Recursive SAH-based Bounding Volume Hierarchy Construction

Dominik Wodniok Michael Goesele

Graduate School of Computational Engineering, TU Darmstadt

Abstract

Advances in research on quality metrics for bounding volume hierarchies (BVHs) have shown that greedy top-down SAH builders construct BVHs with superior traversal performance despite the fact that the resulting SAH values are higher than those created by more sophisticated builders. Motivated by this observation we examine a construction algorithm that uses recursive SAH values of temporarily constructed SAH-built BVHs to guide the construction. The resulting BVHs achieve up to 28% better trace performance for primary rays and up to 24% better trace performance for secondary diffuse rays compared to standard plane sweeping without applying spatial splits. Allowing spatial splits we still achieve up to 20% resp. 19% better performance. While our approach is not suitable for real-time BVH construction, we show that the proposed algorithm has subquadratic computational complexity in the number of primitives, which renders it usable in practical applications.

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

Ray tracing is an important computational primitive used in different algorithms including collision detection, line-of-sight computations, ray tracing-based sound propagation, and most prominently light transport algorithms. An efficient ray tracing implementation needs to rely on an acceleration structure. The most common ray tracing acceleration structures are kd-trees and bounding volume hierarchies (BVHs), with BVHs being the most widespread ones. The reasons for this are the much smaller and controllable memory footprint of BVHs, a more efficient empty space cut-off, faster construction, and a simple update procedure of the structure for animated data, while at the same time offering similar ray tracing performance as kd-trees.

BVHs give best ray tracing performance when constructed with the surface area heuristic (SAH) [4]. State-of-the-art SAH-based BVH builders are the greedy top-down plane-sweeping algorithm from MacDonald and Booth [8] and the extension of this algorithm with spatial splits proposed by Stich et al. [11]. More sophisticated algorithms have been developed (see the summary in Aila et al. [1]) that produce higher quality BVHs with respect to SAH. But the improvements do not translate well to actual measured performance and can in fact even decrease performance. Aila et al. [1] identified geometry that overlaps bounds of subtrees to which it does not belong as a second major factor and proposed the end-point-overlap metric (EPO) to measure this effect. They also revealed the unique characteristic of greedy top-down SAH builders, that they not only optimize SAH but also implicitly minimize EPO, which explains why they perform so well in practice.

To the best of our knowledge no approach has been proposed to date, which directly takes advantage of this implicit correlation of SAH and EPO for greedy top-down builders to construct better BVHs. We examine the possibility to improve EPO further by using recursive SAH evaluation on temporarily built BVHs as an accurate prediction for the SAH cost of subtrees during construction. Further, we reason why the temporary BVHs themselves have to be constructed with SAH to gain any benefit and propose an algorithm that can construct BVHs with recursive SAH in $O(N \log^2 N)$. Due to the computational complexity the algorithm is mainly suitable for static scenes and global illumination algorithms.

Our main contributions are as follows:

- a BVH construction algorithm that produces BVHs with better average performance than state-of-the-art methods and
- a complexity analysis of our algorithm that reveals subquadratic runtime in the number of primitives.

2 Background and Related Work

The strategy chosen for BVH construction has a tremendous influence on ray tracing performance. State-of-the-art construction approaches use the surface area heuristic (SAH) originally introduced by Goldsmith and Salmo [4]. It provides an approximation for the expected cost of traversing a given kd-tree or BVH. Underlying assumptions are that rays originate at infinity, have a uniform ray direction distribution, and do not terminate on intersection. The first and second assumption allow to compute the geometrical conditional probability p_n of intersecting the convex bounding volume of a node n with a random ray given that the ray hits the convex bounds of the surrounding parent node $P(n)$ of n as the surface area ratios of their bounds. Combined with implementation dependent constants c_t for traversal step costs and c_i for primitive intersection test cost the expected traversal cost for a tree node n can be recursively computed as:

$$c(n) = \begin{cases} c_t + p_l c(l) + p_r c(r) & \text{inner node} \\ |n| c_i & \text{leaf node} \end{cases} \tag{1}$$

Here, l and r are the left and right child of n in case of an inner node, and $|n|$ is the number of primitives belonging to n. Evaluating c for the tree root yields the expected cost of the whole tree.

State-of-the-art greedy top-down plane-sweeping construction locally applies an approximation of Equation 1 when splitting a node. Several candidate partitions are generated and rated with c under the assumption that the newly generated children stay leaves. That is we compute:

$$c_{split} = c_t + p_l |l| c_i + p_r |r| c_i \tag{2}$$

The partition with smallest c_{split} is chosen and construction recursively proceeds with the children. The recursion terminates as soon as the smallest c_{split} is higher than the cost for creating a leaf node. Partitions are typically generated by sweeping axis aligned planes through every dimension and checking on which side the bounding volume centroids of primitives fall. With this approach, only planes which contain bounding volume centroids are relevant.

Though the assumptions underlying SAH generally do not apply in practice, SAH guided construction empirically produces the best performing BVHs to date. Unfortunately, SAH-based construction is also the most expensive. Wald et al. [13] introduced an $O(n\log(n))$ algorithm for SAH based BVH construction. Fabianowski et al. [2] changed the SAH assumption of infinitely far away ray origins to origins uniformly distributed in the scene bounds. On average ray tracing performance increases of 3.5% have been reported.

Lauterbach et al. [7] proposed three GPU BVH construction algorithms with different trade-offs between tree quality and construction time: The median split-based LBVH algorithm is fast but has poor tree quality. The second algorithm is a parallel approach for full binned-SAH BVH construction (see Wald [12]) with high tree quality but slower construction. The third algorithm, a hybrid of the former two, strikes a balance: Upper levels are constructed according to the highly parallel first algorithm while the remaining levels expose enough parallelism to be efficiently constructed according to the second one. Pantaleoni and Luebke [9], and Garanzha et al. [3] proposed much faster implementations for the median split and the hybrid algorithm called HLBVH which allow real-time rebuilds for scenes with up to 2 million triangles. An important change to the hybrid algorithm is, that LBVH is used to build the lower levels of the tree first. The roots of the subtrees themselves are then used for binned top-down SAH BVH construction. Thus the expensive part of the algorithm is performed on much less input elements and tree quality is improved in the important upper levels.

Stich et al.'s [11] offline SBVH algorithm drastically improves tree quality for scenes with widely varying degree of tessellation. The key idea is to either use spatial splits or object partitioning depending on which of them yields a better SAH value. When searching for a node split, the best spatial split is determined in addition to the best object split. Spatial splits are only applied when considered beneficial. To date no efficient GPU implementation of this algorithm has been presented. Karras and Aila [6] proposed an approximate but real-time construction algorithm for GPUs that takes any BVH (i.e. LBVH) as input and performs local optimizations on small node subsets (treelets) w.r.t. SAH. They also present a triangle pre-splitting heuristic with strong focus on producing splits, which are likely to be beneficial for tree quality. Resulting trees achieve about 90% of SBVH tree quality.

Plane sweeping only generates left-right type partitions. Popov et al. [10] proposed to allow more general partitionings in order to achieve smaller SAH cost. This is done by pre-generating a set of more general child bound pairs and distributing the primitives to these sets. Though achieving smaller total SAH values, trace performance did not improve equally or even decreased. Further, they also tried to improve their general partition BVH constructor by rating partitions with recursive SAH computed from temporarily built object-median split BVHs. This improved measurements but results were still inferior to the standard plane sweeping algorithm.

Aila et al. [1] analyzed the correlation between measured performance and the recursive SAH value of BVHs constructed with several BVH construction algorithms. Their motivation was the often made observation that more sophisticated construction algorithms that managed to construct lower SAH BVHs improved measurements less than expected or even decreased performance (e.g. Popov et al. [10]). At the same time BVHs with similar SAH value but constructed with different algorithms can give significantly different trace performance. Aila et al. identified end-point-overlap (EPO) as the missing piece of information and proposed the EPO metric to better predict performance of BVHs in combination with SAH. EPO is essentially a measure for the extra traversal cost caused by intersection with primitives which intersect bounds of subtrees they do not belong to. We refer to Aila et al. [1] for computation of EPO of a BVH. The performance predictor is a convex combination of SAH and EPO:

$$p \sim SAH \cdot (1-\alpha) + EPO \cdot \alpha \qquad (3)$$

Here, α is a scene dependent constant. This predictor is only designed for secondary diffuse rays and scalar traversal. α can be as high as 0.98. With such high possible values it becomes clear that optimizing SAH alone is not enough and that even the BVH with total minimum SAH is probably not the best performing one. A further result especially important for the next section, where we describe our algorithm, is that top-down greedy SAH based construction algorithms implicitly reduce node overlap in a way that also minimizes EPO, which gives them an innate superiority over bottom-up or hybrid algorithms.

Finally, for SAH kd-tree construction Havran [5] examined a possibly better prediction model than Equation 2 for subtree SAH costs. He assumed that geometry is distributed uniformly in space, that a spatial-median split strategy is used, and that the subtree root has cubic bounds. He proved that the predicted cost is in $O(N)$, which renders the classic linear cost model (Equation 2) sufficient under these assumptions. Havran further elaborates on minimum total SAH cost kd-tree construction. This requires to recursively evaluate SAH for each split candidate and the recursive evaluation itself has to recursively apply recursive SAH evaluation. This results in a combinatorial explosion which Havran states to be NP-hard and also translates to BVH construction. As a side note, a consequence of the work of Aila et al. [1] would be that the minimum total SAH cost kd-tree would probably also be the best performing one as EPO is always zero for kd-trees.

3 Algorithm

The goal of the approach presented in this section is to give a better predictor for split candidates than the classic linear model given in Equation 2. To achieve this, the model has to improve both SAH as well as EPO. So far greedy top-down builders are the only known builders which, at least implicitly, minimize EPO. We want to take advantage of this characteristic during construction. As EPO minimization only occurs implicitly during construction we cannot estimate it, e.g., just from the number of primitives and bounds of the node to split. Thus we propose to actually construct temporary, greedily built BVHs for the left and right side of each split candidate and use their recursive SAH values. The rating function for a split becomes then

$$c_{split} = c_t + p_l c(t_l) + p_r c(t_r), \qquad (4)$$

where t_l and t_r are the roots of the temporarily constructed BVHs and c is the recursive SAH function introduced in Equation 1. Note that EPO does not directly appear in this rating function. We rely on the correlation of SAH and EPO of the greedily top-down built temporary BVHs to find the split with minimal EPO by finding the split with minimal c_{split}. This also should implicitly guide global construction into directions of lower EPO. Pseudocode for determining the best split is given in Algorithm 1.

Seen from a different angle, our approach can also be interpreted as a middle ground between the NP-hard algorithm proposed by Havran [5] and the recursive SAH evaluation on temporary spatial-median split trees used by Popov et al. [10] in terms of computational complexity and BVH quality. The difference is, that we give a more representative rating for the split candidates than an object-median split as it much closer reflects the way the main BVH itself is constructed. Object-median split construction does not incorporate SAH in any way. Thus, SAH values retrieved from temporary BVHs constructed this way are an unreliable guide for construction that aims at reducing SAH.

We will now focus on the algorithmic aspects of our approach, which we call recursive SAH-based bounding volume hierarchy

```
input : node                                        // node to split
input : $c_t$                                       // cost of a traversal step
input : $c_i$                                       // cost of intersecting a primitive
output: bestP                                       // Best primitive partition
output: bestC                                       // Best partition costs
1  (bestP, bestC) ← $(\emptyset, \infty)$
2  partitions ← generate_partitions(node)
3  foreach partition ∈ partitions do
4      $t_l$ ← build_temporary_bvh(partition.left, $c_t$, $c_i$)
5      $t_r$ ← build_temporary_bvh(partition.right, $c_t$, $c_i$)
6      $c_l$ ← compute_sah($t_l$, $c_t$, $c_i$)
7      $c_r$ ← compute_sah($t_r$, $c_t$, $c_i$)
8      $(p_l, p_r)$ ← compute_intersection_probabilities(partition)
9      $c_{split} \leftarrow c_t + p_l c_l + p_r c_r$
10     if $c_{split}$ < bestC then
11         (bestP, bestC) ← $(partition, c_{split})$
12     end
13 end
```

Algorithm 1: Pseudocode for determining the best node split with recursive SAH.

construction (RBVH). The `generate_partitions` function in Algorithm 1 determines if the main BVH is constructed with plane-sweeping or binning, though more general partitions such as in Popov et al. [10] are possible, too. The `build_temporary_bvh` function for temporary BVH construction can also use arbitrary construction schemes but we only consider top-down plane-sweeping or binning construction. As a result we have four different RBVH algorithms with their own asymptotic computational complexities. We will proceed with deriving complexities for all four cases.

3.1 Computational Complexities

We first recap computational complexity of the standard SAH-based construction. The common plane-sweeping algorithm implementation which sorts in every dimension needs $O(n\log n)$ steps to find a split and $O(n\log^2 n)$ steps in total. Adapting the concepts of Wald and Havran [14] to BVHs allows to find a split in $O(n)$ steps and needs $O(n\log n)$ steps in total. Using binning construction instead results in the same complexities as Wald and Havran, but with a smaller constant. To simplify derivation of the complexities we make the common assumption that a split produces two new nodes with roughly the same number of primitives and that the number of scene primitives N is a power of two.

3.1.1 Sweep-Sweep / Sweep-Binning Construction

We start with the derivation of the complexity of sweep-sweep construction. For a node with N primitives a sweep based construction generates $N-1$ candidate partitions. This results in $2(N-1)$ temporary BVHs that need to be constructed. With a Wald and Havran [14] like approach each temporary BVH can be constructed in $O(n\log n)$, where n is the number of primitives of each side of a candidate partition. Using the hyper factorial $H(x)=\prod_{i=1}^{x} i^i$ and $O(n\log n)=O(\log n^n)$ we can define the recurrence relation $T(N)$ of the algorithm:

$$\begin{aligned}
T(N) &= 2\left(\sum_{i=1}^{N-1} i\log i\right) + 2T\left(\frac{N}{2}\right)\\
&= 2\log\left(H(N)\right) + 2T\left(\frac{N}{2}\right)\\
&= 2\log\left(H(N)\right) + 2\left(\log\left(H\left(\frac{N}{2}\right)\right) + 2T\left(\frac{N}{4}\right)\right)\\
&= 2\sum_{i=0}^{\log N} 2^i \log\left(H\left(\frac{N}{2^i}\right)\right)
\end{aligned} \tag{5}$$

Using the simple-to-derive upper bound $\log(H(x)) < x^2\log x$ we get:

$$\begin{aligned}
T(N) &= 2\sum_{i=0}^{\log N} 2^i \log\left(H\left(\frac{N}{2^i}\right)\right)\\
&< 2\sum_{i=0}^{\log N} 2^i \left(\frac{N}{2^i}\right)^2 \log\left(\frac{N}{2^i}\right)\\
&= 2\sum_{i=0}^{\log N} N^2 2^{-i}\left(\log(N)-i\right)\\
&= 2N^2\left(\log(N)\left(\sum_{i=0}^{\log N} 2^{-i}\right) - \sum_{i=0}^{\log N} 2^{-i} i\right)\\
&\to O\left(N^2\log(N)\right)
\end{aligned} \tag{6}$$

As we only found an upper bound for $\log(H(x))$ the asymptotic complexity $O(N^2\log N)$ is not tight. Using $\log(H(x)) > x^2/2$ we get the lower bound $\Omega(N^2)$ for the asymptotic complexity.

Asymptotic complexity of binning-based temporary BVH construction is the same as for full-sweep-based construction akin to Wald and Havran [14]. Consequently the sweep-binning approach has the same complexity as the sweep-sweep approach.

3.1.2 Binning-Binning / Binning-Sweep Construction

Let $B=2^b, b\in\mathbb{N}$ denote the number of bins for the main BVH. The number of bins for the temporary BVHs is not needed, as it does not appear in the complexity of binned construction. For simplicity we assume that geometry is roughly distributed uniformly in space such that a node with N primitives generates B bins with N/B primitives in each bin after binning. This results in $B-1$ candidate partitions and thus $2(B-1)$ temporary BVHs that need to be constructed. Each temporary BVH itself is constructed in $O(n\log n)$, where n is the number of primitives in the union of all bins on each side of a candidate partition. This results in the following recurrence relation:

$$\begin{aligned}
T(N) &= 2\left(\sum_{i=1}^{B} i\frac{N}{B}\log\left(i\frac{N}{B}\right)\right) + 2T\left(\frac{N}{2}\right)\\
&= 2\sum_{i=0}^{\log N} 2^i\left(\sum_{j=1}^{B} j\frac{N}{2^iB}\log\left(j\frac{N}{2^iB}\right)\right)\\
&\in O(N\log^2 N + BN\log(B)\log(N)) = O(N\log^2 N)
\end{aligned} \tag{7}$$

We used the upper bound of $\log H(x)$ for the derivation. But it has no effect on the asymptotic complexity. Appendix A describes the full derivation. As we can see the binning/binning construction algorithm has subquadratic complexity and thus more relevance in practice. Though the number of bins asymptotically has no effect on runtime it linearly increased runtime in our experiments for problem sizes we used in our tests. The reason for this is that the $BN\log(B)\log(N)$ of T is dominating up to a certain problem size. We proceed with computing bounds for the number of input primitives N for which the number of bins causes the second most dominating term to dominate the $N\log^2 N$ term. Using the lower bound for $\log H(B)$ the second term becomes $BN\log(N)$. Equating the two dominating terms of T for the upper and lower bound of $\log H(B)$ we get:

$$N\log^2 N = BN\log(N) \tag{8}$$

$$N\log^2 N = BN\log(B)\log(N) \tag{9}$$

Solving for N we get the bounds $2^B < N < B^B$. As a result even for the small number of $B=32$ bins the $BN\log(B)\log(N)$ term dominates till $2^{32} < N < 2^{160}$ primitives. Thus, B keeps impacting construction time even for scenes which have a several orders of magnitude higher number of primitives than current scenes in production rendering.

Again, due to the same asymptotic complexity of binning and sweep construction of temporary BVHs binning-sweep construction has the same complexity as binning-binning construction.

3.2 Spatial Splits

To also take advantage of spatial splits akin to SBVH from Stich et al. [11] we cannot simply treat them as an additional technique to RBVH. SBVH uses the linear cost model from Equation 2 which is an upper bound on the cost model of RBVH (Equation 4). This does not allow us to compare split candidates from those techniques in a meaningful way. We simply have to adapt the `generate_partitions` function to also generate spatial partitions in order to remove this problem. This requires to temporarily split primitives for each candidate partition, but potentially allows to find even better split candidates. We included this variant into the evaluation, where we call it recursive spatial split bounding volume hierarchy (RSBVH).

4 Evaluation

To evaluate our proposed construction algorithm we measured the impact on SAH, EPO and traversal performance. To do so we used a number of freely available test scenes (see Table 1). We only evaluated the $O(BN\log^2 N)$ binning-binning algorithm as the superquadratic complexity of the sweep-sweep and and sweep-binning algorithm proved to be impractical in practice. We also included the RSBVH algorithm from Section 3.2 into the evaluation. We chose the standard plane-sweeping approach as the baseline construction algorithm to compare against for construction without spatial splits. We did not include the general partitions with recursive SAH evaluation on temporarily built object-median split BVHs from Popov et al. [10] as the authors stated, that measured performance was inferior to the standard plane-sweeping approach.

For construction with spatial splits we chose the SBVH algorithm from Stich et al. [11]. SBVH allows to specify a parameter which guides spatial split attempts. We follow the authors recommendation and use a value of 10^{-5} for all scenes. Exceptions were *Hairball* were we used 10^{-4} to avoid excessive primitive duplication and *San Miguel* where we had to use 10^{-6} for any spatial splits to occur.

For the main BVH we have configurations with 256 bins and 64 bins for the number of object split bins. For RSBVH the number of spatial bins is 128 and 64 for the configurations with 256 and 64 object split bins respectively. In all cases the number of bins for temporary BVH construction is 32. SAH build constants were set to $(c_t, c_i) = (1.2, 1.0)$. With two baseline BVHs and the four recursive-SAH based BVHs we have a total of six BVHs per scene. All BVH algorithms and configurations along with abbreviations we used for them are listed in Table 2.

We measure performance of front-to-back traversal with primary rays and secondary diffuse rays to compare quality of the different BVHs. To get implementation independent measurements we measured the average number of traversal steps $\overline{n}_s$ and the average number of intersected triangles $\overline{n}_t$ over a varying number of views for each scene. Combined with the SAH constants we define the average measured traversal cost

$$\overline{m} = \overline{n}_s c_t + \overline{n}_t c_i. \tag{10}$$

We also give results for predicted traversal cost with EPO according to Equation 3. As our performance measure is slightly different from the one used by Aila et al. we recomputed the scene dependent α values together with the associated Pearson correlation coefficients (see Table 1). We simply compute α by sampling the $[0,1]$ range and selecting the α which gives the highest correlation. Correlation of p with $\overline{m}$ for diffuse rays is well above 0.99 for most scenes. Though only intended for secondary diffuse rays we also computed a separate α for primary rays. Surprisingly, correlation is also well above or close to 0.99 in this case except for the *Soda* and *Conference* examples.

Table 2: List of algorithms and their configurations we used for evaluation. o and s denote the numbers of object and space partitioning bins used for construction of the main BVH. t is the number of object partitioning bins used for the construction of temporary BVHs.

Algorithm	Abbr.	o	s	t
Baseline Plane-Sweep	BBVH	-	-	-
Baseline SBVH	SBVH	-	128	-
Recursive SAH	RBVH	256	-	32
Recursive SAH	RBVH*	64	-	32
Recursive SAH + SBVH	RSBVH	256	128	32
Recursive SAH + SBVH	RSBVH*	64	64	32

Our implementation of the algorithms only parallelized the foreach loop in Algorithm 1. This is not optimal as it introduces global synchronization between every node split. It is possible to parallelize the whole construction process, though. Our test platform is equipped with two Intel Xeon E5-2650-3930K octacore CPUs. Construction timings are included in our results.

All performance measurements are collected in Table 4. To give a more condensed view of the results Table 3 shows relative improvements averaged over all scenes with BBVH and SBVH as baseline. For each scene measurements are sorted from best to worst with respect to $\overline{m}$ of diffuse rays.

RSBVH managed to improve SAH, EPO and trace performance in all scenes compared to BBVH and SBVH. Performance of primary and diffuse rays improves roughly by the same amount on average. RSBVH achieves 25% and RBVH 11% improvement. But with an average improvement of 19% SBVH performs better than RBVH. SBVH improves SAH only slightly compared to RBVH, but outperforms RBVH in EPO improvements. On average RSBVH manages to improve on SBVH by 8% and 16% for SAH and EPO. But improvements of up to 15% and 70% have been achieved. In terms of trace performance RSBVH achieved improvements of up to 19% over SBVH with an average of 9% for primary rays and 5% for diffuse rays.

Recursive-SAH based construction time is one to two orders of magnitude higher than for the baseline. For the number of primitives of our scenes runtime almost linearly increases with the number of bins. RSBVH* needs a similar or smaller amount of time for construction than RBVH. At the same time its quality is almost identical to RSBVH.

5 Discussion

The proposed RBVH and RSBVH builders managed to reduce SAH as well as EPO by a significant amount. Both algorithms do not handle EPO reduction directly but rely on the implicit correlation of SAH and EPO minimization of greedy top-down builders. Thus, our results also reconfirm the results from Aila et al. [1]. Though SBVH already gives high average reduction in SAH and more so in EPO, RSBVH managed to push both reductions even further. Although RBVH manages to reduce SAH and EPO well, the spatial splits of SBVH prove to be a superior splitting strategy. Considering that RBVH only performs object splits, an EPO reduction of up to 65% and 23% on average is quite impressive. If primitive duplication is not desired RBVH might be an alternative to SBVH.

Table 1: Listing of all scenes used for benchmarking our algorithms along with their number of primitives. α_p and α_d are the EPO weights of primary and diffuse rays along with their Pearson correlation coefficient for the measurements.

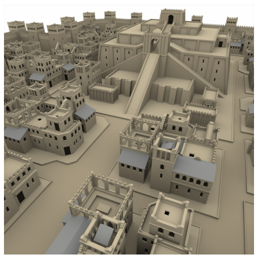
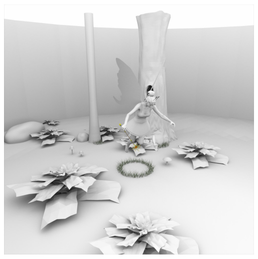
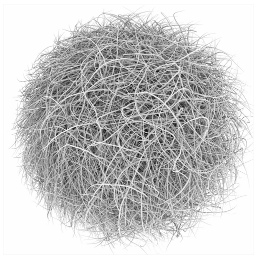
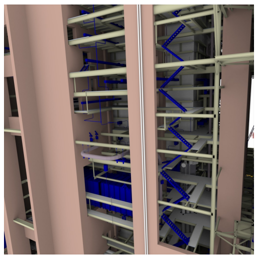
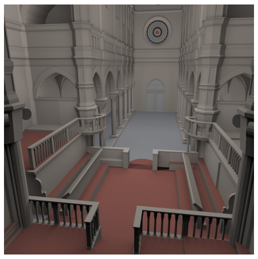
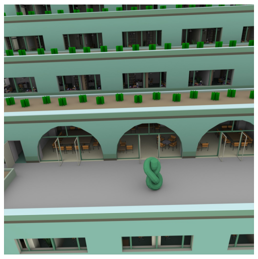
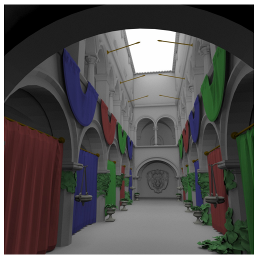
				
Babylon: 488199	Bubs: 1850084	Conference: 282675	Fairy: 172677	Hairball: 2850000
α_p : 1.0 corr.: 0.994	α_p : 0.0 corr.: 0.997	α_p : 0.0 corr.: 0.884	α_p : 0.56 corr.: 0.965	α_p : 0.61 corr.: 0.995
α_d : 0.53 corr.: 0.999	α_d : 0.25 corr.: 0.997	α_d : 0.4 corr.: 0.999	α_d : 0.85 corr.: 0.946	α_d : 0.73 corr.: 0.998
				
Powerplant: 294703	San Miguel: 10483092	Sibenik: 79937	Soda: 2169046	Sponza: 262141
α_p : 0.42 corr.: 0.999	α_p : 0.82 corr.: 0.988	α_p : 0.66 corr.: 0.993	α_p : 0.0 corr.: 0.576	α_p : 0.39 corr.: 0.991
α_d : 0.21 corr.: 0.999	α_d : 0.93 corr.: 0.998	α_d : 0.77 corr.: 0.998	α_d : 0.7 corr.: 0.969	α_d : 0.78 corr.: 0.999

Table 3: Average, minimum, and maximum relative reduction of SAH, EPO, as well as predicted (p) and measured ($\overline{m}$) traversal cost of primary and diffuse rays over all scenes. Algorithms are either compared against BBVH or SBVH as baseline. For each baseline, algorithms are sorted from highest to lowest reduction in traversal cost of diffuse rays. The highest reduction of each attribute is highlighted per baseline.

		Avg. (Min/Max) reduction (%)					
				Primary rays		Diffuse rays	
Baseline	Algorithm	SAH	EPO	p	$\overline{m}$	p	$\overline{m}$
BBVH	RSBVH	-21.5 (-6.3 / -34.8)	-72.2 (-19.7 / -86.2)	-35.1 (-7.9 / -86.2)	**-25.7** (-11.8 / -36.3)	-36.4 (-11.1 / -75.5)	**-24.8** (-8.3 / -32.6)
	RSBVH*	**-21.7** (-6.3 / -36.6)	**-72.4** (-21.2 / -86.3)	**-35.4** (-8.0 / -86.3)	-23.7 (-2.1 / -33.6)	**-36.6** (-11.6 / -74.1)	-24.5 (-6.9 / -32.4)
	SBVH	-14.4 (+3.8 / -26.9)	-64.3 (-19.5 / -83.0)	-27.6 (1.1 / -83.0)	-17.9 (3.0 / -31.8)	-29.6 (-4.5 / -61.2)	-20.1 (-2.1 / -32.3)
	RBVH	-12.7 (-2.7 / -33.3)	-23.2 (-1.4 / -65.3)	-14.5 (-2.7 / -33.3)	-12.0 (-2.3 / -28.3)	-15.9 (-2.7 / -36.3)	-10.7 (-1.9 / -24.5)
	RBVH*	-12.0 (-2.4 / -33.3)	-23.0 (-2.6 / -66.0)	-13.4 (-2.4 / -33.3)	-11.5 (+0.8 / -28.1)	-15.4 (-2.4 / -36.3)	-10.3 (-2.5 / -24.4)
SBVH	RSBVH	-8.2 (-1.9 / -15.6)	-15.7 (+53.3 / -70.6)	-11.6 (-3.0 / -29.0)	**-9.0** (-0.7 / -19.0)	**-10.6** (+0.6 / -37.8)	**-5.7** (-0.5 / -15.9)
	RSBVH*	**-8.4** (-1.3 / -14.7)	**-16.0** (+40.6 / -71.1)	**-11.9** (-3.1 / -25.9)	-6.7 (+7.8 / -19.8)	**-10.6** (-2.0 / -33.2)	-5.2 (+4.0 / -19.1)
	RBVH	+2.6 (+24.0 / -12.0)	+167.8 (+401.8 / +9.5)	+50.7 (+401.8 / -8.7)	+8.9 (+38.7 / -12.3)	+24.4 (+93.6 / -6.4)	+12.6 (+32.1 / -4.0)
	RBVH*	+3.5 (+25.0 / -9.2)	+169.9 (+445.3 / +8.6)	+55.6 (+445.3 / -8.7)	+9.4 (+42.3 / -12.2)	+25.1 (+94.0 / -6.5)	+13.2 (+34.6 / -3.9)

The downside of our proposed algorithms is the large increase in construction time, which makes them highly unsuitable for real-time applications. Our largest test scene, *San Miguel*, took almost 4 hours to construct with RSBVH. However, global illumination computations are one application that requires many intersection tests and can thus offset the initial costs of acceleration structure construction. We also have to remark that our implementation was not heavily optimized and not entirely parallelized. With enough implementational effort it should be possible to significantly increase construction performance. Construction of temporary BVHs could be completely offloaded to a GPU. Only primitive bounds are needed for construction. When primitive bounds are reordered according to the bins they fall into, the GPU can incrementally construct all temporary BVHs without further reloading or reordering. This also should give a significant performance boost.

An important observation we made in Table 3 is that the average predicted performance increase for diffuse rays is around 50% higher than the average measured performance. This is surprising as eight out of ten test scenes have a correlation coefficient of 0.997 or higher, and the remaining two scenes have a coefficient of 0.946 and 0.969. Under these circumstances we would expect the relative increases of predicted and measured performance to be almost identical. After first suspecting an implementational fault we found the explanation in the data. Having for example a look at the data for diffuse ray performance of *San Miguel* in Table 4 we can see that there is a huge discrepancy in predicted and measured relative difference in performance. At the same time the correlation coefficient is 0.998 which is really good. The corresponding α_d of 0.93 suggests that EPO dominates performance in this scene. But an α_d of 0.1 already explains the observed performance quite well. The corresponding correlation coefficient of 0.715 makes it seem that this is not a good choice. Though less pronounced this behavior can be observed for all other scenes with high correlation coefficient. Thus, the Pearson correlation coefficient is not a good metric for determining a good α and an alternative needs to be developed.

6 Future Work

There are several directions for future work. One direction would be to include the general partitions from Popov et al. [10] into `generate_partitions`. But this time the recursive SAH of greedy top-down SAH built BVHs is used. Combining our observations and the observations from Popov et al. [10], we expect further reductions in SAH from this approach. But as the general partitions allow to produce more node overlap, which in turn increases EPO, than the sweep based algorithms we fear that the algorithm might still produce inferior results.

Another direction that definitely should improve results even further is to include spatial splits into `build_temporary_bvh`. This only works in combination with RSBVH as the main construction algorithm has to be able to perform at least the same splits as `build_temporary_bvh`.

A further exciting direction is to directly include EPO into the construction process. We can readily compute EPO of a candidate partition from the temporarily built BVHs combined with the node to split. This would allow us to directly use Equation 3 to guide construction into directions of low p. An unpleasant aspect of this approach is that construction depends on prior knowledge of α. In this regard fast and accurate determination of α for unknown scenes is also an interesting problem.

Acknowledgements

The work of D. Wodniok is supported by the 'Excellence Initiative' of the German Federal and State Governments and the Graduate School of Computational Engineering at Technische Universität Darmstadt. We would like to thank J. Good for *Babylon*, R. Vance for *Bubs*, A. Grynberg and G. Ward for *Conference*, University of Utah for *Fairy*, S. Laine for *Hairball*, UNC for *Powerplant*, G. M. L. Llaguno for *San Miguel*, M. Dabrovic for *Sibenik*, UC Berkeley for *Soda*, and F. Meinl for *Crytek-Sponza*.

References

[1] T. Aila, T. Karras, and S. Laine. On quality metrics of bounding volume hierarchies. In *Proc. HPG*, 2013.

[2] B. Fabianowski, C. Fowler, and J. Dingliana. A Cost Metric for Scene-Interior Ray Origins. In *Proc. EG*, 2009.

[3] K. Garanzha, J. Pantaleoni, and D. McAllister. Simpler and faster HLBVH with work queues. In *Proc. HPG*, 2011.

[4] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 1987.

[5] V. Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, 2000.

[6] T. Karras and T. Aila. Fast parallel construction of high-quality bounding volume hierarchies. In *Proc. HPG*, 2013.

[7] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast BVH construction on GPUs. *CGF*, 2009.

[8] D. J. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Vis. Comput.*, 1990.

[9] J. Pantaleoni and D. Luebke. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In *Proc. HPG*, 2010.

[10] S. Popov, I. Georgiev, R. Dimov, and P. Slusallek. Object partitioning considered harmful: space subdivision for bvhs. In *Proc. HPG*, 2009.

[11] M. Stich, H. Friedrich, and A. Dietrich. Spatial splits in bounding volume hierarchies. In *Proc. HPG*, 2009.

[12] I. Wald. On fast construction of SAH-based bounding volume hierarchies. In *Proc. IEEE IRT*, 2007.

[13] I. Wald, S. Boulos, and P. Shirley. Ray tracing deformable scenes using dynamic bounding volume hierarchies. *ACM Trans. Graph.*, 2007.

[14] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In *Proc. IEEE IRT*, 2006.

A Binning-Binning Construction Complexity

In this appendix, we derive the result from Equation 7 in detail. We used $\log H(B) < B^2 \log B$.

$$\begin{aligned}
T(N) =& 2\left(\sum_{i=1}^{B} i\frac{N}{B}\log\left(i\frac{N}{B}\right)\right) + 2T\left(\frac{N}{2}\right) \\
=& 2\left(\sum_{i=1}^{B} i\frac{N}{B}\log\left(i\frac{N}{B}\right)\right) + \\
& 4\left(\sum_{i=1}^{B} i\frac{N}{2B}\log\left(i\frac{N}{2B}\right)\right) + 4T\left(\frac{N}{4}\right) \\
=& 2\sum_{i=0}^{\log N} 2^i\left(\sum_{j=1}^{B} j\frac{N}{2^iB}\log\left(j\frac{N}{2^iB}\right)\right) \\
=& 2\sum_{i=0}^{\log N} \frac{N}{B}\left(\sum_{j=1}^{B} j\log\left(j\frac{N}{2^iB}\right)\right) \\
=& 2\sum_{i=0}^{\log N} \frac{N}{B}\left(\log H(B) + B\log N - iB - B\log B\right) \\
<& 2\sum_{i=0}^{\log N} \frac{N}{B}\left(B^2\log B + B\log N - iB - B\log B\right) \\
=& 2\sum_{i=0}^{\log N} \left(BN\log B + N\log N - iN - N\log B\right) \\
=& 2\Big(N\log^2 N + BN\log(B)\log(N) - \\
& O(N\log N) - O(\log^2 N)\Big) \\
\in& O(N\log^2 N).
\end{aligned} \tag{11}$$

Table 4: Results for all scenes. $\overline{n}_s$, and $\overline{n}_t$ is the average number of measured traversal steps and triangle intersection tests. p is the EPO measure for BVH performance (Equation 3) and $\overline{m}$ the average measured traversal cost (Equation 10). For each scene builders are sorted from smallest to largest $\overline{m}$. The highest reduction of each attribute is highlighted per scene.

						Primary rays				Diffuse rays			
Scene	Builder	Time(s)	Dupl.	SAH	EPO	$\overline{n}_s$	$\overline{n}_t$	p	$\overline{m}$	$\overline{n}_s$	$\overline{n}_t$	p	$\overline{m}$
Babylon	RSBVH	457.87	69.94%	**39.08**	2.09	28.93	2.61	2.09	**37.33**	**30.49**	**4.26**	**19.29**	**40.85**
	RSBVH*	212.90	70.22%	39.79	**2.07**	30.29	**2.60**	**2.07**	38.94	31.30	**4.26**	19.61	41.81
	SBVH	18.65	40.28%	40.41	2.56	**28.65**	3.21	2.56	37.59	31.21	4.94	20.16	42.39
	RBVH	261.86	-	49.22	12.86	38.77	5.34	12.86	51.86	41.08	6.71	29.77	56.01
	RBVH*	88.89	-	49.67	13.98	40.11	5.34	13.98	53.48	41.92	6.74	30.58	57.04
	BBVH	3.81	-	53.72	15.12	39.83	5.28	15.12	53.08	44.47	7.10	33.07	60.47
Bubs	RSBVH	1835.54	6.46%	15.81	1.85	**23.10**	2.91	15.81	**30.63**	**24.82**	3.95	12.37	**33.73**
	RSBVH*	812.17	6.38%	**15.37**	1.95	23.61	**2.62**	**15.37**	30.96	25.31	**3.87**	**12.07**	34.24
	RBVH	1509.98	-	16.16	2.91	24.26	2.85	16.16	31.96	26.09	4.17	12.90	35.48
	RBVH*	476.78	-	16.16	2.84	24.33	2.84	16.16	32.04	26.13	4.17	12.89	35.53
	SBVH	22.68	2.47%	17.70	**1.76**	26.58	2.98	17.70	34.87	27.42	4.05	13.78	36.95
	BBVH	16.77	-	24.23	8.38	34.26	3.44	24.23	44.55	35.40	4.50	20.33	46.98
Conference	RSBVH*	114.19	83.34%	**33.35**	**2.84**	20.29	2.96	**33.35**	**27.31**	**24.01**	4.98	**21.18**	**33.80**
	RSBVH	221.26	83.09%	33.37	2.93	20.54	**2.93**	33.37	27.58	24.18	**4.97**	21.23	33.99
	SBVH	7.71	30.32%	38.32	3.53	23.16	6.26	38.32	34.05	26.39	7.26	24.44	38.93
	RBVH	133.01	-	38.56	7.09	**19.76**	6.14	38.56	29.85	26.13	9.94	26.01	41.29
	RBVH*	51.45	-	39.85	7.08	19.88	6.05	39.85	29.91	26.64	9.94	26.78	41.91
	BBVH	2.16	-	46.44	9.79	25.44	6.44	46.44	36.97	32.16	10.40	31.82	48.99
Fairy	RSBVH	170.77	32.62%	**31.27**	2.70	**26.49**	**4.47**	15.16	**36.26**	**29.96**	8.22	7.07	**44.17**
	RSBVH*	72.47	33.10%	31.29	**2.65**	27.19	4.48	**15.14**	37.11	30.55	**8.21**	**7.03**	44.87
	RBVH	115.61	-	31.48	2.97	27.73	4.93	15.40	38.21	31.26	8.65	7.33	46.16
	RBVH*	38.26	-	31.46	2.94	28.22	4.92	15.38	38.78	31.56	8.64	7.31	46.51
	SBVH	3.53	9.37%	34.65	2.71	31.39	4.66	16.63	42.33	32.39	8.32	7.60	47.18
	BBVH	1.22	-	33.38	3.37	30.12	4.97	16.45	41.12	32.88	8.73	7.96	48.19
Hairball	RSBVH*	1800.68	199.59%	**384.90**	**8.15**	77.75	26.87	**154.71**	120.17	**69.20**	**29.93**	**109.50**	**112.97**
	RSBVH	4492.63	191.41%	386.54	8.31	**74.61**	**25.83**	155.44	**115.37**	72.72	30.09	110.05	117.36
	SBVH	134.25	40.31%	427.92	28.25	80.56	44.48	183.72	141.15	76.74	47.48	135.76	139.56
	RBVH*	510.46	-	455.34	36.83	83.73	55.45	199.63	155.93	78.66	57.35	149.41	151.74
	RBVH	1700.80	-	453.97	36.72	81.95	54.95	199.03	153.29	79.30	57.40	148.96	152.57
	BBVH	24.74	-	466.36	37.82	85.58	56.08	204.52	158.78	81.33	58.00	153.09	155.59
Powerplant	RSBVH*	130.18	148.76%	32.68	**2.22**	**26.76**	**3.45**	19.76	**35.56**	**29.64**	**5.15**	26.16	**40.72**
	RSBVH	271.60	148.90%	**32.37**	2.28	26.84	3.49	**19.61**	35.70	29.79	5.23	**25.93**	40.98
	SBVH	14.03	84.81%	33.12	3.10	27.10	3.79	20.39	36.30	30.07	5.78	26.70	41.86
	RBVH	96.77	-	41.06	12.97	33.55	10.07	29.15	50.34	35.77	11.65	35.05	54.58
	RBVH*	37.92	-	41.40	12.81	34.03	10.08	29.28	50.91	36.70	11.54	35.28	55.58
	BBVH	2.09	-	43.93	13.16	35.98	10.05	30.88	53.22	38.89	11.78	37.34	58.45
San Miguel	RSBVH*	5588.86	21.81%	16.71	1.80	61.45	**6.24**	4.54	**79.98**	56.70	**8.87**	2.82	**76.91**
	RSBVH	14273.60	21.73%	**16.54**	**1.60**	**61.35**	6.80	**4.35**	80.43	57.25	9.28	**2.63**	77.99
	SBVH	216.34	13.71%	19.59	3.09	63.79	7.04	6.13	83.59	59.86	9.98	4.23	81.82
	RBVH	8926.36	-	17.25	7.52	72.50	9.81	9.31	96.81	67.56	12.97	8.19	94.04
	RBVH*	2952.33	-	17.82	7.49	72.18	9.71	9.39	96.32	68.32	12.87	8.20	94.85
	BBVH	135.45	-	20.28	10.21	84.25	10.00	12.07	111.09	75.18	13.16	10.91	103.38
Sibenik	RSBVH*	31.79	80.53%	**45.54**	**1.23**	**31.15**	3.32	**16.38**	**40.69**	**34.43**	**5.68**	**11.64**	**47.00**
	RSBVH	62.45	81.23%	46.51	1.23	31.82	**3.31**	16.71	41.49	34.82	5.70	11.87	47.49
	SBVH	3.78	34.75%	47.42	1.55	33.53	3.82	17.24	44.06	34.46	6.36	12.33	47.71
	RBVH*	11.25	-	48.79	4.11	36.58	4.47	19.39	48.37	37.02	7.10	14.61	51.52
	RBVH	29.88	-	48.75	4.16	36.29	4.46	19.41	48.01	37.27	7.07	14.64	51.79
	BBVH	0.48	-	53.64	5.00	40.43	4.39	21.63	52.90	39.68	7.01	16.43	54.63
Soda	RSBVH	2363.37	25.83%	**61.09**	**2.34**	**29.63**	3.52	**61.09**	**39.07**	**28.67**	4.92	**19.78**	**39.33**
	SBVH	47.35	14.03%	66.52	2.68	31.05	3.48	66.52	40.75	29.42	4.98	21.64	40.29
	RSBVH*	969.97	25.54%	61.21	2.47	33.90	**3.24**	61.21	43.92	30.98	**4.74**	19.92	41.91
	RBVH*	480.51	-	66.22	9.80	29.82	4.77	66.22	40.55	32.11	6.84	26.56	45.37
	RBVH	1432.23	-	66.15	10.17	30.12	4.84	66.15	40.99	32.07	6.91	26.80	45.39
	BBVH	21.31	-	77.93	13.70	33.03	5.23	77.93	44.87	36.92	7.51	32.78	51.81
Sponza	RSBVH	259.73	58.20%	64.94	4.59	**41.29**	4.05	41.59	**53.60**	42.61	**6.06**	17.87	**57.19**
	RSBVH*	115.17	58.01%	**64.16**	4.21	42.56	4.79	**40.96**	55.87	**42.45**	6.47	**17.40**	57.41
	SBVH	9.77	29.16%	70.12	**3.00**	46.43	**3.78**	44.14	59.50	42.68	6.23	17.76	57.45
	RBVH*	48.53	-	70.72	7.98	46.38	7.08	46.44	62.74	48.43	9.09	21.79	67.21
	RBVH	138.12	-	70.86	7.85	47.77	7.16	46.48	64.49	49.21	8.90	21.71	67.95
	BBVH	2.09	-	83.14	12.95	65.31	5.76	55.97	84.13	63.73	8.40	28.39	84.87