitting a node/leaf given that it hits the root node of the tree, $|l|$ is the number of primitives in the leaf l, and C_T and C_I are the traversal and intersection costs respectively. Assuming that the shapes of node and leaf bounding volumes are convex, $P(.|T)$ can be expressed as the ratio of the surface area of the node/leaf to the surface area of the root node. Thus $P(.|T) = SA(.)/SA(T)$.

$Exp(T)$ gives the expected cost for traversing the tree with a uniformly randomly chosen ray. It assumes that the ray does not hit any primitive and processes all nodes and primitives it encounters along its path. Even though this assumption rarely holds in practice, results show that there is a strong correlation between $Exp(T)$ and traversal performance [Havran 2000]. Thus, construction algorithms aim at minimizing $Exp(T)$.

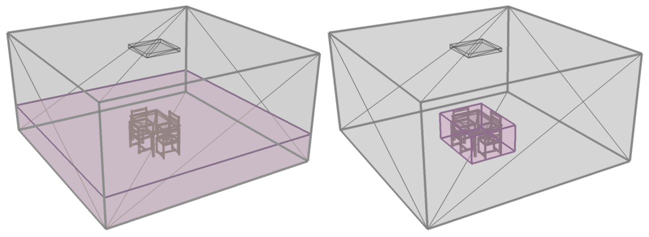

Figure 2: *A partition of the root node of the* SHIRLEY6 *scene produced by the classic BVH partitioning algorithm (left) and our geometric partitioner (right). Note that the classic algorithm has also created nested siblings.*

For binary trees, Equation 1 can be expressed recursively:

$$Exp(N) = \begin{cases} C_T + P(N_L|N)E(N_L) + P(N_R|N)E(N_R) & ,node \\ C_I\ |N| & ,leaf \end{cases} \qquad (2)$$

where N_L and N_R are the left and right children of N, and $|N|$ is the number of primitives in N. It is easy to see that $Exp(T) = Exp(Root(T))$. Again, $P(.|N) = SA(.)/SA(N)$, assuming that the nodes and leaves of the tree have convex shape.

A tree that minimizes $Exp(T)$ can be constructed in a top-down manner by splitting the set of primitives and minimizing (2) at each node. Since the cost of a subtree is not known during construction, $Exp(N_L)$ and $Exp(N_R)$ are usually approximated using the number of primitives in the respective sets. This corresponds to the worst possible case (and thus cost), i.e. when the construction algorithm creates leaves from the left and right subtrees. This approximation is know as the surface area heuristic, or SAH [MacDonald and Booth 1990]. Thus, instead of minimizing (2) for a non-leaf node, construction algorithms minimize

$$E(N) = C_T + C_I\left(P(N_L|N)\ |N_L| + P(N_R|N)\ |N_R|\right). \qquad (3)$$

Even though it is possible to construct BVHs with non top-down approaches, practice has shown that trees built with such methods perform worse than the ones built in a top-down fashion and according to the SAH.

2.3 Classic BVH Construction

Existing BVH construction algorithms do not allow sharing of primitives among leaves. Thus, their task boils down to partitioning the set of primitives of each node into two disjoint subsets that minimize an objective function – the SAH. Techniques that allow sharing of primitives add an initial (pre-splitting) step to the construction algorithm, executed *before* construction starts [Ernst and Greiner 2007; Dammertz and Keller 2008].

More formally, at a node N that will enclose the set of primitives S, the BVH builder has to partition S into $S_L \oplus S_R = S$, such that it minimizes the SAH:

$$E(N) = C_T + C_I \frac{SA(S_L)|S_L| + SA(S_R)|S_R|}{SA(N)} \qquad (4)$$

where $SA(S)$ denotes the surface area of the tight AABB around all primitives in S. The builder then continues by recursively creating the left and right children of N (N_L and N_R) from the sets S_L and S_R respectively.

Since S can be partitioned in $2^{|S|}$ ways, finding the optimal one is NP hard in the general case. To overcome this, the classic BVH

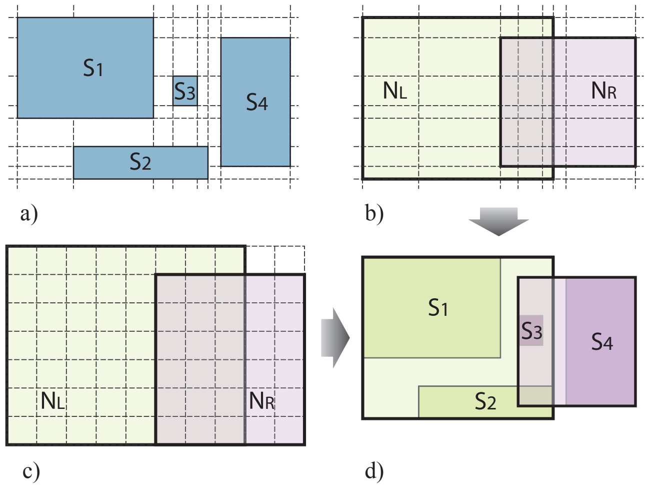

Figure 3: *Geometric partitioning. a) The set of objects. b) Candidate AABBs picked at object boundaries. c) Candidate AABBs picked on a grid. d) Objects fully contained in both AABBs are put into the one with smaller surface area, and bounds are tightened.*

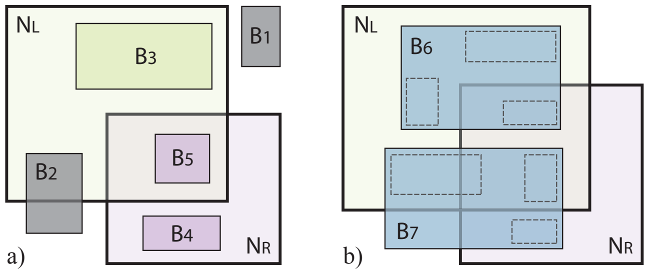

Figure 4: *Accept and reject tests for the cost calculation. a) Nodes B_1 and B_2 would reject the configuration. All primitives of B_3 are put into N_L and the primitives of B_4 – into N_R. B_5 is put into the child with smaller surface area (N_R in this case). b) Undecided cases: the nodes of $B6$ and $B7$ cannot be uniquely assigned neither to N_L nor to N_R. The algorithm needs to refine them recursively.*

construction algorithm [Goldsmith and Salmon 1987] performs a plane sweep partitioning, much like KD tree construction algorithms. Furthermore, it assumes finely tessellated geometry and approximates the primitives with points, using the centers of their AABBs (centroids). These centroids are chosen as interesting split points (events), each giving a potential partition $S = S_L \oplus S_R$. A primitive goes into S_L iff its centroid is to the left of the event. The algorithm keeps track of the left and right subsets and their AABBs at each event and updates them incrementally in $O(1)$ time. At each event, the SAH cost is evaluated and finally S_L and S_R are created at the event with minimal cost.

Based on this scheme, a number of BVH construction algorithms have been proposed that speed up construction by using some form of SAH approximation [Günther et al. 2007; Wald 2007]. Attempts have also been made to improve tree quality by using stochastic search methods [Ng and Trifonov 2003] and by post-processing the constructed trees [Kensler 2008]. Such methods, however, have achieved only marginal performance improvements.

3 Geometric Partitioning

The method presented in this section has been originally motivated by the rather heuristic approach for partition search in the classic construction algorithm, which can miss partitions with significantly lower cost (see Figure 1). Our new geometric partitioning algorithm tries to explore the whole space of possible child AABB arrangements of a node.

3.1 From NP Complete to Polynomial

The exponential number of possible ways to partition a set S makes it prohibitively expensive to search for the best partition in a brute-force manner even for small scenes. However, if a primitive overlaps both children of a node, it should go to the child with the smaller probability (i.e. the one with smaller surface area) according to the SAH. While this observation is SAH-specific, our intuition is that it will hold for other cost functions as well.

According to the above observation, if we know the AABBs of the children, we know exactly how to form S_L and S_R. This also gives us the basis for an alternative partitioning algorithm: Instead of testing all $2^{|S|}$ possibilities of partitioning S, we can test all possible configurations of child AABBs and form S_L and S_R for each configuration, in order to calculate the cost. Additionally, each configuration should be tested for feasibility, i.e. if the AABB of each primitive in S is contained in at least one child AABB.

Since we do not allow primitives to be split, the events where the child AABBs can start and end are defined at the boundaries of the AABBs of the primitives (see Figure 3b). Thus, the number of possible child AABB configurations is $O(|S|^{12})$.

The number of considered configurations can be further reduced with the observation that each side of the AABB of the parent is shared by at least one child AABB. Taking symmetry into account, there are exactly $2^5|S|^6$ ways to choose the child AABB configurations. Thus, partitioning has a complexity of $O(|S|^6 Q)$, where Q is the complexity of the cost and feasibility estimation. The time for partitioning the root node dominates the construction time and thus the construction algorithm has an overall complexity of $O(N^6 Q)$ for N primitives.

3.2 Cost and Feasibility of a Configuration

When estimating the cost and feasibility of a configuration, instead of touching each primitive, we use an auxiliary BVH over the primitives in S. This BVH is built using a centroid-based split in the middle approach. Each node of the auxiliary BVH stores the tight bounds of its children as well as the count of primitives in the subtree rooted at the node. Its construction takes $O(|S| \log |S|)$ time and does not impact the overall complexity of the algorithm.

Starting from the root of the auxiliary BVH, each traversed node N is tested against a number of trivial accept and reject tests, as illustrated in Figure 4. If N is trivially rejected, the configuration is declared as infeasible. If N passes an accept test, one of the counters ($|S_L|$ or $|S_R|$) is increased with the number of primitives in the subtree below N. If N was neither rejected nor accepted, its children are processed recursively. Once in a leaf, the contained primitives are tested, and either all primitives are accepted (and $|S_L|$ and $|S_R|$ accordingly updated) or the whole configuration is rejected. Our empirical tests show that the described algorithm performs a query in $O(\sqrt{|S|})$ time on average.

3.3 A Grid Approximation

Even though the above presented algorithm has a polynomial run time, it is still prohibitively slow for non-trivial scenes. To make it usable in practice, we use an approximation over the configuration search space: we pick the AABB events on a regular grid instead on the boundaries of the primitives (see Figure 3c).

By changing the resolution of the grid we can to control the complexity of the algorithm. For practical reasons, we aim at a run time of $O(N^{1.5})$. To achieve that, we limit the number of grid cells to $K_R\sqrt{|S|}$, with $K_R = const$ controlling the grid resolution. For the actual resolution $R_X \times R_Y \times R_Z$ of the grid we run a binary search algorithm that determines these values so that $R_X R_Y R_Z \approx K_R\sqrt{|S|}$ and $R_X : R_Y : R_Z \approx D_X : D_Y : D_Z$, with $D_{X|Y|Z}$ being the size of the AABB of the current node.

3.4 Results and Discussion

We implemented the geometric partitioning algorithm in the RTfact real-time ray tracing framework [Georgiev and Slusallek 2008]. We performed tests on the Bunny, Sponza, Fairy Forest, Conference, Venice, and Soda Hall scenes (Figure 8) and compared it to the classic BVH construction. We were primarily interested in the rendering performance for random rays and primary rays with different packet sizes. We additionally measured the global expected cost and the average ray cost (function of the average number of traversal and intersection steps) for random rays. Random rays could start at arbitrary locations inside the scene and were traced using a single ray traversal algorithm. For primary rays, we used the following traversal algorithms: single ray, 4-wide packet, range packet (introduced by [Wald et al. 2007] and discussed in [Overbeck et al. 2008]), and partition packet traversal (introduced by [Overbeck et al. 2008]). All benchmarks in this paper trace 1 million rays per frame, and all measurements have been gathered on a single core of a Intel Core2 2.6GHz processor. Table 3 summarizes the results for the geometric partitioner.

To our surprise, the performance and expected cost of the trees produced by the new construction algorithm were actually worse, even though we modified it to also consider classic BVH partitions. The better SAH cost partitions found by the geometric partitioner had actually resulted in lower ray tracing performance.

Suspecting that SAH might be the cause for the unexpected results, we changed our algorithm to evaluate the cost for each configuration by building the left and right subtrees using a split-in-the-middle approach and taking their expected cost Exp (Equation 1). The results of the modified algorithm on Sponza are summarized in Table 1. The produces trees indeed had better expected cost but their rendering performance (proportional to Cost$_{ray}$ in the table) had remained poor. Similar results have been reported in [Ng and Trifonov 2003; Kensler 2008]. Using a modified traversal algorithm, with non-terminating rays starting at the scene boundaries, we managed to achieve the expected correlation of Exp and ray tracing performance.

	Early ray termination	Centroid sweep	Grid w/ recursive cost evaluation
$Exp(T)$	-	142.6	124.0
Traversal	yes	62.5	81.0
Intersection	yes	15.2	8.0
Cost$_{ray}$	yes	85.3	93.0
Traversal	no	122.3	137.4
Intersection	no	48.7	26.7
Cost$_{ray}$	no	195.3	177.5

Table 1: *Performance of* Sponza *with recursive cost evaluation vs. classic construction. The tree cost does not correlate to the rendering performance if the rays terminate.*

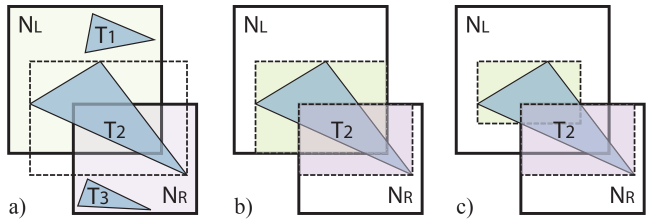

Figure 5: *Triangle splitting.* ***a)*** *A configuration is feasible iff the geometry is covered by the children's AABBs.* ***b)*** *A triangle can be split by clipping it to the child AABBs or* ***c)*** *by clipping it to the child with smaller area and putting what remains in the other.*

We also performed tests on the Shirley6 scene, where the geometric partitioner managed to actually produce a better performing tree than the classic construction algorithm. Due to the geometric symmetry in this scene, the classic algorithm produced two siblings contained in each other, resulting in significant overlap (see Figure 2). The geometric partitioner managed to find a partition with lower expected cost and better performance. Extreme cases like this, however, occur rarely in most scenes.

Up to this point, in contrast to the common belief, our experiments had confirmed that: 1) minimizing SAH locally can have noticeable adverse effects on Exp, and 2) minimizing Exp does not by itself guarantee better rendering performance. On the other hand, with the exception of [Ng and Trifonov 2003; Kensler 2008], all other SAH based construction algorithms do not seem to have the above two problems. We also noticed that while those algorithms operate in a space subdivision manner, ours does not.

Considering the above observations, we form our main hypothesis: to achieve good practical results, a construction algorithm should not only aim at minimizing the SAH cost, but also at *space subdivision*. Our intuition is that better space subdivision increases the chances of a ray to terminate early, which works around the unrealistic assumption in the cost model that rays will miss all geometry.

At this point we could either continue using our algorithm and design an adequate cost function that accounts for early termination, or we could try modifying our algorithm to enforce better space subdivision. We decided to go with the second option, as we already had some intuition how to achieve it.

4 A Generic Construction Algorithm

The classic construction algorithm already does a good job in separating the children of a node spatially, given that it can not split primitives. In order to be able to further study the partitioning problem, we decided to introduce primitive splitting in our framework and thus to further refine the search space. We also modified the cost function, in order to be able to control space subdivision. The result is an algorithm that generalizes all SAH-based KD tree and BVH construction algorithms.

4.1 Primitive Splitting

Up to this point, we have considered a configuration feasible, if each primitive is fully contained in at least one of the child AABBs. We now relax this condition and require each primitive to be fully contained in the union volume of the child AABBs, and allow primitives to be split (see Figure 5). To calculate the SAH cost of a fixed configuration, we count each split primitive twice – once in the left and once in the right child.

The actual splitting can be done in more than one ways. For triangles, we do it by clipping each triangle with both child AABBs independently. Alternatively, triangles could be clipped against the AABB of the child with smaller probability and the remaining geometry stored in the other child (see Figure 5). Although this will not affect the SAH cost immediately, it might reduce it in descendent nodes. However, numerical instabilities prevented us from using this method.

4.2 Defining the Search Space

When primitive splitting is allowed, the AABBs of a configuration are not anymore constrained to the primitive AABB boundaries and can be chosen arbitrarily inside the parent AABB. To explore this continuous space we approximate it by a uniform grid. Note that with a fine enough resolution the grid will cover the search space of all known construction algorithms, including classic BVH construction, our geometric partitioner, and the KD tree construction algorithm.

For practical reasons, we use moderate grid resolutions and artificially augment the search space with the configurations considered by other construction algorithms – namely KD tree construction and classic BVH construction.

4.3 Controlling Space Subdivision

As discussed in Section 3.4, the correlation between tree cost and ray tracing performance breaks when nodes can overlap arbitrarily. Therefore, we modify Equation 4 by adding a term that can bias the cost of a node depending on how much its children overlap:

$$E = C_T + \left(C_O \frac{V(N_L \cap N_R)}{V(N)} + 1 \right) \cdot \cdot C_I \frac{SA(S_L)|S_L| + SA(S_R)|S_R|}{SA(N)} \tag{5}$$

where C_O is a parameter that controls the overlap penalty and $V(B)$ is the volume of a box B. Choosing C_O large enough enforces strict space subdivision, while $C_O = 0$ results in no overlap penalty.

4.4 The Algorithm

With the help of the theory presented in the above sections, we design a generic BVH construction algorithm.

Similarly to the classic construction, the generic algorithm first finds a partition with minimal cost and then sifts the primitives into the left and right children accordingly. Partition searching is decomposed into a configuration oracle and a cost calculator (Algorithm 1). The oracle generates child AABB configurations, while the cost calculator estimates the feasibility of each configuration and the cost of the resulting partition. Sifting is performed like in standard BVH construction, with the addition of primitive splitting and tightening of the bounds fixed by the oracle.

Partition searching is parametrized by the oracle O and the overlap penalty P. By changing the oracle, we can simulate different construction strategies. Oracles can be combined by merging their search spaces.

In a sense, this is the most optimal SAH-based construction algorithm for binary tree acceleration structures. With a fine enough grid, our algorithm will cover the configuration space of every other top-down SAH based approach. However, for the same configuration, our algorithm will always find a better partition w.r.t. SAH.

Algorithm 1 The generic algorithm: partition searching.

```
1: function FINDOPTIMALPARTITION($O, P$)
                         ▷ $O \equiv$ AABB Oracle, $P \equiv$ Overlap Penalty
2:     for all $C \in O$ do          ▷ $C \equiv$ Current Configuration
3:         $cost \leftarrow$ GETCOSTANDFEASIBILITY($C, P$)
4:         if ($C$ is feasible) and ($cost < cost_{best}$) then
5:             $C_{best} \leftarrow C$, $cost_{best} \leftarrow cost$
6:         end if
7:     end for
8:     return CREATEPARTITION($C_{best}$)
9: end function
```

4.5 Results and Discussion

We performed a number of experiments using the generic construction algorithm on the following scenes: BUNNY, SPONZA, FAIRY FOREST, CONFERENCE, VENICE, and SODA HALL. We parameterized the algorithm with different overlap penalties and used combinations of three basic oracles: $CentroidGen$ generates the same AABBs as the classic BVH construction; $KDNodeGen$ – AABBs considered in KD tree construction; $GridGen$ – AABBs with vertices fixed on a regular grid. The results are summarized in Table 4.

We used three setups of the generic algorithm that resulted in different construction strategies. The first setup, named *C-BVH*, has parameters $P = 0$ and $O =$ *CentroidGen*. It corresponds to classic construction and applies the same partitioning logic. The second, named *GK-BVH*, has a parameter $O =$ *KDNodeGen*, which mimics KD tree construction. The value of P does not influence the cost in this case. The third setup, *G-BVH*, has parameters $P = 10^{16}$ and $O = \{$*CentroidGen* $\cup$ *KDNodeGen* $\cup$ *GridGen*$\}$, which allows it to perform a more extended search. Both *GK-BVH* and *G-BVH* enforce strict space subdivision.

For comparison, we also built a KD tree for each scene and converted it to a BVH by removing the empty leaves and refitting the AABBs of the nodes.

Our results show that BVHs built with strict space subdivision perform consistently faster for small ray packets on all scenes. Such trees are generally deeper and facilitate early ray termination.

On scenes with low depth and/or geometric complexity, *GK-BVH* trees perform similarly or worse than *C-BVH* trees with large coherent packets, most notably with the range intersector. One reason for this is that the range intersector relies much more on coherent traversal than on early ray termination. Since *GK-BVH* trees tend to be deeper and to reduce the amount of primitive intersections for single rays, they provide less coherency for ray packets in the lower levels. However, on more complex scenes with high depth or geometric complexity (e.g., VENICE and SODA HALL), even wide packet traversal algorithms benefit from stricter space subdivision.

We also performed experiments with different values for the overlap penalty P. Unsurprisingly, trees built with $P = 0$ had very similar performance to the ones reported in Table 3. Increasing the parameter gradually improved the trees, until they reached the performance of *G-BVH* (see Figure 6). We did not observe any noticeable difference for values of P larger than 10^3. Another parameter we experimented with was the grid resolution K_R for the smaller scenes. Increasing K_R beyond 64 resulted in minor improvements,

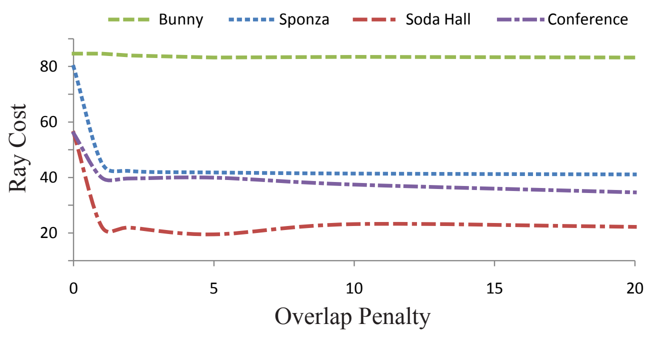

Figure 6: *Impact of the overlap penalty on the ray cost for G-BVH. Most scenes benefit from stricter space subdivision. The penalty has little influence on scenes with uniform geometry distribution.*

while exploding construction times. We do not include the measurements in this paper.

It can also be noted that *G-BVH* trees perform on par with or only marginally better than *GK-BVH* trees, while *G-BVH* construction times can be orders of magnitude larger, depending on the grid resolution. Additionally, BVHs converted from KD trees (*KD-BVH*) tend to be larger in size and generally perform worse than *GK-BVH*. Thus, taking size, construction time, and performance into account, we can conclude that *GK-BVH* performs best for moderately to highly complex scenes.

5 A Spatial Split Construction Algorithm

Based on the results from Section 4.5, we developed a simple BVH construction algorithm that performs space subdivision. It mimics the behavior of the generic algorithm, but with search space restricted to KD tree events (i.e. *GK-BVH*), as they produced optimal BVHs in the experiments from Table 4.

The new construction algorithm performs axis aligned sweep plane partitioning, and takes as events the boundaries of the primitive AABBs along each axis. Each position of the sweep plane uniquely defines the AABBs of the left and right children. During the plane sweep, the algorithm incrementally updates the counts of the primitives that go to the left, right, and both children. Also similar to KD tree construction, the algorithm calculates the SAH cost at each event and takes the best split among all three dimensions. During sifting, the tight bounds of the left and right children are incrementally computed. This allows the BVH to cut empty space faster than a KD tree. The complexity of the algorithm is $O(N \log N)$ when implemented according to [Wald and Havran 2006].

5.1 Results and Discussion

The spatial split algorithm (*SS-BVH*) produces trees nearly identical to *GK-BVH* (see Section 4.5). However, its implementation is much simpler and more numerically stable. The results for *GK-BVH* in Table 4 also apply to *SS-BVH*.

Finally, we compared *SS-BVH* to early split clipping (*ESC-BVH*) [Ernst and Greiner 2007] and the edge volume heuristic (*EVH-BVH*) [Dammertz and Keller 2008] on two variants of SPONZA: the original one and one rotated about the major axes.

As can be noted from Table 2, the pre-split algorithms cannot improve the BVH cost and performance on the original scene. In contrast, *SS-BVH* increases performance by more than 50%.

Scene	Type	Size	$Exp(T)$	Cost$_{ray}$	FPS
SPONZA	C-BVH	1.0 MB	142	85	0.60
original	EVH-BVH$_{t=14}$	1.0 MB	142	85	0.60
	EVH-BVH$_{t=16}$	1.0 MB	142	85	0.60
	ESC-BVH$_{80}$	1.1 MB	185	103	0.50
	ESC-BVH$_{200}$	1.1 MB	167	96	0.55
	G-BVH	2.6 MB	121	63	0.95
	SS-BVH	2.8 MB	119	63	0.96
SPONZA	C-BVH	1.0 MB	144	396	0.11
rotated	EVH-BVH$_{t=14}$	1.3 MB	148	190	0.25
	EVH-BVH$_{t=16}$	1.6 MB	154	170	0.28
	ESC-BVH$_{80}$	1.3 MB	155	179	0.29
	ESC-BVH$_{200}$	1.2 MB	147	196	0.26
	G-BVH	11.4 MB	86	100	0.59
	SS-BVH	1.5 MB	134	199	0.24
	SS-BVH	2.6 MB	120	175	0.27
	SS-BVH	4.3 MB	101	134	0.36
	SS-BVH	7.7 MB	89	105	0.54
	SS-BVH	10.3 MB	87	100	0.57

Table 2: *Comparison of our construction algorithms to pre-split methods for single primary rays on two variations of* SPONZA. *Our algorithms achieve larger speed-up on both scenes, and pre-split methods only help for non-axis-aligned geometry. The size and quality of SS-BVH can be controlled by the termination criterion.*

On the rotated SPONZA, no choice of parameters for the pre-splitting methods can increase performance by more than a factor of 3 over *C-BVH*. Depending on the termination criterion, *SS-BVH* can achieve up to 6 times speed-up, and for the same size, tree performance is comparable to that of pre-splitting methods.

SS-BVH achieves construction times between the ones of classic BVHs and KD trees. These can be further improved using approximation schemes like [Popov et al. 2006; Hunt et al. 2006].

A further natural extension is to combine the algorithm with classic BVH construction. By choosing the best partition from either schemes at each step, a trade-off between tree size and space subdivision can be made. Additionally, this trade-off can be controlled by penalizing the overlap.

5.2 Improving Cost Estimation

One disadvantage of the spatial split algorithm is that it does not maintain tight child bounds during the plane sweep. This is the correct behavior for KD tree construction but not for BVHs which store tight bounds in their nodes. Thus, in order to compute the correct probabilities, the construction algorithm must keep track of the tight AABBs around the primitive subsets at each event.

An efficient solution is to maintain a priority queue, which stores the primitives currently intersected by the sweep plane, sorted by their end event. During the plane sweep, the queue is updated by removing the primitives that end at the current position and by inserting the ones that begin there. For the actual accumulation of the left and right AABBs at each event, a two pass sweep plane scheme (in increasing and decreasing directions) can be used, like in [Wald et al. 2007]. At each step of the sweep, AABBs are extended with the primitives that leave the split queue, as well as the part of each primitive in the queue that lies to the left, respectively right, of the current plane position.

A further issue with computing tight AABBs during the plane sweep can be the right choice of events. KD tree construction algorithms take the boundaries of the primitive AABBs, as the cost function has been proven to be linear between such two events [Havran 2000]. This is not true if keeping tight bounding boxes during the sweep (Figure 7).

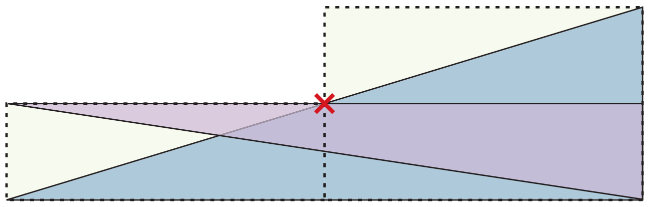

Figure 7: *Taking an additional event through the red cross will result in a split with SAH cost* 5.5 *for* $C_I = 3$ *and* $C_T = 1$*. The spatial build algorithm will not consider this event and will create a leaf with cost* 6.

It can be proven that for BVHs the SAH cost function is piece-wise quadratic between two primitive AABB events. The discontinuity points and the coefficients of the function between them can be determined from the primitives in the split queue with an algorithm similar to line segment intersections [de Berg et al. 2000]. Thus, the global SAH minimum can also be computed analytically.

6 Conclusion and Future Work

In this paper, we have presented a study on construction algorithms that aim at improving the cost and rendering performance of BVHs. We have developed a theoretical framework for searching an optimal geometric partition of a set of primitives in polynomial time. Based on the obtained results, we formed a hypothesis that the SAH works well only for hierarchical structures that maintain space subdivision.

We have then developed a generic BVH construction algorithm, which can in theory find an optimal partition of the set of primitives at each node. The algorithm can split primitives consistently with the cost function and can control the space subdivision. We have shown that all other known SAH based construction algorithms for BVHs and KD trees can be considered special cases of the generic algorithm.

Based on many experiments with the generic algorithm, we have concluded that the optimal partition strategy for most scenes is to perform space subdivision. Finally, we have developed a simple and robust SAH-based BVH construction algorithm that creates efficient trees.

Throughout the paper we point out many directions for future work. We believe that the most relevant open problem remains the development of a better cost model for tree construction that can account for early ray termination. This way, explicit enforcement of space subdivision would no longer be needed, and more optimal trees could be obtained w.r.t. size and ray tracing performance.

References

ARVO, J., AND KIRK, D. 1989. A Survey of Ray Tracing Acceleration Techniques. *An Introduction to Ray Tracing*, 201–262.

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

DAMMERTZ, H., AND KELLER, A. 2008. Edge Volume Heuristic – Robust Triangle Subdivision for Improved BVH Performance. In *IEEE/Eurographics Symposium on Interactive Ray Tracing*.

DAMMERTZ, H., HANIKA, J., AND KELLER, A. 2008. Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. *Computer Graphics Forum 27*, 4 (jun).

DE BERG, M., VAN KREVELD, M., OVERMARS, M., AND SCHWARZKOPF, O. 2000. *Computational Geometry: Algorithms and Applications*, second ed. Springer-Verlag.

ERNST, M., AND GREINER, G. 2007. Early Split Clipping for Bounding Volume Hierarchies. *Symposium on Interactive Ray Tracing 0*, 73–78.

GEORGIEV, I., AND SLUSALLEK, P. 2008. RTfact: Generic Concepts for Flexible and High Performance Ray Tracing. In *IEEE/Eurographics Symposium on Interactive Ray Tracing*.

GOLDSMITH, J., AND SALMON, J. 1987. Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Comput. Graph. Appl. 7*, 5, 14–20.

GÜNTHER, J., POPOV, S., SEIDEL, H.-P., AND SLUSALLEK, P. 2007. Realtime Ray Tracing on GPU with BVH-based Packet Traversal. In *Proceedings of the IEEE/Eurographics Symposium on Interactive Ray Tracing 2007*, 113–118.

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.

KENSLER, A. 2008. Tree Rotations for Improving Bounding Volume Hierarchies. In *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, 73–76.

LAUTERBACH, C., GARLAND, M., SENGUPTA, S., LUEBKE, D., AND MANOCHA, D. 2008. Fast BVH Construction on GPUs. In *(Proceedings of Eurographics)*.

MACDONALD, D. J., AND BOOTH, K. S. 1990. Heuristics for Ray Tracing Using Space Subdivision. *Visual Computer 6*, 3.

NG, K., AND TRIFONOV, B. 2003. Automatic Bounding Volume Hierarchy Generation Using Stochastic Search Methods. In *CPSC532D Mini-Workshop "Stochastic Search Algorithms"*.

OVERBECK, R., RAMAMOORTHI, R., AND MARK, W. R. 2008. Large Ray Packets for Real-time Whitted Ray Tracing. In *IEEE/Eurographics Symposium on Interactive Ray Tracing*.

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.

WALD, I., AND HAVRAN, V. 2006. On Building Fast KD-Trees for Ray Tracing, and on Doing That in O(NlogN). In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing* , 61–69.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26*, 1, 6.

WALD, I., BENTHIN, C., AND BOULOS, S. 2008. Getting Rid of Packets: Efficient SIMD Single-Ray Traversal using Multi-branching BVHs. In *IEEE/Eurographics Symposium on Interactive Ray Tracing 2008*.

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

WALTER, B., BALA, K., KULKARNI, M., AND PINGALI, K. 2008. Fast Agglomerative Clustering for Rendering. In *Proceedings of the IEEE Symposium on Interactive Ray Tracing*.

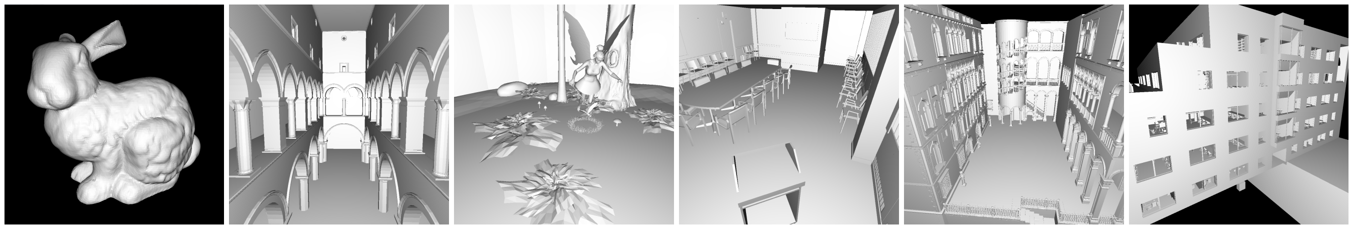

Figure 8: *Our test scenes and the view points for the benchmarks. Respective triangle counts are given in brackets. From left to right:* Bunny *(69K),* Sponza *(67K),* Fairy Forest *(174K),* Conference *(282K),* Venice *(1,236K),* Soda Hall *(2,169K).*

Scene	Build type	$Exp(T)$	Cost$_1^{rand}$	Trav.$_1^{rand}$	Int.$_1^{rand}$	FPS$_1^{rand}$	FPS$_4$	FPS$_{256}^{range}$	FPS$_{256}^{partition}$
Bunny	C-BVH	66.6	32.5	24.7	5.2	0.7	4.4	13.0	11.0
	GP-BVH ($K_R = 2^6$)	68.1	33.7	26.6	4.7	0.7	4.2	12.8	10.4
Sponza	C-BVH	142.6	68.3	46.7	14.4	0.4	1.4	8.1	4.8
	GP-BVH ($K_R = 2^6$)	167.1	85.5	54.6	20.6	0.3	0.9	7.7	3.6
	GP-BVH ($K_R = 2^{12}$)	150.0	75.8	51.0	16.5	0.4	0.8	7.2	3.4
Fairy	C-BVH	58.5	33.5	23.6	6.6	1.0	1.9	5.9	5.3
	GP-BVH ($K_R = 2^6$)	68.9	46.9	29.5	11.6	0.7	1.0	3.4	3.6
	GP-BVH ($K_R = 2^{12}$)	80.7	44.3	27.8	11.0	0.7	1.0	3.3	3.5
Conference	C-BVH	86.2	51.3	35.1	10.8	0.6	1.9	6.5	5.8
	GP-BVH ($K_R = 2^6$)	88.0	57.9	40.8	11.4	0.6	1.8	4.8	5.5
	GP-BVH ($K_R = 2^{12}$)	85.2	54.8	40.8	9.3	0.6	1.9	4.6	5.3
Venice	C-BVH	95.0	38.3	29.7	5.7	0.8	1.7	1.9	3.3
	GP-BVH ($K_R = 2^6$)	108.8	61.8	51.1	7.1	0.5	1.0	1.9	2.8
Soda Hall	C-BVH	166.0	43.1	36.5	4.4	0.8	2.0	6.6	5.3
	GP-BVH ($K_R = 2^6$)	167.5	56.8	46.7	6.7	0.5	1.5	4.4	4.1

Table 3: *Performance comparison of several BVHs produced by the geometric partitioner (GP-BVH) with different grid resolution coefficients K_R (see Section 3), compared to classic centroid sweep constructed trees (C-BVH). The resolution of the grid is chosen so that the number of cells is $\sqrt{K_R N}$. The traversal algorithms are described in Section 3.4. Although GP-BVH also considers C-BVH partitions at each step and chooses the one with lower SAH cost, the resulting trees have higher expected cost and lower ray tracing performance. Statistics have been gathered on a single core of an Intel Core2 Duo processor with image resolution of* 1024^2.

Scene	Type	Size	$Exp(T)$	Cost$_1^{rand}$	Trav.$_1^{rand}$	Int.$_1^{rand}$	FPS$_1^{rand}$	FPS$_4$	FPS$_{256}^{range}$	FPS$_{256}^{partition}$
Bunny	C-BVH	1.0 MB	66.6	32.6	24.7	5.2	0.7	4.4	13.0	11.0
	G-BVH	3.1 MB	67.2	31.6	26.0	3.7	0.8	4.1	7.3	9.3
	GK-BVH	3.1 MB	69.0	32.3	26.4	3.9	0.8	4.0	7.5	9.3
	KD-BVH	6.6 MB	95.3	43.7	39.2	3.0	0.7	3.0	6.0	7.0
Sponza	C-BVH	1.0 MB	142.6	68.3	46.7	14.4	0.4	1.4	8.1	4.8
	G-BVH	2.6 MB	121.2	43.2	32.8	6.9	0.8	2.0	7.6	5.9
	GK-BVH	2.8 MB	119.3	43.0	33.5	6.3	0.8	2.0	7.4	5.8
	KD-BVH	8.0 MB	120.7	43.1	38.0	3.4	0.8	1.8	7.3	5.2
Fairy	C-BVH	2.5 MB	58.5	34.0	23.6	6.9	1.0	1.9	5.9	5.1
	G-BVH	12.2 MB	69.9	36.4	25.1	7.5	1.0	1.9	3.8	5.1
	GK-BVH	11.6 MB	70.0	36.3	25.2	7.4	1.0	1.9	3.7	5.1
	KD-BVH	15.4 MB	73.2	37.6	30.8	4.5	0.9	1.7	3.5	4.3
Conference	C-BVH	4.1 MB	86.2	51.3	35.1	10.8	0.6	1.9	6.5	5.8
	G-BVH	4.4 MB	74.5	36.2	29.9	4.2	1.0	2.1	4.6	5.8
	GK-BVH	4.2 MB	74.0	37.2	30.9	4.2	1.0	2.1	4.2	5.7
	KD-BVH	30.0 MB	77.0	39.1	33.1	4.0	0.9	2.0	5.8	5.6
Venice	C-BVH	17.7 MB	95.0	38.3	29.7	5.7	0.8	1.7	2.0	3.3
	G-BVH	86.7 MB	84.0	29.2	25.0	2.8	1.1	2.1	1.8	3.7
	GK-BVH	66.8 MB	86.7	29.6	25.4	2.8	1.2	2.2	1.9	3.9
	KD-BVH	141.0 MB	102.7	35.5	32.3	2.1	1.0	1.9	1.9	3.4
Soda Hall	C-BVH	32.4 MB	166.0	43.1	36.5	4.4	0.8	2.0	6.6	5.3
	G-BVH	152.0 MB	121.3	23.6	19.5	2.7	1.2	3.2	10.8	8.7
	GK-BVH	80.0 MB	126.2	24.6	20.1	3.0	1.2	3.1	10.7	8.7
	KD-BVH	252.0 MB	136.0	29.3	27.6	1.1	1.0	2.8	11.0	7.7

Table 4: *Comparison of different configurations of our generic BVH construction algorithm. The construction algorithms – C-BVH, G-BVH, GK-BVH, and KD-BVH, are described in Section 4.5 and the traversal algorithms in Section 3.4. Performance can be dramatically improved for small packets and incoherent rays, when space subdivision is enforced. This can also increase tree size which can in turn reduce coherence for large ray packets. All traversal algorithms, however, benefit from space subdivision on scenes with high geometric and/or depth complexity. Statistics have been gathered on a single core of an Intel Core2 Duo processor with image resolution of* 1024^2.