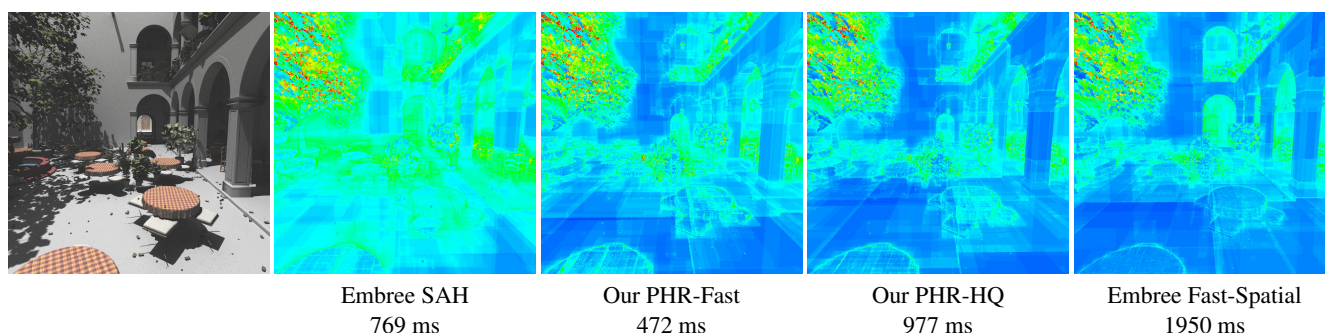# Parallel BVH Construction using Progressive Hierarchical Refinement

J. Hendrich and D. Meister and J. Bittner

Czech Technical University in Prague, Faculty of Electrical Engineering



|  |  |  |  |
|---|---|---|---|
| Embree SAH | Our PHR-Fast | Our PHR-HQ | Embree Fast-Spatial |
| 769 ms | 472 ms | 977 ms | 1950 ms |

**Figure 1:** *The San Miguel scene rendered in the Embree path tracer [WWB\*14]. Visualization of the number of traversal steps for primary rays using BVHs from different builders (the red color corresponds to 100 traversal steps per ray). From left-to-right: Embree SAH, our PHR-Fast, our PHR-HQ, and Embree Fast-Spatial. The bottom row shows the build times. Our PHR-Fast method provides 1.6× lower build time than Embree SAH, while the PHR-HQ method has 2× lower build time than Embree Fast-Spatial. In both comparisons, the builders provide equivalent ray tracing performance.*

## Abstract

*We propose a novel algorithm for construction of bounding volume hierarchies (BVHs) for multi-core CPU architectures. The algorithm constructs the BVH by a divisive top-down approach using a progressively refined cut of an existing auxiliary BVH. We propose a new strategy for refining the cut that significantly reduces the workload of individual steps of BVH construction. Additionally, we propose a new method for integrating spatial splits into the BVH construction algorithm. The auxiliary BVH is constructed using a very fast method such as LBVH based on Morton codes. We show that the method provides a very good trade-off between the build time and ray tracing performance. We evaluated the method within the Embree ray tracing framework and show that it compares favorably with the Embree BVH builders regarding build time while maintaining comparable ray tracing speed.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Raytracing—I.3.5 [Computer Graphics]: Object Hierarchies—I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—

## 1. Introduction

Bounding volume hierarchies (BVHs) are one of the most efficient spatial data structures for organizing scene primitives. Most contemporary ray tracing implementations use BVHs for accelerating ray scene intersections. This makes the task of constructing high-quality BVH, i.e. a BVH leading to a low number of intersection evaluations thus high ray tracing speed, very important. This problem has been studied very intensively and currently a number of very efficient solutions exist that lead to high-quality BVHs [SFD09, Wal12a, KA13, GHFB13, GBDAM15]. Some techniques are even applicable for fully dynamic scenes as they can construct the BVH in real-time for moderately complex scenes. As a general rule of thumb, a higher quality BVH means longer build time.

The BVH can be constructed in $O(n \log n)$ time even for the relatively expensive full sweep method based on surface area heuristic. Recently several methods were proposed which lead to BVHs with a lower cost (i.e. expected number of intersections) than that of full sweep SAH [Ken08, BHH13, KA13, GHFB13, GD16]. While the SAH cost model correlates with the actual ray tracing performance,

Aila et al. [AKL13] identified the supremacy of top-down builders regarding the correlation of the cost and ray tracing performance. In other words, being able to quickly construct a BVH using a top-down algorithm, with its quality comparable to the state-of-the-art BVH optimization methods, boosts the overall rendering performance of ray tracing.

In this paper, we revisit the idea of Hunt et al. [HMF07], who proposed the build-from-hierarchy method using the scene graph structure to accelerate kD-tree construction. We apply the build-from-hierarchy in a different context: we build a BVH instead of a kD-tree and use triangle soup instead of scene graph as the input. Both points bring our method closer towards the current state-of-the-art ray tracing frameworks and in turn to the actual usage in practice. It has not been clear whether build-from-hierarchy can compete with other methods when a scene graph (i.e. the input hierarchy) is not available. We show that the method is very competitive even with simple non-SIMD implementation and consequently we believe that our paper will renew the interest in the build-from-hierarchy techniques in general.

Starting from a triangle soup, we first build an auxiliary BVH using a very fast BVH builder. This gives us quick access to the scene structure, i.e. a coarse description of the spatial distribution of the primitives. At the core of the method, we use a new adaptive method for accessing the auxiliary BVH during the construction of the final one. To further improve the quality of the constructed BVH for ray tracing, we propose a fast way of integrating spatial splits in the BVH construction process. We show that this approach is competitive with the fastest available BVH builders for multi-core architectures, such as AAC [GHFB13], Bonsai [GBDAM15], or Fast-Binning [Wal12a]. An excerpt of a comparison with existing methods implemented in the Embree ray tracing framework [WWB*14] is shown in Figure 1.

## 2. Related Work

Bounding volume hierarchies have a long tradition in rendering and collision detection. Kay and Kajiya [KK86] designed one of the first BVH construction algorithms using spatial median splits. Goldsmith and Salmon [GS87] proposed the measure currently known as the *surface area heuristic* (SAH), which predicts the efficiency of the hierarchy during the BVH construction. The vast majority of currently used methods for BVH construction use a top-down approach based on SAH. A particularly popular method is the fast approximate SAH evaluation using binning proposed by Havran et al. [HHS06] and Wald et al. [Wal07, Wal12b].

A number of parallel methods for BVH construction have been proposed for both GPUs and multi-core CPUs. Lauterbach et al. [LGS*09] proposed a GPU algorithm which uses the Morton code based primitive sorting. Hou et al. [HSZ*11] proposed partial breadth-first search construction order to control the GPU memory consumption. Pantaleoni and Luebke [PL10] proposed the HLBVH algorithm that combines Morton sorting with SAH based tree construction. Garanzha et al. [GPM11] improved on this method by using SAH for the top part of the constructed BVH.

Karras [Kar12] and Apetrei [Ape14] proposed GPU-based methods for efficient parallel LBVH construction. These techniques

achieve impressive performance but construct a BVH of a lower quality than the SAH based builders.

Significant interest has been devoted to methods which construct high-quality BVHs albeit at increased computational time compared to the fastest builders. Walter et al. [WBKP08] proposed to use bottom-up agglomerative clustering. Gu et al. [GHFB13] used parallel approximative agglomerative clustering for accelerating the bottom-up BVH construction. Ganestam et al. [GBDAM15] proposed the Bonsai method, which uses a two-level BVH construction scheme similar to HLBVH and subsequent tree pruning. In their follow-up work, they show how to integrate spatial splits [SFD09] into the Bonsai algorithm [GD16].

Hunt et al. [HMF07] proposed to construct a kD-tree from the scene graph hierarchy. Kensler [Ken08], Bittner et al. [BHH13], and Karras and Aila [KA13] optimize the BVH by performing topological modifications of the existing tree. These approaches allow decreasing the expected cost of a BVH beyond the cost achieved by the traditional top-down approach.

Due to the widespread availability of SIMD instructions on today's CPU architectures, it is beneficial to construct multi-BVHs, i.e. hierarchies with a higher branching factor. Wald et al. [WBB08] and Dammertz et al. [DHK08] designed methods for constructing and traversing the multi-BVH, which are particularly important in combination with modern ray tracing frameworks such as Embree [WWB*14]. Further improvements of the multi-BVH can be achieved by adapting it to a particular ray distribution as suggested by Gu et al. [GHB15].

The paper is further structured as follows: Section 3 presents an overview of the proposed method, Section 4 presents the details of the proposed method, Section 5 gives the results and their discussion, and Section 6 concludes the paper.

## 3. Efficient Top-Down BVH Construction

### 3.1. Handling Large Data

Constructing an efficient BVH is an optimization problem equivalent to hierarchical clustering. The core issue for efficient BVH construction is the large amount of data that has to be organized at the beginning of the computation. A number of successful methods gain speed and efficiency by simplifying the initial phase of the computation through quick partitioning of the data into subsets that are later processed using a more sophisticated algorithm. This is the case of the HLBVH [GPM11] that uses Morton codes to cluster the data, and thus to reduce the search space for partitioning. Similarly, the Bonsai algorithm [GBDAM15] uses simple spatial median partitioning to establish independent data clusters that are processed using more involved partitioning schemes. The AAC algorithm [GHFB13] uses a simple top-down subdivision scheme based on Morton codes, which allows for efficient bottom-up clustering phase.

Another example of handling the initial scene complexity is the build-from-hierarchy method proposed by Hunt et al. [HMF07], which was implemented in their innovative Razor ray tracing system. The method uses explicit knowledge of scene hierarchy to reduce the complexity of kD-tree construction significantly.

We follow the path visible in the above-discussed techniques with the aim of providing a highly scalable and efficient BVH construction algorithm. Similarly to HLBVH, Bonsai, or AAC, we use Morton code based spatial sorting to establish the initial scene partitioning. Following Hunt et al. [HMF07], we use the hierarchy resulting from the initial partitioning to construct the final BVH. We make substantial modifications to this phase of the algorithm that aim to better distribute the workload among different levels of the hierarchy and to efficiently perform spatial splits during the BVH construction. Despite the increased quality of the final BVH, we keep the second phase of the algorithm at $O(n)$ complexity by bounding the working set when creating each interior node, similarly to Hunt et al. [HMF07].

## 3.2. Algorithm Overview

Our algorithm starts by constructing an auxiliary BVH that provides a scalable description of the scene structure. We use the LBVH algorithm [LGS*09] implemented on multi-core CPU architecture. The auxiliary BVH serves as a valuable hierarchical representation of the structure and distribution of objects. By establishing and progressively refining the cuts of the auxiliary BVH, we can efficiently construct the final BVH by a thorough analysis and partitioning of these cuts (see Figure 2).
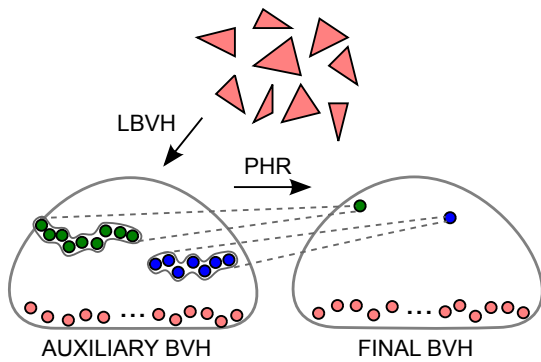


**Figure 2:** *Overview of the method. Fast LBVH builder constructs the auxiliary BVH, which is used to build the final high-quality BVH using progressive hierarchical refinement (PHR).*

The construction of the final BVH starts by finding the initial cut of the auxiliary BVH. This cut is formed as the smallest set of nodes which spans the BVH horizontally and the nodes have their surface smaller than a particular threshold. At each step of the algorithm, the nodes on this cut are examined and we search for the best partitioning of these nodes into two sets. We use full sweep SAH in all three axes to find the partitioning. As the size of the cut is small compared to the scene size, this search is very fast. Then the cut is partitioned using the best split found and two new cuts are formed. These cuts are refined according to a new threshold corresponding to the reached subdivision depth. Should a node on the cut be refined, we just replace the node in the cut with its children. The algorithm then continues by applying the method recursively in the new subtrees with the corresponding refined cuts. The illustration of the main steps performed on the auxiliary BVH cut, when building a node in the final BVH, is shown in Figure 3.

## 4. Progressive Hierarchical Refinement

This section describes the proposed method in detail by presenting and discussing its individual components. For now, let us assume that we have an auxiliary BVH at our disposal. The details of constructing the auxiliary BVH will be given at the end of this section.

## 4.1. Finding the Initial Cut

Our method starts with identifying the nodes of the auxiliary BVH that will form a cut of the hierarchy. This cut contains nodes that are just below a given threshold on their surface area. We will discuss how to determine this threshold in Section 4.3. The threshold is set in a way that it provides an initial cut with size below a given hard bound (e.g. 2048 nodes); which is, in general, several orders of magnitude lower than the total number of scene primitives.

The algorithm for finding the cut uses a priority queue and a simple test that checks whether the visited node has a surface area larger than a threshold. If this is the case, its children are put into the priority queue. If this is not the case or the node is a leaf it is appended to the cut. When there are no more nodes to visit or the cut together with the unprocessed nodes have reached the maximum cut size, we terminate the cut search.

## 4.2. Splitting the Cut

The core of the method is splitting the current BVH cut into two disjoint sets. As the cut is small, we can use a relatively expensive evaluation of the best split without significant performance penalty. Thus, we use full-sweep SAH in all three axes to subdivide the current cut. We sort the cut along all three axes based on node centroids. Then for all axes, we perform a sweep that computes a cost of the node subsets on the left and right of the sweep plane. For a given position $i$ of the sweep plane, the cost is given as

$$C(i) = S_L(i)n_L(i) + S_R(i)n_R(i), \tag{1}$$

where $S_L(i)$ and $S_R(i)$ are the surface areas of the bounding boxes of left and right subsets and $n_L(i)$ and $n_R(i)$ are the numbers of nodes on the left and on the right from the sweep plane, respectively. Then we select the axis and the position of the sweep plane that yield the minimum cost.

There is a minor difference from the traditional SAH cost evaluation: we use the numbers of nodes $n_L(i)$ and $n_R(i)$ instead of the numbers of triangles in the subtree [HMF07]. We discovered that tracking the numbers of nodes provides slightly better results for most tested scenes (in a range of a few percent). We expect that this is because the numbers of nodes better reflect the complexity of hierarchically organizing the corresponding scene part than the raw triangle numbers. Using the numbers of triangles yields a cost corresponding to the node being a leaf, which is obviously not the case for nodes higher in the hierarchy. The node counts, on the contrary, are somewhere between the actual triangle counts and final (but yet unknown) costs of the subtrees being constructed. Moreover, the numbers of nodes $n_R(i)$ and $n_L(i)$ for current $i$ are readily available values, whereas the triangle counts rely on storing them as additional information for each node of the auxiliary BVH.
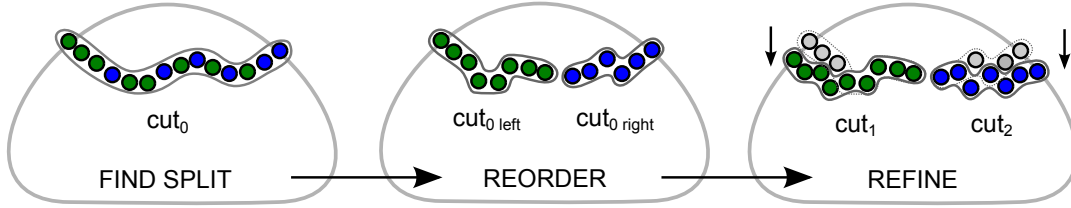
**Figure 3:** *Illustration of the hierarchical cut refinement. (left) When computing the split, we determine the optimal subdivision of the current cut ($cut_0$) based on full sweep SAH. That results in two sets of nodes on the current cut which correspond to the nodes that will be contained in the left and right subtree of the subdivided cut (shown in green and blue). (center) The nodes on the cut are reordered and split into two disjoint cuts. (right) The new cuts ($cut_1$ and $cut_2$) are formed by hierarchical refinement based on their surface area.*

### 4.3. Refining the Cut

By subdividing the BVH cut into two parts, the number of nodes in each of the two new cuts is reduced. Repeating this step several times would lead to cut size of 1, and thus we would lose all the information about the scene structure that the cut can provide. Hence we refine the cut to maintain its properties. Hunt et al. [HMF07] suggested to keep a constant size of the cut that is specified by the user (they used the cut size 500 for the results in the paper). While this is a valid approach that maintains the linear complexity of the algorithm [HMF07], this technique becomes expensive especially for medium to high BVH depths, where the algorithm has to cope with a large number of cuts with their sizes comparable to the number of primitives in their subtree.

Instead, we propose an adaptive way of refining the cut that aims to progressively reduce the cut size, and thus to better balance the computational effort among different levels of the BVH. This approach is similar to adapting the number of clusters handled by the AAC algorithm [GHFB13] in their bottom-up BVH construction.

Our method is based on thresholding the surface area of nodes forming the cut while taking into account the current depth. The cut is refined as follows: for each node forming the cut, we compare its surface area with the threshold. If the surface area of the node is greater than the threshold, the node is replaced by its children. Otherwise, the node is kept in the cut. When refining the cut, we intentionally descend just one level in the tree. This simplifies the implementation as no stack is needed to refine the cut.

We propose to use an adaptive threshold that is based on the depth of the currently computed node in the BVH. The threshold is computed as

$$t_d = \frac{S}{2^{\alpha d + \delta}}, \tag{2}$$

where $S$ is the surface area of the scene bounding box, $d$ is the current depth, $\alpha$ and $\delta$ are parameters of the method. The $\delta$ parameter determines the size of the initial cut for $d = 0$. The $\alpha$ parameter describes how quickly will the cut size shrink with increasing depth. The setting of these parameters depends on the desired trade-off between the build time and the trace speed. We used two settings of these parameters: $\alpha = 0.5$ and $\delta = 6$ for fast builds (PHR-Fast) and $\alpha = 0.55$ and $\delta = 9$ for high-quality builds (PHR-HQ).

Note that the formula is inspired by regular subdivision of the scene into non-overlapping voxels. In this case, the size of the voxel

in depth $d$ is given as $S_d \approx \frac{S}{2^{\frac{2}{3}d}}$. The bounding boxes of the input hierarchy do not follow this ideal subdivision case, but keeping the working set of nodes in the cut with areas staying close to this function proved beneficial for the performance vs. quality trade-off of the algorithm.

An example of the lengths of the cut obtained using the proposed adaptive threshold function is given in Figure 4. As we show and discuss in Section 5, the adaptive threshold leads to consistently better results than a predefined cut size originally proposed by Hunt et al. [HMF07].
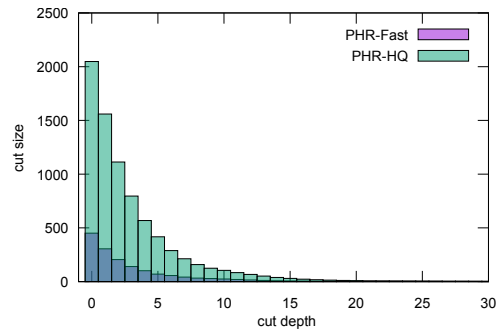


**Figure 4:** *The average cut size at different BVH depths for the San Miguel scene. PHR-HQ (green) works on substantially larger cuts than PHR-Fast (purple). The adaptive threshold function implies larger cut size at the top of the tree, which has a crucial impact on the overall tree quality.*

### 4.4. Spatial Splits

Spatial splits proposed by Stich et al. [SFD09] can improve the BVH quality significantly. However, their evaluation is relatively expensive. We make use of the availability of the limited size BVH cut and propose a novel technique for spatial splits evaluation.

When splitting a cut, we also evaluate its result when applying spatial splits on the cut nodes. This contrasts with the traditional spatial splits evaluation which is performed directly on geometric primitives (triangles).

The actual spatial split evaluation is done using an algorithm similar to the kD-tree construction. We sort the boundaries of the node bounding boxes and perform plane sweep while evaluating

the split cost. Note, however, that in this case, the classification of the node into the left and right subtree is not based on the centroid of the node but on the two extents of the node bounding box in the given axis. A node is classified as lying in both left and right subtrees if the sweep plane intersects its bounding box. For each cost evaluation, we clip the left and right bounding boxes by the sweep plane which potentially reduces the respective areas ($S_L$ and $S_R$). Thus, we obtain

$$C_S(i) = S_L^c(i)n_L^*(i) + S_R^c(i)n_R^*(i), \qquad (3)$$

where $S_L^c(i)$ and $S_R^c(i)$ are surface areas of clipped bounding volumes of nodes intersecting the left and right volumes defined by the sweep plane, $n_L^*(i)$ and $n_R^*(i)$ are the numbers of nodes intersecting these left and right volumes.

If the spatial split cost $C_S$ is lower than the cost $C$, we perform a spatial split. We actually do not subdivide any bounding boxes nor primitives themselves. Instead, we just clip the two bounding boxes for the left and right subsets using the split plane. The clipped bounding box is always passed over with the current cut to apply clipping also in the deeper levels of the tree. More precisely, when the cost function is evaluated deeper in the tree, the bounding boxes of the nodes being processed are always clipped by the clipping bounding box for the given cut that was passed from the parent nodes.

If spatial splits are used, it can happen that some nodes on the refined cut do not intersect the clipped bounding box passed over with the current cut. These nodes are simply skipped, i.e. they are not placed into the refined cut. An illustration of a spatial split applied on the current cut is shown in Figure 5.

To avoid performing spatial splits at lower parts of the tree where their benefit is usually low, we use a simple rule proposed by Stich et al. [SFD09]. Spatial splits are performed only when the ratio of the surface area of the currently constructed node and the bounding box of the scene is above a given threshold (e.g. 1e-3).

The proposed spatial splits method is very simple and requires minimal modifications of the other parts of the code. The results show that this method, although being very fast, improves on the trace performance up to 20%.

### 4.5. Multi-BVH

The trace performance of the BVH can be improved by increasing the branching factor of interior nodes, which results in a shallower BVH [WBB08, DHK08]. This is particularly important for SIMD optimized ray tracers. We construct the multi-BVH by postponing the formation of interior nodes in the final BVH until $n$ children are available or no child interior node exists. If the current number of child nodes is lower than $n$, the algorithm processes the largest available interior child node first [WBB08].

### 4.6. Parallelization

The parallelization of the method is relatively straightforward. We use a shared work queue which contains entries representing unconstructed parts of the tree. Each entry contains an index of a node in the new BVH and an array representing the corresponding cut in
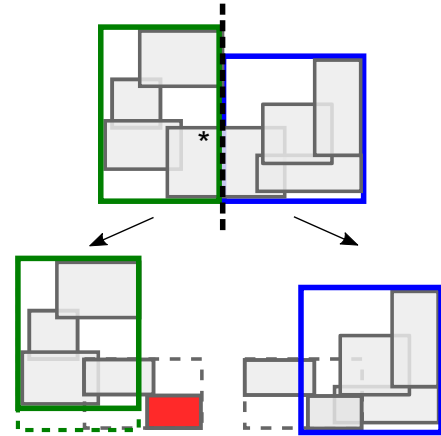


**Figure 5:** *Illustration of a spatial split. A spatial split is selected for nodes forming the current BVH cut. Two subsets of nodes are formed for the left and right subtree, and the bounding volumes of these subsets are clipped by the splitting plane. The node denoted by the * symbol is split by the split plane and therefore it is placed in both subsets. Left and right subsets are refined to form new cuts. Particularly, the node * is replaced by its children (see the images in the bottom). Note that for the left subset, one of the child nodes (shown in red) does not intersect the currently clipped bounding volume, and thus it is discarded from the cut in this branch. The clipped bounding box can thus be shrunk to reflect this.*

the auxiliary BVH. A thread pops an entry from this queue, finds a subdivision of the cut and if the node does not become a leaf, it stores the right child in the shared queue so that other threads can fetch this entry and process it. The left branch is processed immediately by the same thread unless it is a leaf node. Note that to prevent large synchronization overhead, we only store the nodes in the shared work queue up to a given depth (e.g. 12) or when we find out that some threads are idle (using a counter of active threads).

### 4.7. Constructing the Auxiliary BVH

We use parallel sorting based on Morton codes and subsequent binary search to identify the nodes of the auxiliary BVH. We first compute Morton codes for all triangles. For that, each thread is assigned a continuous span of $\lceil n/t \rceil$ triangles, where $n$ is the number of scene triangles and $t$ is the number of threads. For parallel sorting, we use approximate parallel bucket-sort. In each thread, we insert the primitives into $k$ buckets. We used $k = 2^{12}$ that corresponds to sorting according to the highest 12 bits of the Morton code. Then the buckets are merged in parallel by evaluating $\lceil k/t \rceil$ buckets per thread. The auxiliary BVH is constructed using a shared work queue. Each thread fetches a node from the queue and finds the intervals corresponding to its children by identifying an entry with a change in the highest bit of the Morton code within the current interval [LGS*09]. Finally, bounding boxes are updated by a parallel recursive procedure using synchronized access to node's data. We use an atomic counter for each node: the thread that visits the node as the second node will update node's bounding box using the bounding boxes of its children.

## 5. Results

We implemented the proposed method in C++ using standard language constructs. In the current implementation, we did not exploit SIMD instructions. We performed a series of tests comparing the build times, BVH costs, and the ray tracing performance of our method and several reference methods on nine test scenes. As a reference, we used our implementations of the method of Hunt et al. [HMF07] (denoted Hunt-50, Hunt-500) and the AAC method of Gu et al. [GHFB13] (AAC). We also evaluated the methods available in the latest version of Embree ray tracer, i.e. Embree 2.11 (Embree SAH, Embree Fast-Spatial, Embree Morton). The Hunt-50 and Hunt-500 methods used a fixed length cut of the length of 50 and 500, respectively. The cut of a given length is established by partial sort and subsequent refinement of the largest nodes, which proved more efficient than using a priority queue. We did not use the fast-scan and lazy-build also proposed by Hunt et al. [HMF07] as these are specific to kD-tree construction and the Razor system. For the AAC method, we used parameters corresponding to AAC-Fast settings [GHFB13]. For all methods, we used a 4-ary multi-BVH [WBB08, DHK08].

The trace times were evaluated using a simple path tracer provided as a tutorial in Embree. The times are computed as average times for three different representative views of each scene using $1024 \times 1024$ resolution and 1 sample per pixel. The measurements were performed using 16 working threads on a PC equipped with $2 \times$ Intel Xeon E5-2620. The measured results are summarized in Table 1. Below we discuss the results from different points of view.

### 5.1. Construction Speed

The table shows that the lowest build times among the reference methods are achieved by the Embree Morton method, usually followed by Hunt-50, AAC, Embree SAH, Hunt-500, and Embree Fast-Spatial. Our PHR-Fast method provides build times usually between Embree Morton and Hunt-50; thus, it is the second fastest builder tested. The PHR-HQ method has build times mostly between Embree SAH and Hunt-500. The PHR-HQ build times are about twice lower than the build times of Embree Fast-Spatial.

On scenes with a simple structure (e.g. Buddha), the PHR-Fast and PHR-HQ methods perform comparably to Hunt-50 and Hunt-500, respectively. On larger scenes, PHR-Fast and PHR-HQ achieve about 20% lower build times than Hunt-50 and Hunt-500 while leading to slightly higher quality BVHs.

### 5.2. BVH Quality and Ray Tracing Performance

The overall highest BVH quality in terms of trace times was achieved by the Embree Fast-Spatial method. The PHR-HQ method closely follows in half of the test scenes. In two tested scenes (Hairball and Soda Hall), PHR-HQ actually provided marginally better trace performance. In four test scenes, PHR-HQ provides about 10%-20% lower trace performance than Embree Fast-Spatial.

The PHR-Fast method provides trace performance usually between Embree Morton and Embree SAH while being closer to Embree SAH in terms of trace speed. The build times on the other hand

are closer to Embree Morton, which makes the PHR-Fast method a good candidate for interactive applications.

We provide a graphical comparison of build time vs. trace time for all tested methods in Figure 6. The figure also sketches the *Pareto front* known from multi-objective optimization [RW05] that indicates the methods which will perform superior to the others for some combination of build time vs. trace time, i.e. the number of rays traced. According to the measurements, the Pareto front is formed by Embree Morton, PHR-Fast, Embree SAH, PHR-HQ, and Embree Fast-Spatial methods.
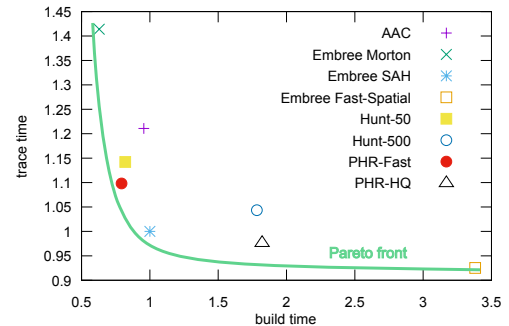


**Figure 6:** *The relative performance of different methods with respect to Embree SAH as the reference method and averaged over all test scenes.*

### 5.3. Influence of Spatial Splits

Spatial splits provide trace speedup particularly for large architectural scenes with primitives of different sizes. The influence of spatial splits can be seen by relating to the Hunt-50 and Hunt-500 methods, which do not employ spatial splits. The trace speedup due to our fast spatial splits method is in the range of 2% and 20%. Spatial splits had the largest positive impact in the San Miguel scene.

Our node level spatial splits could also be combined with triangle presplitting [KA13]. We have not yet evaluated this possibility, but we consider it an interesting topic for future work.

### 5.4. Parameters

Prior to determining the settings for PHR-Fast and PHR-HQ, we conducted a series of measurements to evaluate the dependence of the method on the $\alpha$ and $\delta$ parameters. We tested values of $\alpha$ slightly below $\frac{2}{3}$ (an explanation of this threshold is given at the end of Section 4.3). We determined the range of measured $\delta$ values by balancing the amount of work done at each subdivision step with the potential of finding a precise enough split while focusing on the splits near the root. In particular, we tested all the combinations of $\alpha$ and $\delta$ from the sets {0.45, 0.5, 0.55, 0.65} and {5, 6, 7, 8, 9, 10}, respectively. From these measurements, we selected the two characteristic settings (PHR-Fast, PHR-HQ) that roughly defined the Pareto front for all scenes. In a selected subset of scenes, the representative parameter settings can be slightly different, and thus better performance/quality ratios could be achieved. For example, in San Miguel and Power Plant scenes, the optimal values are $\alpha = 0.5, \delta = 7$ for the high quality scenario (saving about 35%

build time while actually decreasing the trace time by about 3%). In Conference and Hairball scenes, it is $\alpha = 0.5$, $\delta = 5$ for the fast build scenario (saving 9% and 33% build time, respectively, while keeping the same trace time).

### 5.5. Using PHR with Other Builders

We have tested the proposed method with other BVH builders for constructing the initial BVH (AAC and Binning-SAH); the results were generally worse concerning Pareto optimality. For example, using AAC allowed us to build a BVH with a slightly lower cost (a few percent) for the same parameter settings of PHR; using different PHR parameters, a similar quality BVH could be constructed faster.

### 5.6. Discussion and Limitations

The results indicate that the method is configurable in a large range of build time vs. BVH quality trade-off. On one side, this is a positive feature; on the other hand, this makes it difficult to find optimal parameters ($\alpha$ and $\delta$) for the PHR method. We used two representative settings, but we observed that in a number of scenes setting different values to these parameters would provide better build times or trace performance while not significantly altering the other value.

The PHR-Fast method seems a good fit for interactive applications. In fact, we could avoid constructing the auxiliary BVH from scratch and reuse it over multiple frames by simple refitting to further reduce the build time of PHR-Fast. This strategy was already suggested by Hunt et al. [HMF07], but its verification in practice remains an open topic.

The current implementation of the method uses a moderately optimized C++ code, but we did not yet exploit SIMD constructs. We expect that by exploiting SIMD we could further lower the build times while keeping the same BVH quality. On a similar note, the sweep SAH algorithm used at the core of our method could be replaced by SIMD optimized binning SAH.

### 6. Conclusion and Future Work

We proposed a novel scalable algorithm for BVH construction on multi-core CPU architectures. The algorithm employs a two-phase process: first it constructs an auxiliary BVH using the LBVH algorithm, followed by constructing the final BVH using progressively refined cuts of the auxiliary BVH. The progressive refinement of the cut size is driven by adapting surface area thresholds based on the current depth of the constructed node. We provided a simple way of integrating spatial splits in the BVH construction process.

The results show that the method yields superior build performance compared to the high-quality builders implemented in the Embree framework while closely matching their ray tracing performance. In comparison with the strategy of Hunt et al. [HMF07] adapted to BVH construction, our method brings about 20% build time improvement while also providing a few percent improvement in the trace performance.

There are a number of avenues for future work. We see a great potential in the integration of static and dynamic content by using the progressive hierarchical refinement to merge a high-quality static hierarchy and dyn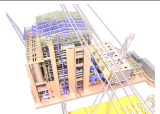amically constructed auxiliary BVH. This technique could have immediate applications in the video game technology. The proposed method scales in a large build time vs. quality range. Finding optimal parameters for a given scene and target frame rate in interactive applications remains an open problem.

### Acknowledgements

### References

[AKL13] AILA T., KARRAS T., LAINE S.: On Quality Metrics of Bounding Volume Hierarchies. In *Proceedings of High Performance Graphics* (2013), pp. 101–108. 2

[Ape14] APETREI C.: Fast and Simple Agglomerative LBVH Construction. In *Computer Graphics and Visual Computing (CGVC)* (2014). 2

[BHH13] BITTNER J., HAPALA M., HAVRAN V.: Fast Insertion-Based Optimization of Bounding Volume Hierarchies. *Computer Graphics Forum 32*, 1 (2013), 85–100. 1, 2

[DHK08] DAMMERTZ H., HANIKA J., KELLER A.: Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. *Computer Graphics Forum 27*, 9 (2008), 1225–1233. 2, 5, 6

[GBDAM15] GANESTAM P., BARRINGER R., DOGGETT M., AKENINE-MÖLLER T.: Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees. *Journal of Computer Graphics Techniques (JCGT) 4*, 3 (2015), 23–42. 1, 2

[GD16] GANESTAM P., DOGGETT M.: SAH Guided Spatial Split Partitioning for Fast BVH Construction. *Computer Graphics Forum 35*, 2 (2016), 285–293. 1, 2

[GHB15] GU Y., HE Y., BLELLOCH G. E.: Ray Specialized Contraction on Bounding Volume Hierarchies. *Computer Graphics Forum 34*, 7 (2015), 309–318. 2

[GHFB13] GU Y., HE Y., FATAHALIAN K., BLELLOCH G. E.: Efficient BVH Construction via Approximate Agglomerative Clustering. In *Proceedings of High Perform. Graphics* (2013), pp. 81–88. 1, 2, 4, 6

[GPM11] GARANZHA K., PANTALEONI J., MCALLISTER D.: Simpler and Faster HLBVH with Work Queues. In *Proceedings of High Performance Graphics* (2011), pp. 59–64. 2

[GS87] GOLDSMITH J., SALMON J.: Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications 7*, 5 (1987), 14–20. 2

[HHS06] HAVRAN V., HERZOG R., SEIDEL H.-P.: On the Fast Construction of Spatial Data Structures for Ray Tracing. In *Proceedings of Symposium on Interactive Ray Tracing* (2006), pp. 71–80. 2

[HMF07] HUNT W., MARK W. R., FUSSELL D.: Fast and Lazy Build of Acceleration Structures from Scene Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing* (2007), pp. 47–54. 2, 3, 4, 6, 7

[HSZ*11] HOU Q., SUN X., ZHOU K., LAUTERBACH C., MANOCHA D.: Memory-Scalable GPU Spatial Hierarchy Construction. *IEEE Trans. on Visualization and Comp. Graphics 17*, 4 (2011), 466–474. 2

[KA13] KARRAS T., AILA T.: Fast Parallel Construction of High-Quality Bounding Volume Hierarchies. In *Proceedings of High Performance Graphics* (2013), pp. 89–100. 1, 2, 6

[Kar12] KARRAS T.: Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees. In *Proceedings of High Performance Graphics* (2012), pp. 33–37. 2

| | build time [ms] | SAH cost [-] | frame time [ms] | trace speed [MRays/s] | build time [ms] | SAH cost [-] | frame time [ms] | trace speed [MRays/s] | build time [ms] | SAH cost [-] | frame time [ms] | trace speed [MRays/s] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| |  Conference #triangles 331k | | | |  Happy Buddha #triangles 1087k | | | |  Soda Hall #triangles 2169k | | | |
| Embree Morton | 18 | 23 | 663 | 27 | 70 | 33 | 60 | 30 | 111 | 39 | 62 | 34 |
| Hunt-50 | 20 | 18 | 549 | 32 | 99 | 28 | 57 | 31 | 143 | 33 | 55 | 39 |
| AAC | 21 | 15 | 499 | 36 | 109 | 31 | 60 | 30 | 177 | 31 | 56 | 38 |
| Embree SAH | 31 | 15 | 508 | 35 | 101 | 26 | 54 | 33 | 184 | 29 | 50 | 43 |
| Hunt-500 | 42 | 15 | 508 | 35 | 202 | 27 | 56 | 32 | 322 | 30 | 51 | 42 |
| Embree Fast-Spatial | 81 | 14 | 521 | 34 | 377 | 26 | 54 | 33 | 586 | 26 | 48 | 45 |
| PHR-Fast | 23 | 15 | 530 | 34 | 88 | 29 | 58 | 31 | 134 | 27 | 49 | 44 |
| PHR-HQ | 39 | 15 | 509 | 35 | 222 | 26 | 54 | 33 | 259 | 25 | 46 | 46 |
| |  Hairball #triangles 2880k | | | |  Crown #triangles 4868k | | | |  Pompeii #triangles 5632k | | | |
| Embree Morton | 165 | 216 | 1027 | 7 | 247 | 12 | 313 | 15 | 289 | 67 | 224 | 14 |
| Hunt-50 | 246 | 171 | 735 | 9 | 331 | 11 | 292 | 16 | 391 | 36 | 156 | 20 |
| AAC | 278 | 190 | 864 | 8 | 380 | 12 | 316 | 15 | 458 | 44 | 177 | 18 |
| Embree SAH | 225 | 164 | 695 | 10 | 411 | 10 | 253 | 18 | 453 | 30 | 138 | 23 |
| Hunt-500 | 518 | 160 | 692 | 10 | 761 | 11 | 269 | 17 | 873 | 31 | 141 | 22 |
| Embree Fast-Spatial | 1607 | 157 | 715 | 10 | 1163 | 10 | 243 | 19 | 1442 | 20 | 108 | 29 |
| PHR-Fast | 341 | 166 | 724 | 10 | 262 | 12 | 304 | 15 | 321 | 35 | 156 | 20 |
| PHR-HQ | 1064 | 159 | 680 | 10 | 589 | 11 | 273 | 17 | 631 | 28 | 133 | 23 |
| |  San Miguel #triangles 7880k | | | |  Vienna #triangles 8637k | | | |  Powerplant #triangles 12759k | | | |
| Embree Morton | 430 | 39 | 1563 | 8 | 506 | 46 | 203 | 17 | 620 | 20 | 227 | 15 |
| Hunt-50 | 512 | 30 | 1244 | 10 | 707 | 25 | 150 | 23 | 674 | 17 | 184 | 18 |
| AAC | 606 | 26 | 1221 | 10 | 713 | 29 | 168 | 20 | 1090 | 16 | 196 | 16 |
| Embree SAH | 769 | 25 | 1060 | 12 | 722 | 19 | 120 | 28 | 1166 | 13 | 146 | 22 |
| Hunt-500 | 1112 | 27 | 1136 | 11 | 1491 | 20 | 133 | 26 | 1560 | 14 | 155 | 21 |
| Embree Fast-Spatial | 1950 | 22 | 879 | 14 | 1687 | 15 | 115 | 29 | 3312 | 10 | 114 | 28 |
| PHR-Fast | 472 | 23 | 1050 | 12 | 534 | 24 | 155 | 22 | 652 | 15 | 165 | 20 |
| PHR-HQ | 977 | 21 | 914 | 14 | 1089 | 17 | 123 | 28 | 1388 | 12 | 140 | 23 |

**Table 1:** *Performance comparison of the tested methods. The reported numbers are averaged over three different viewpoints for each scene. For computing the SAH cost, we used $c_T = 1$ and $c_I = 1$. The leaf cost is computed using the Embree cost model, i.e. triangle quartets are considered a single primitive.*

[Ken08] KENSLER A.: Tree Rotations for Improving Bounding Volume Hierarchies. In *Proceedings of Symposium on Interactive Ray Tracing* (2008), pp. 73–76. 1, 2

[KK86] KAY T. L., KAJIYA J. T.: Ray Tracing Complex Scenes. *Computer Graphics (SIGGRAPH '86 Proceedings) 20*, 4 (1986), 269–278. 2

[LGS*09] LAUTERBACH C., GARLAND M., SENGUPTA S., LUEBKE D., MANOCHA D.: Fast BVH Construction on GPUs. *Comput. Graph. Forum 28*, 2 (2009), 375–384. 2, 3, 5

[PL10] PANTALEONI J., LUEBKE D.: HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry. In *Proceedings of High Performance Graphics* (2010), pp. 87–95. 2

[RW05] RUZIKA S., WIECEK M. M.: Approximation methods in multiobjective programming. *Journal of Optimization Theory and Applications 126*, 3 (2005), 473–501. 6

[SFD09] STICH M., FRIEDRICH H., DIETRICH A.: Spatial splits in bounding volume hierarchies. In *Proceedings of High Performance Graphics* (2009), pp. 7–13. 1, 2, 4, 5

[Wal07] WALD I.: On fast Construction of SAH based Bounding Volume Hierarchies. In *Proceedings of the Symposium on Interactive Ray Tracing* (2007), pp. 33–40. 2

[Wal12a] WALD I.: Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture. *IEEE Transactions on Visualization and Computer Graphics 18*, 1 (2012), 47–57. 1, 2

[Wal12b] WALD I.: Fast Construction of SAH BVHs on the Intel Many Integrated Core (MIC) Architecture. *IEEE Transactions on Visualization and Computer Graphics 18*, 1 (2012), 47–57. 2

[WBB08] WALD I., BENTHIN C., BOULOS S.: Getting rid of packets-efficient simd single-ray traversal using multi-branching bvhs. In *Proceedings of Symposium on Interactive Ray Tracing* (2008), IEEE, pp. 49–57. 2, 5, 6

[WBKP08] WALTER B., BALA K., KULKARNI M., PINGALI K.: Fast Agglomerative Clustering for Rendering. In *Proceedings of Symposium on Interactive Ray Tracing* (2008), pp. 81–86. 2

[WWB*14] WALD I., WOOP S., BENTHIN C., JOHNSON G. S., ERNST M.: Embree: A kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics 33*, 4 (2014), 143:1–143:8. 1, 2