Construction of Bounding Volume Hierarchies with SAH Cost Approximation on Temporary Subtrees

Dominik Wodniok[a,b], Michael Goesele[a,b]

[a] *TU Darmstadt, Rundeturmstraße 12, 64283 Darmstadt, Germany*
[b] *Graduate School of Computational Engineering, Dolivostraße 15, 64293 Darmstadt, Germany*

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 three construction algorithms that use recursive SAH values of temporarily constructed SAH-built BVHs to guide the construction and perform an extensive evaluation. The resulting BVHs achieve up to 37% better trace performance for primary rays and up to 32% better trace performance for secondary diffuse rays compared to standard plane sweeping without applying spatial splits. Allowing spatial splits still achieves up to 18% resp. 15% 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. Additionally, we describe an approach for improving the forecasting abilities of the SAH-EPO ray tracing performance predictor which also increases its relevance for primary rays.

Keywords: ray tracing, bounding volume hierarchy

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 having gained popularity in the last decade. 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) [1]. State-of-the-art SAH-based BVH builders are the greedy top-down plane-sweeping algorithm from MacDonald and Booth [2] and the extension of this algorithm with spatial splits proposed by Stich et al. [3]. More sophisticated algorithms have been developed (see the summary in Aila et al. [4]) 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. [4] 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
- a complexity analysis of our algorithm that reveals subquadratic runtime in the number of primitives.
- a spatial split-based algorithm, which applies temporary spatial splits to push quality of BVHs even further
- an approach for reducing the forecasting error of the ray tracing performance predictor from Aila et al. [4] which also enables more accurate predictions for primary rays and
- a comparison with a related algorithm proposed by Popov et al. [5] and a hybrid of their algorithm with ours.

This paper is an extended version of an earlier conference paper by Wodniok and Goesele [6] and adds the last three contributions.

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 [1]. 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:

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

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.

The state-of-the-art greedy top-down plane-sweeping construction from MacDonald and Booth [2] 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:

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

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. [7] introduced an $O(n \log(n))$ algorithm for SAH-based BVH construction. This is achieved by replacing the sorting step with a $O(n)$ binning step adapted from binned kd-tree construction by Popov et al. [8]. For this a limited number of equidistant split planes with respect to the bounds of the primitives are computed. Consecutive split planes define bins to which primitives are distributed. A best split candidate can then be computed from the binning results in linear time. With a sufficient number of bins hierarchy quality is practically identical to full sweep construction. Fabianowski et al. [9] 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.

2.1. Fast Construction

Lauterbach et al. [10] proposed three GPU-based BVH construction algorithms with different trade-offs between tree quality and construction time: The median split-based linear BVH (LBVH) algorithm is fast but has poor tree quality. The second algorithm is a parallel approach for full binned-SAH BVH construction (see Wald [11]) 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 [12], and Garanzha et al. [13] proposed much faster implementations for the median split and the hybrid algorithm called hierarchical LBVH (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.

Ganestam al. [14] proposed a hybrid algorithm called *BONSAI*. Similar to the original hybrid algorithm from Lauterbach et al. [10] it first performs a spatial-median split partitioning on the input to produce sufficiently small chunks of spatially coherent primitives. Then for each chunk top-down plane-sweeping SAH-based BVHs are constructed in parallel. Finally, in the spirit of HLBVH an SAH-based top-level BVH is constructed on the chunks. Quality of the final hierarchy on average is identical to full sweep-based construction but construction time is much lower.

Bittner et al. [15] presented the first incremental BVH construction algorithm which can produce high quality BVHs. This was previously not considered possible. While average hierarchy quality is higher than for a full sweep construction, the actual average measured performance is slightly lower. Nonetheless, construction is faster than top-down sweep construction.

As dynamic scenes are not in focus of our work we only give a very brief overview on related algorithms. One approach is to simply construct a new BVH each frame as fast as possible. This was the purpose of LBVH from Lauterbach et al. [10] and derived work (Pantaleoni and Luebke [12], and Garanzha et al. [13]). For state-of-the-art in refitting-based approaches we refer to the algorithm from Yin et al. [16] and the references therein.

2.2. Higher Quality BVHs

Stich et al.'s [3] offline spatial split BVH (SBVH) algorithm drastically improves tree quality for scenes with widely

varying degree of tessellation. Their 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 [17] 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. Ganestam et al. [18] present an alternative triangle pre-splitting algorithm, which can optionally directly be integrated into the clustering phase of the BONSAI algorithm. They report traversal performance improvements to be similar to Karras and Aila [17] while producing less duplicate triangles.

Plane sweeping only generates left-right partitions with respect to a splitting plane. Popov et al. [5] proposed to allow more general partitionings in order to achieve smaller SAH cost partitions if possible. This is done by pre-generating a set of more general child bound pairs and distributing the primitives to these sets. They call this process geometric partitioning. 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 spatial-median split BVHs. This improved measurements but results were still inferior to the standard plane sweeping algorithm.

An alternative construction approach is the agglomerative clustering algorithm from Walter et al. [19]. Initially each primitive is in its own leaf node. Then, in a bottom-up fashion the algorithm iteratively generates a new parent for the pair of nodes which produces the smallest parent bounds surface area. The newly fostered child nodes are removed from the list of candidates while the new parent node is added to this list. The authors empirically show that runtime of their implementation is somewhere between linear and quadratic. While this algorithm can produce hierarchies with higher quality than top-down SAH-based construction, Aila et al. [4] observed that it can also produce hierarchies with drastically lower quality, or expose low traversal performance even when SAH cost is low. Gu et al. [20] presented a more efficient but approximative implementation of this algorithm, which also inherits its downsides.

Aila et al. [4] 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 constructed BVHs with lower SAH cost improved measurements less than expected or even decreased performance (e.g., Popov et al. [5], Walter et al. [19], and Bittner et al. [15]). At the same time BVHs with similar SAH cost 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. [4] for computation of EPO for a BVH. The traversal cost predictor p is a convex combination of SAH and EPO:

$$p = SAH \cdot (1-\alpha) + EPO \cdot \alpha \tag{3}$$

Here, α is a scene dependent constant. Given the actual average measured traversal cost m for a BVH it is assumed that

$$m \sim p \tag{4}$$

holds. This predictor is only designed for secondary diffuse rays and scalar traversal. α values as high as 0.98 have been computed. 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 not necessarily the best performing one. Aila et al. [4] discovered 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. This result is especially important for the next section, where we describe our algorithm.

Finally, for SAH kd-tree construction Havran [21] 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. [4] 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

```
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
(bestP, bestC) ← (∅, ∞)
partitions ← GeneratePartitions(node)
foreach partition ∈ partitions do
    t_l ← BuildTemporaryBVH(partition.left, c_t, c_i)
    t_r ← BuildTemporaryBVH(partition.right, c_t, c_i)
    c_l ← ComputeSAH(t_l, c_t, c_i)
    c_r ← ComputeSAH(t_r, c_t, c_i)
    (p_l, p_r) ←
      ComputeIntersectionProbabilities(partition)
    c_split ← c_t + p_l c_l + p_r c_r
    if c_split < bestC then
        (bestP, bestC) ← (partition, c_split)
    end
end
```

Algorithm 1: Pseudo-code for determining the best node split with recursive SAH.

their recursive SAH values. The rating function for a split then becomes

$$c_{split} = c_t + p_l c(t_l) + p_r c(t_r), \tag{5}$$

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. Pseudo-code for determining the best split is given in Algorithm 1. A depiction of candidate cost computation with temporary BVH construction is given in Figure 1. Note that `BuildTemporaryBVH` and `ComputeSAH` can be merged into a single function as SAH cost can be computed incrementally during construction. There is no need to store the temporary hierarchy at any point. It is also not beneficial to store the hierarchy of the best candidate for reuse as it only has the quality of the standard algorithm and different hierarchies on smaller subsets have to be constructed for the newly created child nodes.

Seen from a different angle, our approach can be interpreted as a middle ground between the *NP*-hard algorithm proposed by Havran [21] and the recursive SAH evaluation on temporary spatial-median split trees used by Popov et al. [5] in terms of computational complexity and BVH quality. The difference is, that we give a more representative cost estimate for the split candidates than a spatial-median split as it much closer reflects the way the main BVH itself is constructed. Spatial-median split construction does not incorporate SAH in any way. Thus, SAH values retrieved from temporary BVHs constructed in this way are unreliable for construction that aims at reducing SAH.

We will now proceed with the algorithmic aspects of our approach, which we call recursive SAH-based BVH construction

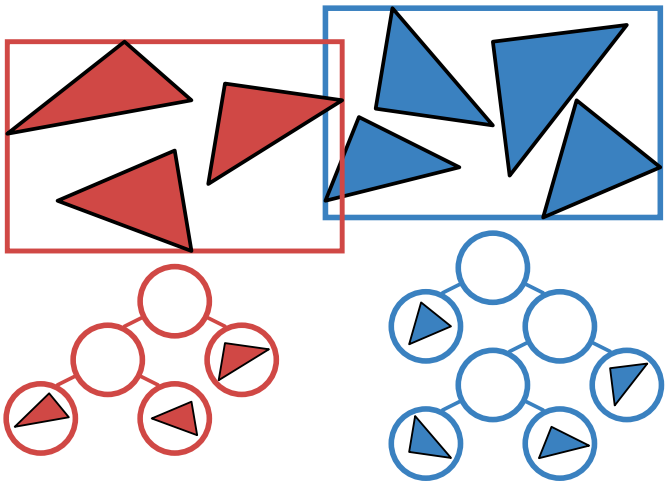

Figure 1: Depiction of candidate cost evaluation in Algorithm 1 for a candidate partition (red and blue boxes) generated by `GeneratePartitions` for a set of input primitives with e.g. a plane-sweeping or binning approach. Classic SAH candidate cost assumes that both sides of the partition will stay leaves. With the `BuildTemporaryBVH` function we construct separate temporary BVHs for each side of the partition and compute their SAH cost to obtain a candidate cost estimate. Depending on the construction strategy used in `BuildTemporaryBVH` quality of the estimate can be better or worse than for the classic candidate cost.

(RBVH). The `GeneratePartitions` 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. [5] are also possible. The `BuildTemporaryBVH` function for temporary BVH construction can also use arbitrary construction schemes but we only consider EPO-reducing greedy top-down plane-sweeping or binning construction. Alternatives would be the fast to construct bottom-up LBVH and HLBVH approaches or the agglomerative algorithm from Walter et al. [19]. Data collected by Aila et al. [4] shows that these algorithms can increase EPO drastically, which renders them a poor choice for `BuildTemporaryBVH`. Depending on whether we choose sweeping or binning for `GeneratePartitions` and `BuildTemporaryBVH` 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 [22] for kd-trees to BVHs allows to implement an algorithm with lower complexity. For this three copies of all primitives have to be sorted in each dimension in a pre-computation step. This allows to find the best split in $O(n)$ steps. The arrays which do not correspond to the dimension of the best split can then be updated in $O(n)$. As a result the whole algorithm has a complexity of $O(n \log n)$. To the best of our knowledge this has not been done before. Using binning construction also has $O(n \log n)$ complexity, 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 our proposed Wald and Havran [22] 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:

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

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

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

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. An algorithm based on the common naïve approach which sorts in every step has a higher complexity of $O\left(N^2\log^2 N\right)$. A derivation of this result is provided found in Appendix A.

Asymptotic complexity of binning-based temporary BVH construction is the same as for sweep-based construction akin to Wald and Havran [22]. 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 relevant, 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. The full derivation is described in Appendix B. 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 construction and our approach for 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. [3] 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 5). This does not allow us to compare split candidates from those techniques in a meaningful way. We simply have to adapt the `GeneratePartitions` 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. This variant is included into the evaluation, where we call it recursive spatial split bounding volume hierarchy (RSBVH).

Going one step further we can include spatial splits into `BuildTemporaryBVH` to construct temporary SBVHs. This again allows to find better split candidates. To unfold the full potential of this approach temporary SBVHs have to be constructed for spatial partitions as well as object partitions generated by `GeneratePartitions`. Constructing temporary SBVHs but not generating spatial partitions in `GeneratePartitions` is detrimental for BVH quality as the main BVH might be misguided into directions which are only beneficial if spatial splits are enabled. RSBVH construction with temporary SBVHs is also included into the evaluation.

4. Improving Accuracy of the SAH-EPO Predictor

In our previous work [6] we observed that while the achieved correlations of p from Equation 3 and average measured traversal cost m are higher than 0.99 in most scenes, the average predicted performance improvements were up to 50% higher than the actual average measured performance improvement, but could not provide an explanation for this. Before presenting our solution to this problem we first recap how the scene dependent α value of Equation 3 is computed.

Given a set of BVHs $\mathscr{B}$ constructed for a scene and triples (SAH_i, EPO_i, m_i) of SAH, EPO, and measured traversal cost for each BVH $i \in \mathscr{B}$ Aila et al. [4] compute α by solving

$$\alpha = \underset{x\in[0,1]}{\arg\max}\ \ r\left(\{(p_i(x), m_i) \,|\, i \in \mathscr{B}\}\right), \tag{10}$$

where r is the centered sample Pearson correlation coefficient and $p_i(x)$ corresponds to Equation 3 for a pair (SAH_i, EPO_i) of a BVH i. α can be computed by simply sampling the $[0,1]$ range and selecting the α which gives the highest correlation. For a chosen α we can then compute the *mean absolute percentage error* (MAPE) of the expected and measured speedups $p_{ij} = p_i/p_j$ and $m_{ij} = m_i/m_j$ for pairs of BVHs i, j:

$$MAPE = \frac{1}{|\mathscr{B}|^2 - |\mathscr{B}|} \sum_{\substack{i,j\in\mathscr{B} \\ i\neq j}} \left| \frac{p_{ij} - m_{ij}}{m_{ij}} \right| \tag{11}$$

Computing α with the centered Pearson correlation resulted in average MAPE errors as high as 65%.

The reason for this high error is that solving Equation 10 is equivalent to finding the α, which gives the smallest least squares error of a simple linear regression with an intercept term. The intercept term breaks, however, the proportionality assumption between measured and estimated cost (Equation 4) as it allows to not contain the origin. To force regression through the origin we propose to use the *uncentered sample Pearson correlation* (equivalent to the *cosine similarity* function):

$$r_{uncentered} = \frac{\sum_i p_i m_i}{\sqrt{\sum_i (p_i)^2}\ \sqrt{\sum_i (m_i)^2}} \tag{12}$$

Solving Equation 10 with the uncentered correlation is equivalent to finding the α, which gives the smallest least squares error of a simple linear regression without an intercept term. Table 1 shows the computed α values along with their corresponding correlation and MAPE for the centered and uncentered sample Pearson correlation coefficients for all test scenes. Though only intended for secondary diffuse rays we also computed a separate α for primary rays.

Table 1 shows that our approach reduces MAPE significantly. The obtained α values can differ drastically from the original approach with an extreme case of 0.89 and 0.38 for *Babylon* with diffuse rays, where MAPE is reduced from 60.2% to 6.2%. Overall we can observe that our approach gives less of an emphasis on EPO for diffuse rays in all scenes. On average we reduce MAPE from 22% to 4.6% for diffuse rays.

Our measurements show that the SAH-EPO predictor indeed gives higher errors for primary rays for which it is not intended. MAPE is highest for *Babylon* with 116.5%. But our approach was able to reduce the error to 9.4%. On average we reduced MAPE from 29.2% to 8.1%. Unlike for diffuse rays, we cannot observe the overall trend that α values with our approach are lower than with the centered correlation. Note that average MAPE for primary rays with the uncentered correlation is also lower than average MAPE with the centered correlation and diffuse rays.

5. Evaluation Setup

Our test platform is equipped with two Intel Xeon E5-2650 v2 octacore CPUs. Construction timings are included in our results. Our implementation of the algorithms only parallelized the for-each loop in Algorithm 1. This is not optimal as it introduces global synchronization between every node split. With additional effort it is possible to parallelize the whole construction process, though.

5.1. Scenes and Algorithms

To evaluate our proposed construction algorithm we measured the impact on SAH, EPO and traversal performance. We used a number of freely available test scenes (see Table 2) partly from McGuire [23] and the Mitsuba renderer [24]. We only evaluated the $O(N \log^2 N)$ binning-binning algorithm as the superquadratic complexity of the sweep-sweep and and sweep-binning algorithm proved to be impractical in practice. The RSBVH algorithm and the extension with temporary SBVHs (RSSBVH) from Section 3.2 is also included into the evaluation. As the baseline construction algorithm we chose the standard plane-sweeping approach. We also evaluated our RBVH algorithm with recursive SAH evaluation on temporarily built spatial-median split BVHs (RMBVH).

Further, we included the geometric partitions with (RMGBVH) and without (GBVH) temporarily built spatial-median split BVHs from Popov et al. [5]. In this connection we also evaluated the inclusion of geometric partitions into the `GeneratePartitions` function of our RBVH algorithm (RGBVH).

As the baseline for construction with spatial splits we chose the SBVH algorithm from Stich et al. [3]. SBVH allows to

	Primary rays						Diffuse rays					
	Centered			Uncentered			Centered			Uncentered		
Scene	α	corr.	MAPE	α	corr.	MAPE	α	corr.	MAPE	α	corr.	MAPE
Babylon	0.98	0.993	116.5%	0.53	0.997	**9.4%**	0.89	0.997	60.2%	0.38	0.998	**6.2%**
Bubs	0.00	0.935	**5.7%**	0.00	0.998	**5.7%**	0.25	0.989	8.0%	0.00	0.999	**4.6%**
Conference	0.49	0.971	9.2%	0.45	0.998	**8.9%**	0.64	0.999	9.9%	0.27	0.999	**2.5%**
Epic	1.00	0.936	62.2%	0.66	0.996	**10.9%**	1.00	0.955	65.9%	0.60	0.998	**8.4%**
Fairy	0.48	0.855	**7.4%**	0.80	0.997	8.0%	0.75	0.928	3.6%	0.68	0.999	**3.4%**
Hairball	1.00	0.886	74.9%	0.87	0.993	**13.2%**	0.96	0.993	32.2%	0.72	0.999	**2.8%**
Powerplant	0.75	0.997	23.0%	0.47	0.999	**4.1%**	0.61	0.999	17.3%	0.28	0.999	**3.4%**
Rungholt	0.00	0.602	**2.6%**	0.00	0.999	**2.6%**	0.00	0.664	**2.6%**	0.00	0.999	**2.6%**
San Miguel	0.61	0.990	16.7%	0.43	0.999	**9.0%**	0.66	0.993	28.2%	0.28	0.999	**7.8%**
Sibenik	0.47	0.995	5.9%	0.62	0.999	**3.5%**	0.74	0.997	12.7%	0.38	0.999	**3.1%**
Soda	0.61	0.975	14.0%	0.45	0.998	**9.7%**	0.61	0.998	11.8%	0.35	0.999	**4.6%**
Sponza	0.74	0.920	**12.3%**	0.75	0.995	12.4%	0.79	0.979	10.0%	0.62	0.999	**5.9%**
Average	0.59	0.921	29.2%	0.50	0.998	**8.1%**	0.66	0.958	21.9%	0.38	0.999	**4.6%**

Table 1: Listing of determined α values for primary and diffuse rays for all scenes used for benchmarking our algorithms obtained with the centered and uncentered sample Pearson correlation. Each α is accompanied by its corresponding correlation coefficient and mean absolute percentage error (MAPE). Lowest MAPE is highlighted for each combination of ray type and scene.

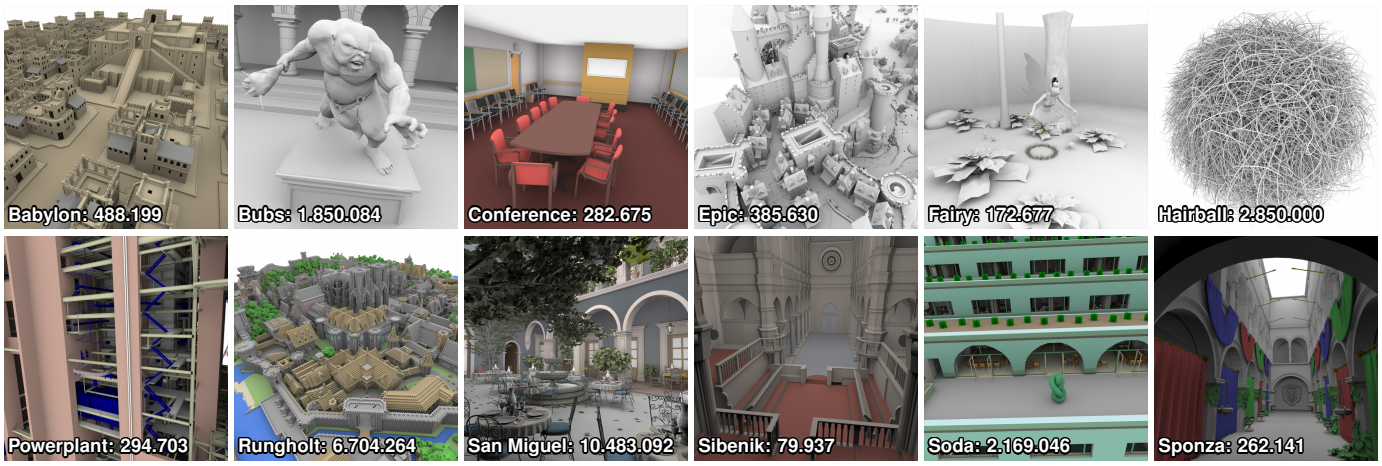

Figure 2: Listing of all twelve scenes used for benchmarking our algorithms along with their number of primitives.

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* where 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. The same parameter values are also applied to the temporary SBVH construction of the RSSBVH algorithm.

In total we have up to nine different BVHs per scene. We omit results for RGBVH and RMGBVH for the five largest scenes as expected total computation time is several months (a year for RGBVH on San Miguel) and we expect the results to be similar to the results for the smaller scenes.

For the main BVH we have 256 object split bins and 128 spatial bins. The number of object and spatial bins for temporary BVH construction is 32. SAH build constants were set to $(c_t, c_i) = (1.2, 1.0)$. They correspond to the constants of the GPU ray tracing kernels from Aila et al. [25], which we used for collecting traversal statistics. All BVH algorithms and configurations along with abbreviations we used for them are listed in Table 2.

5.2. *Performance Measurements*

We measure performance of front-to-back traversal for primary rays and secondary diffuse rays to compare quality of the different BVHs. To get implementation and platform 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 and BVH. Combined with the SAH constants we define the average measured traversal cost

$$m = \overline{n}_s c_t + \overline{n}_t c_i. \tag{13}$$

Abbr.	Algorithm	o	s	t
BBVH	Baseline Plane-Sweep [2]	-	-	-
SBVH	Baseline SBVH [3]	256	128	-
RBVH	RSAH	256	-	32
RMBVH	BBVH + temp. median splits	256	-	-
RSBVH	RSAH + SBVH	256	128	32
RSSBVH	RSBVH + temp. SBVH	256	128	32
GBVH	Geometric splits [5]	256	-	-
RMGBVH	GBVH + temp. median splits	256	-	-
RGBVH	GBVH + RSAH	256	-	32

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. In case of RSSBVH the number of temporary spatial bins is equal to t.

We also give results for predicted traversal cost with EPO according to Equation 3. For this we computed the scene dependent α values with the novel approach described in Section 4 using the uncentered Pearson correlation (see Table 1 for the specific α values).

6. Results

All measurements are collected in Table 5 and Table 6. To give a more condensed view of the results Table 3 shows relative improvements averaged over all scenes with BBVH and SBVH as baseline. Measurements for each scene are sorted from best to worst with respect to measured traversal cost m of diffuse rays.

Our spatial split-based algorithms improved 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 with a slightly higher improvement for primary rays. The average improvement of SBVH is higher than for our RBVH algorithm without spatial splits. SBVH improves SAH only slightly compared to RBVH, but outperforms RBVH in EPO improvements. The RMBVH algorithm which, unlike RBVH, uses temporary spatial median splits performs worse than BBVH and overall performs second worst of all evaluated algorithms. The SAH costs of temporarily constructed spatial median split BVHs were less representative for subtree costs than the standard cost estimate from Equation 2 and misguided the construction process.

In the *Rungholt* scene no algorithm was able to significantly outperform the other algorithms. The baseline BBVH already gives the third best results and its output is identical to SBVH, which was not able to find any beneficial spatial splits. The scene mainly consists of triangles of the same shape and size which are also parallel to the main coordinate system planes. BBVH seems to be sufficient for scenes with such characteristics. This is also reconfirmed by identical results we obtained for the *Lost Empire* scene available from McGuire [23] which has the same characteristics but is not included in our results in the interest of brevity.

6.1. Geometric Object Partitions

On average our algorithms perform better than algorithms based on the geometric splits from Popov et al. [5]. In fact, we can reconfirm results from Popov et al. that GBVH and RMGBVH perform worse than standard plane-sweeping. Given that spatial median splits already decreased performance of RMBVH over BBVH, it is no surprise that RMGBVH performs worse than GBVH. RMBGVH proved to be the worst performing of all evaluated algorithms. Only RGBVH, our RBVH algorithm extended with geometric splits, managed to produce better results than BBVH on average from all algorithms with geometric splits. Still, RGBVH proved to be less effective than RBVH.

6.2. Construction Time

RBVH- and RSBVH- based construction time is one to two orders of magnitude higher than for the baseline. RSSBVH additionally takes 2 to 8 times longer than RSBVH. One possibility to reduce construction time is to reduce the number of object and spatial bins for the main and temporary BVHs of our algorithms, as this reduces the number of temporary BVHs to construct. Therefore we evaluated configurations where the number of object and spatial bins is set to 64 or 32. Average results for relative construction time, quality, and measured cost are collected in Table 4. On average construction time is reduced by a factor of 2.5 and 4.5 with 64 and 32 bins respectively. Surprisingly, the effects on quality and measured performance are small on average. Performance of RBVH decreases by just half a percent with both reduced bin counts for diffuse rays. RSBVH performance decreases by up to one and two percent for diffuse and primary rays, while RSSBVH performance actually improved slightly for diffuse rays despite having the largest average increase in EPO.

All geometric split-based algorithms are the most expensive ones with RGBVH already taking 12 hours to construct for our smallest scene *Sibenik* and more than two days for *Babylon*.

7. Discussion

The proposed RBVH, RSBVH and RSSBVH builders managed to reduce SAH as well as EPO by a significant amount. All three 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. [4]. 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 22% on average is quite impressive. If primitive duplication is not desired RBVH might be an alternative to SBVH.

Compared with SBVH, RSBVH and RSSBVH on average produce twice the number of duplicates. An exception to this is *Hairball*, where we measured five times as many duplicates.

Baseline	Algorithm	Avg. (Min/Max) reduction (%)					
				Primary rays		Diffuse rays	
		SAH	EPO	p	m	p	m
BBVH	RSSBVH	**-21.6** (-4.0 / -35.8)	**-71.9** (-20.8 / -90.6)	**-32.6** (-4.0 / -47.0)	**-24.0** (-2.2 / -37.1)	**-28.3** (-4.0 / -39.7)	**-22.5** (-0.9 / -32.2)
	RSBVH	-19.4 (-3.9 / -34.7)	-68.1 (-19.6 / -85.8)	-30.1 (-3.9 / -41.3)	-23.9 (+0.6 / -37.0)	-26.0 (-3.9 / -35.9)	-22.4 (+0.3 / -32.6)
	SBVH	-12.5 (+3.8 / -26.8)	-58.2 (0.0 / -82.8)	-22.3 (0.0 / -38.7)	-17.7 (+1.7 / -33.7)	-18.6 (0.0 / -33.3)	-18.5 (0.0 / -32.0)
	RBVH	-11.7 (-2.7 / -33.3)	-22.2 (-1.4 / -65.3)	-13.3 (-2.7 / -33.3)	-10.4 (+0.9 / -28.3)	-12.7 (-2.7 / -33.3)	-8.9 (+0.4 / -23.7)
	RGBVH	-10.6 (-4.0 / -17.8)	-11.4 (+8.3 / -42.5)	-11.1 (-2.7 / -24.9)	-7.7 (+2.3 / -20.2)	-11.0 (-3.2 / -22.0)	-7.5 (-1.9 / -22.8)
	GBVH	+2.6 (+21.1 / -21.8)	+31.7 (+96.0 / -63.6)	+9.8 (+36.4 / -21.8)	+22.2 (+46.2 / -6.2)	+7.1 (+30.4 / -21.8)	+11.9 (+31.6 / -13.8)
	RMBVH	+22.1 (+69.4 / -12.9)	+60.6 (+171.0 / -7.3)	+29.6 (+90.7 / -12.9)	+33.4 (+84.5 / -7.6)	+26.4 (+80.1 / -12.9)	+26.6 (+74.8 / -4.9)
	RMGBVH	+26.7 (+70.0 / -0.5)	+113.7 (+228.6 / +29.2)	+45.0 (+103.3 / +10.4)	+60.8 (+124.3 / -1.0)	+37.2 (+86.8 / +7.3)	+44.9 (+93.7 / +2.6)
SBVH	RSSBVH	**-10.3** (-4.0 / -19.4)	**-35.1** (-1.7 / -70.6)	**-13.5** (-4.0 / -27.5)	**-7.4** (-1.7 / -17.6)	**-12.1** (-4.0 / -20.1)	**-4.7** (+0.2 / -15.1)
	RSBVH	-7.8 (-2.0 / -15.6)	-18.6 (+52.6 / -70.6)	-9.8 (-0.7 / -28.4)	-7.1 (+2.1 / -18.3)	-8.9 (-2.4 / -18.6)	-4.6 (+0.3 / -15.4)
	RBVH	+1.7 (+23.8 / -12.1)	+143.7 (+394.0 / -20.1)	+13.4 (+46.9 / -8.9)	+10.7 (+46.0 / -9.3)	+8.5 (+35.9 / -8.9)	+12.9 (+36.0 / -3.7)
	RGBVH	+4.9 (+21.2 / -7.5)	+198.5 (+451.3 / +25.0)	+22.4 (+50.2 / +0.2)	+18.5 (+51.3 / -8.3)	+15.0 (+37.5 / -3.0)	+18.4 (+39.8 / -1.3)
	GBVH	+18.3 (+46.0 / -0.2)	+331.6 (+894.5 / +22.3)	+44.2 (+98.1 / -0.2)	+51.1 (+93.2 / +0.4)	+33.4 (+75.3 / -0.2)	+39.4 (+72.2 / -0.2)
	RMBVH	+42.4 (+124.4 / -0.4)	+459.9 (+1081.3 / +2.0)	+72.7 (+194.9 / +3.2)	+66.9 (+178.5 / -0.7)	+59.3 (+157.3 / +3.2)	+59.1 (+149.0 / +2.7)
	RMGBVH	+51.4 (+125.2 / -1.0)	+700.9 (+1710.1 / +60.3)	+104.7 (+219.3 / +13.7)	+111.3 (+227.8 / -2.6)	+80.8 (+171.9 / +7.6)	+88.7 (+176.0 / +5.0)

Table 3: Average, minimum, and maximum relative reduction of SAH, EPO, as well as predicted (p) and measured (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.

Algo.	Bins	Avg. reduction (%)				
		Time	SAH	EPO	m_p	m_d
RBVH	64	-64.5	+0.7	+0.3	+0.2	+0.5
	32	-78.1	+1.2	+0.3	-0.2	+0.5
RSBVH	64	-57.3	-0.2	0.0	+1.7	+0.4
	32	-75.5	+0.6	+4.3	+2.2	+1.0
RSSBVH	64	-61.3	0.0	0.0	-0.5	-0.2
	32	-79.0	+0.1	+5.8	+0.7	-0.2

Table 4: Average relative difference of construction time, SAH, EPO, as well as measured traversal cost of primary (m_p) and diffuse (m_d) rays for the RSAH-based algorithms in percent when the number of object- and space partitioning bins for the main BVH are reduced to 64 or 32.

Our subtree cost estimate predicts more accurately whether a spatial split will pay off and seems to find more opportunities where a spatial split is more beneficial than an object split. As a result more spatial splits are performed. The slightly more accurate cost estimate of RSSBVH results in a slightly higher number of duplicates compared to RSBVH.

The downside of our proposed algorithms is the large increase in construction time, which renders them unsuitable for realtime applications. Our largest test scene, *San Miguel*, took almost 4 hours to construct with RSBVH and almost 14 hours with RSSBVH. 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. Results from Section 6.2 showed that reducing the number of bins might be an option as construction can be sped up by a factor of 4.5 without severe

effects on measured performance. Alternatively, 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.

7.1. Insufficiency of the SAH-EPO Metric

Compared with RSBVH, RSSBVH manages to reduce SAH 1.3 times and EPO twice as much over SBVH. But disappointingly the average reduction of measured traversal cost is only about 3% higher. Actually, RSSBVH only performed best in five out of twelve scenes for diffuse rays, while RSBVH performed best in the remaining seven scenes. Considering that RSSBVH managed to achieve the highest reduction in SAH *and* EPO at the same time for eight scenes, this is an unexpected result. Though both SAH and EPO are clearly smaller in *Babylon*, *Bubs*, *Conference*, and *Sponza* than with any other builder, measured traversal cost is higher than for RSBVH in those scenes. The most severe example for this observation is *Sponza*, where RSSBVH also falls behind SBVH. RSSBVH has 14% lower SAH and 60% lower EPO cost than SBVH for this scene. The predicted decrease in traversal cost is 17%. But measured traversal cost is slightly higher than for SBVH. Both, the SAH metric and the combined SAH-EPO metric (which is designed for diffuse rays), fail to explain these observations. Thus, there must be an unidentified effect which is not captured in the former metrics. We were unable to identify this extra cost and consider it an open question for future work.

7.2. Inferiority of Geometric Object Splits

The GBVH, RMGBVH, and RGBVH algorithm can be interpreted as extending BBVH, RMBVH, and RBVH with geometric object splits from Popov et al. [5]. In all three cases adding geometric object splits resulted in decreasing the quality of the original algorithms. Considering that geometric splits only extend the set of available partitions of the original algorithms and are only selected if considered beneficial this seems perplexing at first. Analyzing constructed BVHs we found that GBVH indeed finds geometric splits where c_{split} from Equation 2 is up to 40% lower than the best split found with plane-sweeping. To explain this we first write Equation 1 for the SAH cost of a BVH in its iterative form. This partitions the cost of a BVH in costs C_I for processing inner nodes and costs C_L for processing leaves:

$$c_{BVH} = c_I + c_L = c_t \sum_{i \in I} p_i + c_i \sum_{l \in L} p_l |l| \qquad (14)$$

I is the set of inner nodes and L is the set of leaves. p_i and p_l are the probabilities of intersecting an inner or leaf node w.r.t. the root bounds and $|l|$ is the number of primitives in a leaf. We found that while c_L for GBVH is essentially identical to BBVH larger SAH costs for GBVH mainly stem from a larger c_I. Further analysis revealed that GBVH has a strong tendency to create children with a larger discrepancy in surface area of the children bounds than BBVH. Let $\underline{A}$ and $\overline{A}$ denote the surface area of the smaller and larger child respectively. On average $\underline{A}$ is smaller and $\overline{A}$ is bigger for GBVH compared with BBVH. This seems to be necessary to allow GBVH to locally find smaller c_{split}. But at the same time $\underline{A} + \overline{A}$ turns out to be larger than for BBVH on average which directly causes the increase in C_I. That is, GBVH is worse in reducing the average area of the children bounds, an effect not captured in Equation 2. This increase in c_I is most severe in the upper levels, where surface area of bounds is larger anyway. Our results comply with the observation from Aila et al. [4]: It is not clear that locally minimizing Equation 2 globally reduces SAH, but the minimization maximizes saved worst-case triangle cost. On average overlap of children bounds is also increased. This might be a direct consequence of the larger average child bounds and explains the increase in EPO of algorithms with geometric splits.

We also observed the increase in average children bounds area for RMGBVH and RGBVH. Both algorithms are based on Equation 5, which includes the cost of temporarily constructed hierarchies, but c_I is also increased for those algorithms. Though this time the increase in cost for the direct child nodes is included in the candidate cost, the extra cost gets lost in the costs for the temporary hierarchies. Concluding, the supremacy of greedy top-down builders declared by Aila et al. [4] does not hold for algorithms based on geometric splits.

We conducted experiments with GBVH where we included traversal costs for children into Equation 2. This proofed to be insufficient as quality decreased. Overcoming the blindness of Equation 2 towards increases in c_I should also benefit the plane-sweeping algorithm and is left as an open problem for future work.

8. Future Work

There are several directions for future work. One direction would be to find a BVH quality metric which explains the observations made for RSSBVH in Section 7.1.

Another direction would be 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. Further, we would have to actually store temporary hierarchy nodes, which so far is not necessary with our algorithms (see Section 3). In this regard fast and accurate determination of α for unknown scenes is also an interesting problem.

The treelet-based BVH optimization algorithm proposed by Karras and Aila [17] is only practical for small treelet sizes as their minimum SAH BVH construction algorithm has $O(\exp n)$ computational complexity and space requirements. While our RBVH algorithm does not construct minimum SAH BVHs, its computational complexity and space requirements allows for much larger treelets. Combined with its high quality output it would be interesting to see what can be achieved.

Finally, additionally to the EPO metric Aila et al. [4] proposed the leaf count variability (LCV) metric, which in a convex combination with SAH and EPO can explain SIMD performance of the efficient GPU ray tracing kernels of Aila et al. [25]. Aila et al. [4] showed that top-down greedy SAH-based construction algorithms implicitly reduce LCV besides EPO. We assume that this property should be naturally inherited and boosted by our algorithms. That is, our BVHs might be specially suited for SIMD traversal. Experimental validation is left for future work.

Acknowledgments. 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*, A. Grynberg and G. Ward for *Conference*, University of Utah for *Fairy*, S. Laine for *Hairball*, UNC for *Powerplant*, R. Vance for *Bubs*, G. M. L. Llaguno for *San Miguel*, M. Dabrovic for *Sibenik*, UC Berkeley for *Soda*, kescha for *Rungholt*, Epic Games for *Epic*, and F. Meinl for *Crytek-Sponza*.

[1] Goldsmith J, Salmon J. Automatic creation of object hierarchies for ray tracing. IEEE Computer Graphics and Applications 1987;7:14–20.
[2] MacDonald DJ, Booth KS. Heuristics for ray tracing using space subdivision. The Visual Computer 1990;6:153–66.
[3] Stich M, Friedrich H, Dietrich A. Spatial splits in bounding volume hierarchies. In: Proc. HPG. 2009, p. 7–13.
[4] Aila T, Karras T, Laine S. On quality metrics of bounding volume hierarchies. In: Proc. HPG. 2013, p. 101–7.
[5] Popov S, Georgiev I, Dimov R, Slusallek P. Object partitioning considered harmful: space subdivision for bvhs. In: Proc. HPG. 2009, p. 15–22.
[6] Wodniok D, Goesele M. Recursive sah-based bounding volume hierarchy construction. In: Proc. Graphics Interface. 2016, p. 101–7.
[7] Wald I, Boulos S, Shirley P. Ray tracing deformable scenes using dynamic bounding volume hierarchies. ACM TOG 2007;26:6.
[8] Popov S, Günther J, Seidel HP, Slusallek P. Experiences with streaming construction of SAH KD-trees. In: Proc. IEEE IRT. 2006, p. 89–94.
[9] Fabianowski B, Fowler C, Dingliana J. A Cost Metric for Scene-Interior Ray Origins. In: Proc. EG. 2009, p. 49–52.
[10] Lauterbach C, Garland M, Sengupta S, Luebke D, Manocha D. Fast BVH construction on GPUs. CGF 2009;28:375–84.
[11] Wald I. On fast construction of SAH-based bounding volume hierarchies. In: Proc. IEEE IRT. 2007, p. 33–40.
[12] Pantaleoni J, Luebke D. HLBVH: Hierarchical LBVH construction for real-time ray tracing of dynamic geometry. In: Proc. HPG. 2010, p. 87–95.
[13] Garanzha K, Pantaleoni J, McAllister D. Simpler and faster HLBVH with work queues. In: Proc. HPG. 2011, p. 59–64.
[14] Ganestam P, Barringer R, Doggett M, Akenine-Möller T. Bonsai: rapid bounding volume hierarchy generation using mini trees. Journal of Computer Graphics Techniques 2015;4.
[15] Bittner J, Hapala M, Havran V. Incremental bvh construction for ray tracing. Computers & Graphics 2015;47:135–44.
[16] Yin M, Li S. Fast bvh construction and refit for ray tracing of dynamic scenes. Multimedia tools and applications 2014;72:1823–39.
[17] Karras T, Aila T. Fast parallel construction of high-quality bounding volume hierarchies. In: Proc. HPG. 2013, p. 89–99.
[18] Ganestam P, Doggett M. Sah guided spatial split partitioning for fast bvh construction. In: CGF. 2016, p. 285–93.
[19] Walter B, Bala K, Kulkarni M, Pingali K. Fast agglomerative clustering for rendering. In: Proc. IEEE IRT. 2008, p. 81–6.
[20] Gu Y, He Y, Fatahalian K, Blelloch G. Efficient bvh construction via approximate agglomerative clustering. In: Proc. HPG. 2013, p. 81–8.
[21] Havran V. Heuristic ray shooting algorithms. Ph.d. thesis; 2000.
[22] Wald I, Havran V. On building fast kd-trees for ray tracing, and on doing that in O(N log N). In: Proc. IEEE IRT. 2006, p. 61–9.
[23] McGuire M. Computer graphics archive. `http://graphics.cs.williams.edu/data`; 2011. [accessed: 2016.08.02].
[24] Jakob W. Mitsuba renderer. `http://www.mitsuba-renderer.org`; 2010. [accessed: 2016.07.19].
[25] Aila T, Laine S. Understanding the efficiency of ray traversal on GPUs. In: Proc. HPG. 2009, p. 145–9.

Appendix A. Naïve Sweep-Sweep Construction Complexity

The common naïve top down SAH-based plane-sweeping construction algorithm implementation according to MacDonald and Booth [2] sorts all node primitives every time it determines the best SAH-based split. This results in an overall complexity of $O\left(N\log^2 N\right)$. For a node with N primitives a sweep-sweep algorithm based on this implementation first has to sort its primitives in $O\left(N\log N\right)$ and then construct $2(N-1)$ temporary BVHs for its $N-1$ candidate partitions. This results in the following recurrence relation:

$$T(N) = N\log N + 2\left(\sum_{i=1}^{N-1} i\log^2 i\right) + 2T\left(\frac{N}{2}\right)$$

Using the upper bound $\sum_{i=1}^{N-1} i\log^2 i < N^2\log^2 N$ we get:

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

At this point we drop the lower order addend:

$$\begin{aligned} T(N) &= 2\sum_{i=0}^{\log N} 2^i\left(\frac{N}{2^i}\right)^2\log^2\frac{N}{2^i} \\ &= 2N^2\sum_{i=0}^{\log N} 2^{-i}\left(\log N - i\right)^2 \\ &\to O\left(N^2\log^2 N\right) \end{aligned} \tag{A.1}$$

Appendix B. Binning-Binning Construction Complexity

In this appendix, we derive the result from Equation 7 in detail.

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

Here we use the upper bound $\log H(B) < B^2\log B$.

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

Scene	Builder	Time	Dupl.	SAH	EPO	Primary rays				Diffuse rays			
						$\overline{n}_s$	$\overline{n}_t$	p	m	$\overline{n}_s$	$\overline{n}_t$	p	m
Babylon	RSBVH	454.78s	69.19%	39.13	2.14	27.71	**2.80**	19.48	36.05	**29.52**	**4.21**	24.98	**39.63**
	RSSBVH	3285.23s	71.89%	**37.13**	**1.65**	**27.05**	2.86	**18.28**	**35.32**	29.60	4.25	**23.56**	39.77
	SBVH	16.77s	39.73%	40.47	2.60	27.68	3.44	20.35	36.66	30.41	4.91	25.99	41.40
	RBVH	250.25s	-	49.22	12.86	37.97	5.58	29.91	51.15	39.73	6.57	35.31	54.25
	RGBVH	63.82h	-	48.96	14.35	38.96	6.33	30.57	53.08	41.42	7.30	35.72	57.01
	BBVH	3.82s	-	53.72	15.12	38.67	5.49	33.21	51.89	42.64	6.98	38.95	58.15
	GBVH	268.74s	-	59.07	23.76	50.24	6.53	40.31	66.82	53.21	7.43	45.56	71.29
	RMBVH	43.87s	-	79.18	28.38	61.78	4.98	52.19	79.12	63.12	6.72	59.75	82.47
	RMGBVH	10.09h	-	85.22	47.13	90.69	7.57	64.98	116.40	87.12	7.69	70.65	112.23
Bubs	RSBVH	1874.22s	6.43%	15.82	1.87	**23.11**	2.91	15.82	30.64	**25.53**	4.11	15.82	**34.74**
	RSSBVH	3385.47s	6.46%	**15.55**	**0.99**	23.18	**2.66**	**15.55**	**30.47**	25.74	**4.07**	**15.55**	34.96
	RBVH	1500.20s	-	16.16	2.91	24.26	2.85	16.16	31.96	26.84	4.32	16.16	36.53
	SBVH	22.94s	2.46%	17.74	1.83	26.59	2.98	17.74	34.88	28.12	4.20	17.74	37.94
	GBVH	7992.01s	-	18.96	3.05	31.64	3.82	18.96	41.78	30.67	4.47	18.96	41.28
	RMBVH	236.90s	-	21.10	8.00	31.69	3.12	21.10	41.15	34.28	4.40	21.10	45.54
	BBVH	16.52s	-	24.23	8.38	34.26	3.44	24.23	44.55	36.02	4.65	24.23	47.87
Conference	RSBVH	219.29s	82.75%	33.39	2.95	21.77	**3.23**	19.76	**29.35**	24.37	**5.25**	25.24	**34.50**
	RSSBVH	1044.97s	81.21%	**32.95**	**2.80**	22.42	4.05	**19.45**	30.95	**24.27**	5.71	**24.88**	34.83
	SBVH	5.72s	30.14%	38.35	3.54	23.64	6.57	22.76	34.93	26.32	7.71	29.03	39.29
	RBVH	126.05s	-	38.56	7.09	**21.22**	6.89	24.47	32.35	26.26	10.29	30.14	41.80
	RGBVH	18.85h	-	38.16	7.31	22.61	7.28	24.34	34.41	26.23	10.66	29.90	42.13
	BBVH	2.11s	-	46.44	9.79	26.80	7.34	30.03	39.50	32.25	10.77	36.63	49.47
	GBVH	73.86s	-	47.29	10.60	28.74	7.86	30.85	42.35	33.98	10.95	37.46	51.72
	RMGBVH	3.12h	-	54.07	15.87	42.14	8.31	36.96	58.87	42.31	11.72	43.84	62.49
	RMBVH	24.99s	-	65.56	19.04	47.72	6.97	44.73	64.24	50.59	10.42	53.10	71.12
Epic	RSSBVH	2786.21s	66.59%	**17.74**	**1.64**	**41.77**	**4.21**	**7.16**	**54.34**	**41.38**	**6.83**	**8.16**	**56.48**
	RSBVH	433.48s	64.89%	18.20	1.79	44.19	4.25	7.42	57.28	41.83	**6.83**	8.44	57.02
	SBVH	11.26s	32.73%	19.66	2.99	42.54	5.03	8.71	56.07	41.42	7.89	9.74	57.60
	RBVH	237.50s	-	19.47	6.02	52.24	6.55	10.63	69.24	49.79	9.44	11.47	69.20
	RGBVH	69.45h	-	19.27	6.33	52.12	6.88	10.77	69.43	50.98	9.75	11.57	70.92
	BBVH	3.34s	-	21.33	7.06	54.70	6.98	11.95	72.62	52.70	9.85	12.84	73.08
	RMBVH	44.19s	-	21.25	8.10	68.23	6.96	12.61	88.83	64.62	9.75	13.43	87.29
	GBVH	134.93s	-	21.59	9.94	71.29	8.33	13.94	93.88	66.03	10.18	14.66	89.41
	RMGBVH	11.59h	-	21.22	9.82	82.39	8.39	13.73	107.26	73.29	10.24	14.44	98.18
Fairy	RSBVH	174.66s	32.23%	**31.27**	2.70	**27.29**	4.79	**8.41**	**37.54**	**30.59**	8.48	**12.00**	**45.19**
	RSSBVH	642.83s	33.78%	31.52	**2.67**	27.80	**4.76**	8.42	38.12	31.02	**8.40**	12.06	45.62
	RBVH	111.01s	-	31.48	2.97	28.87	5.38	8.66	40.03	31.84	8.99	12.25	47.20
	RGBVH	12.32h	-	32.03	3.39	29.04	5.62	9.11	40.46	32.01	9.08	12.71	47.49
	SBVH	3.17s	9.31%	34.65	2.71	32.60	5.03	9.08	44.15	32.93	8.63	13.11	48.14
	BBVH	1.40s	-	33.38	3.37	31.65	5.44	9.36	43.42	33.49	9.07	13.14	49.26
	RMGBVH	7258.42s	-	34.30	4.35	31.33	5.41	10.33	43.00	34.53	9.10	14.10	50.54
	RMBVH	18.48s	-	34.50	4.31	32.06	5.37	10.34	43.84	34.83	9.05	14.14	50.85
	GBVH	34.54s	-	35.65	4.79	42.47	6.23	10.95	57.19	40.72	9.13	14.83	57.99

Table 5: Results for the first five scenes in the top row of Table 2. $\overline{n}_s$, and $\overline{n}_t$ average the average number of measured traversal steps and triangle intersection tests. p is the EPO measure for BVH performance (Equation 3) and m the average measured traversal cost (Equation 13). For each scene builders are sorted from smallest to largest m. The highest reduction of each attribute is highlighted per scene.

Scene	Builder	Time	Dupl.	SAH	EPO	Primary rays				Diffuse rays			
						$\overline{n}_s$	$\overline{n}_t$	p	m	$\overline{n}_s$	$\overline{n}_t$	p	m
Hairball	RSBVH	4544.30s	191.41%	**386.54**	**8.31**	**74.61**	25.84	**57.29**	**115.38**	**72.09**	29.81	**113.51**	**116.32**
	RSSBVH	9.51h	205.43%	392.11	**8.31**	75.42	**25.78**	58.01	116.29	73.32	**28.79**	115.06	116.77
	SBVH	136.44s	40.24%	428.01	28.27	80.55	44.51	80.03	141.16	75.80	46.58	139.45	137.54
	RBVH	1685.17s	-	453.97	36.72	81.95	54.95	90.75	153.29	78.30	56.16	152.77	150.12
	BBVH	23.80s	-	466.36	37.82	85.58	56.08	93.31	158.78	80.17	56.76	157.01	152.96
	RMBVH	377.09s	-	494.60	44.18	95.76	56.02	102.51	170.93	90.46	57.05	169.46	165.60
	GBVH	2.38h	-	470.67	49.83	130.61	75.36	104.32	232.10	95.47	58.79	166.88	173.36
Powerplant	RSSBVH	1944.74s	149.42%	**30.65**	**2.21**	**29.83**	**3.78**	**17.28**	**39.58**	**31.23**	**5.33**	**22.57**	**42.80**
	RSBVH	251.61s	148.99%	32.39	2.29	31.47	3.98	18.24	41.74	32.36	5.44	23.84	44.26
	SBVH	11.88s	84.53%	33.15	3.16	31.01	4.45	19.05	41.67	31.79	6.19	24.64	44.34
	RBVH	96.65s	-	41.06	12.97	39.74	13.14	27.85	60.83	39.78	12.55	33.08	60.28
	RGBVH	21.09h	-	40.19	14.24	40.90	13.94	27.99	63.03	40.90	12.91	32.82	61.99
	BBVH	2.01s	-	43.93	13.16	41.58	13.01	29.46	62.90	41.98	12.79	35.19	63.17
	GBVH	88.82s	-	47.02	22.01	52.27	14.53	35.26	77.25	51.42	13.16	39.92	74.87
	RMBVH	21.88s	-	74.39	35.66	84.96	14.11	56.18	116.06	80.72	13.53	63.39	110.39
	RMGBVH	2.61h	-	74.67	43.23	99.42	17.28	59.89	136.58	90.23	14.11	65.74	122.38
Rungholt	RSSBVH	5.92h	3.13%	105.51	**2.63**	**35.38**	**2.10**	105.51	**44.56**	**37.32**	3.28	105.51	**48.06**
	GBVH	4.12h	-	109.66	4.20	36.26	2.23	109.66	45.74	37.54	3.35	109.66	48.41
	BBVH	54.26s	-	109.86	3.43	36.13	2.20	109.86	45.56	37.65	3.33	109.86	48.51
	SBVH	160.02s	0.0%	109.86	3.43	36.13	2.20	109.86	45.56	37.65	3.33	109.86	48.51
	RSBVH	4797.41s	3.09%	105.56	2.64	36.44	2.11	105.56	45.84	37.81	3.28	105.56	48.64
	RBVH	2432.67s	-	**105.49**	2.74	36.57	2.11	**105.49**	45.99	37.84	**3.27**	**105.49**	48.68
	RMBVH	492.98s	-	113.41	3.50	37.26	2.14	113.41	46.86	38.76	3.28	113.41	49.79
San Miguel	RSBVH	4.06h	21.62%	16.56	**1.63**	**60.83**	6.59	10.15	79.59	**57.50**	**9.18**	12.31	**78.18**
	RSSBVH	13.99h	22.66%	**15.83**	2.12	60.84	**6.47**	**9.95**	**79.48**	59.24	9.22	**11.93**	80.31
	SBVH	211.53s	13.67%	19.63	3.12	61.75	6.79	12.55	80.89	60.21	9.94	14.94	82.19
	RBVH	2.5h	-	17.25	7.52	71.80	9.63	13.08	95.79	68.43	12.90	14.48	95.01
	BBVH	130.53s	-	20.28	10.21	83.08	9.76	15.97	109.46	76.48	13.17	17.42	104.94
	GBVH	28.47h	-	24.56	18.06	119.92	12.35	21.77	156.26	103.77	13.53	22.71	138.06
	RMBVH	2042.66s	-	27.59	20.50	142.36	10.84	24.55	181.67	120.56	13.81	25.58	158.48
Sibenik	RSSBVH	431.10s	80.91%	**44.56**	**1.18**	**32.90**	3.53	**17.62**	**43.01**	**34.04**	**5.83**	**27.96**	**46.68**
	RSBVH	58.39s	78.92%	46.55	1.25	33.21	**3.52**	18.42	43.38	34.84	**5.83**	29.22	47.64
	SBVH	2.22s	33.76%	47.48	1.58	35.94	4.33	18.97	47.45	34.60	6.62	29.92	48.14
	RBVH	28.97s	-	48.75	4.16	37.40	6.32	21.06	51.21	37.01	7.65	31.69	52.07
	RGBVH	2.1h	-	49.06	4.70	36.54	6.53	21.51	50.38	37.14	7.77	32.09	52.33
	BBVH	0.52s	-	53.64	5.00	42.42	6.35	23.43	57.25	39.38	7.56	35.03	54.81
	GBVH	10.27s	-	54.31	6.27	46.70	6.97	24.47	63.01	43.06	8.14	35.93	59.81
	RMBVH	6.62s	-	68.53	9.53	61.30	6.10	31.89	79.67	52.92	7.52	45.96	71.03
	RMGBVH	1027.88s	-	69.85	14.74	65.46	7.32	35.63	85.87	60.17	8.32	48.77	80.53
Soda	RSSBVH	2.24h	25.65%	**58.67**	**1.90**	31.61	**3.34**	**33.07**	41.26	30.43	**5.06**	**38.57**	**41.57**
	RSBVH	2396.55s	25.66%	61.12	2.35	31.33	3.54	34.61	**41.13**	**30.42**	5.24	40.31	41.75
	SBVH	45.64s	13.96%	66.56	2.70	32.79	3.76	37.76	43.12	30.69	5.61	43.94	42.43
	RBVH	1451.70s	-	66.15	10.17	**31.15**	5.23	40.90	42.61	34.31	7.78	46.32	48.96
	BBVH	21.35s	-	77.93	13.70	35.95	5.68	48.96	48.82	38.79	8.50	55.18	55.05
	GBVH	2.07h	-	85.44	26.85	53.42	6.93	59.01	71.03	49.03	8.79	64.69	67.63
	RMBVH	252.34s	-	110.30	31.90	64.83	6.61	74.93	84.41	62.15	8.49	82.53	83.08
Sponza	RSBVH	257.75s	57.56%	64.97	4.60	**39.83**	3.93	19.81	**51.73**	41.72	**6.02**	27.58	**56.08**
	SBVH	7.41s	28.79%	70.20	3.02	45.11	**3.77**	19.94	57.90	41.94	6.28	28.59	56.61
	RSSBVH	2052.19s	61.37%	**60.85**	**1.22**	42.90	4.33	**16.24**	55.81	**41.56**	6.86	**23.92**	56.73
	RGBVH	35.97h	-	69.17	7.45	48.60	7.19	23.00	65.51	46.62	8.28	30.94	64.22
	RBVH	133.39s	-	70.86	7.85	47.03	7.11	23.72	63.54	48.44	9.07	31.83	67.20
	GBVH	96.92s	-	78.14	9.87	66.42	8.48	27.07	88.17	57.70	8.68	35.86	77.92
	RMBVH	25.91s	-	85.60	12.00	61.66	5.15	30.54	79.14	61.99	7.12	40.02	81.51
	BBVH	2.06s	-	83.14	12.95	63.60	5.75	30.63	82.07	62.33	8.41	39.66	83.20
	RMGBVH	4.46h	-	90.75	16.87	86.60	9.41	35.48	113.32	73.47	9.35	44.99	97.52

Table 6: Results for the remaining scenes of Table 2. See Table 5 for a description of each measurement.