EUROGRAPHICS 2016 / J. Jorge and M. Lin
(Guest Editors)

Volume 35 (2016), Number 2

SAH guided spatial split partitioning for fast BVH construction

Per Ganestam and Michael Doggett

Lund University

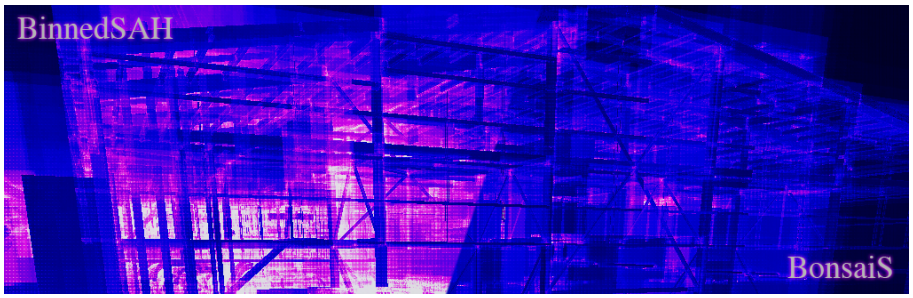


Figure 1: *Traversal cost visualization of the Power Plant model (12,759,246 triangles). The binned SAH builder is represented by the left half of the figure and our splitting approach together with the Bonsai BVH algorithm is represented by the right half. A brighter color means a higher traversal cost. The binned BVH is built in 1311ms and the Bonsai BVH using our triangle split approach is built in 1881ms. However, traversal cost is significantly reduced and rendering performance is improved by 60% when using our triangle split BVH.*

Abstract

We present a new SAH guided approach to subdividing triangles as the scene is coarsely partitioned into smaller sets of spatially coherent triangles. Our triangle split approach is integrated into the partitioning stage of a fast BVH construction algorithm, but may as well be used as a stand alone pre-split pass. Our algorithm significantly reduces the number of split triangles compared to previous methods, while at the same time improving ray tracing performance compared to competing fast BVH construction techniques. We compare performance on Intel's Embree ray tracer and show that BVH construction with our splitting algorithm is always faster than Embree's pre-split construction algorithm. We also show that our algorithm builds significantly improved quality trees that deliver higher ray tracing performance. Our algorithm is implemented into Embree's open source ray tracing framework, and the source code will be released late 2015.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Three-Dimensional Graphics and Realism—Display Algorithms

1. Introduction

For ray tracing techniques [Whi80], such as path tracing [Kaj86], to be a feasible choice of rendering it is necessary to accelerate triangle intersection computations with an underlying accelerating spatial data structure [KK86,PH10]. This data structure, often arranged as a binary tree, is used to speed up traversal of a three-dimensional scene to determine which triangle a ray intersects. Common types of data structures for ray tracing are kd-trees, grids, and the Bounding Volume Hierarchy (BVH), where the latter has gained a lot of popularity in recent years.

It is also important that the data structure is of high quality, in the sense that a higher quality data structure results in better ray tracing performance. A commonly used algorithm to construct high quality BVHs is a greedy top-down approach called sweep SAH, that uses the Surface Area Heuristic (SAH) [MB90]. Another crucial aspect of BVH construction, especially for animated scenes and real-time ray tracing, is that the data structure must be constructed quickly, since it may need to be rebuilt or updated every frame.

An often overlooked, but still important aspect for high quality tree construction is triangle splitting. Triangle splitting creates multiple references to a single triangle enabling the axis aligned nodes

in a BVH to have a tighter fit, resulting in improved ray traversal times.

Analyzing the results in Section 5 reveals some disadvantages of current triangle splitting approaches which may be either a lack of robustness or significantly increased BVH construction times and memory usage.

We propose a new fast top-down triangle splitting algorithm that produces BVHs with a quality similar to those of Split BVH (SBVH) [SFD09] but with build times close to those of the fast binned [Wal07] BVH construction approach. Our algorithm may be used as a preprocess to construction, or in conjunction with other top-down BVH approaches. By pairing our method with top-down construction algorithms, our splitting approach can take advantage of the partitioning stage already present in the algorithm. We evaluate our new algorithm by implementing it into the industry grade Embree [WWB*14] ray tracing system. We use our method both with sweep SAH as a pre-split pass and integrated with the Bonsai BVH construction algorithm [GBDAM15] and compare our results with some of the high performing and high quality BVH builders available in Embree.

Our main contribution is the fastest CPU based triangle split BVH builder to achieve ray tracing performance comparable to SBVH. Furthermore, our triangle split approach can be used as a preprocess to any BVH builder, and thus improve ray tracing quality by simply adding a pre-split module to existing frameworks. To attain our results we have developed a new simple but effective recursive triangle split approach and paired it with a novel in-place parallel partitioning scheme for recursively growing data.

2. Previous Work and Background

Havran and Bittner [HB02] presented the idea of split clipping, where the bounding box of an object is split to reduce empty overlap between object bounding boxes and *kd*-tree nodes. Rather than actually splitting triangles and storing the resulting polygons, or tessellating the polygons into several triangles, only the bounding boxes of triangles are actually split and re-computed, to create tighter axis-aligned bounds around the triangles. By performing split clipping rather than actual triangle splitting both computational complexity and the memory footprint are reduced.

Ernst and Greiner [EG07] applied a similar concept to BVHs by splitting triangle bounding boxes in a preprocess, called Early Split Clipping (ESC), before using a typical BVH construction pass.

Dammertz and Keller [DK08] suggested splitting triangles based on the tightness of the bounding boxes on each triangle edge and introduced a heuristic based on the volumes of the triangles' bounding boxes, the Edge Volume Heuristic (EVH).

Karras and Aila [KA13] introduced a new heuristic to the pre-splitting approach and the concept of a split budget. Their heuristic significantly improves split candidate selection and consequently improves BVH quality compared to earlier methods. By using a split budget they also reduce the risk of producing too many triangle splits.

Although computing triangle splits prior to BVH construction is a tempting approach, it also comes with certain impediments. As noted by Karras and Aila [KA13], both ESC [EG07] and the EVH [DHK08] fail in robustness. With the lack of knowledge of which triangle is valuable to split, some triangles may be split when they shouldn't, and some that should may not be split enough, or not split with the right split plane; sometimes resulting in worse ray tracing performance than a BVH without triangle splitting. Another shortcoming shared by the earlier pre-split approaches is that an a priori choice has to be made. Since ESC only relies on a predefined constant, it has to be hand tuned for every scene. The heuristic of EVH tries to mitigate the hand-tuning issue, however, it still has a constant that needs to be chosen. Although the pre-splitting heuristic by Karras and Aila [KA13] produces very good BVHs and is robust across a wide range of scenes, there is still the choice of split budget size. Sometimes a split budget of 10% of the triangles is enough, however, in other scenes a split budget of 50% is necessary to reach the potential increase in ray tracing performance. Thus the risk of performing too many splits is still inherent, since the BVH of some scenes cannot be improved by triangle splitting.

Stich et al. [SFD09] and Popov et al. [PGDS09] proposed similar ideas, where primitives are considered for splitting into both children during BVH construction, which resulted in tighter bounding boxes on a larger range of triangles than previous approaches. The top-down approaches by Stich et al. [SFD09] and Popov et al. [PGDS09] have the potential of producing superior BVHs. However, at the mercy of long build times. At each level of recursion, SBVH by Stich et al. [SFD09] picks the better of either using the best object split computed by sweep SAH, or the best spatial split among 256 equidistant split planes. To reduce the risk of performing too many triangle splits, and potentially running out of memory, SBVH incorporates a bias so that it may choose sweep SAH object split, even though spatial splits in fact could reduce SAH costs further. As of yet, the SBVH algorithm by Stich et al. [SFD09] produces the highest quality BVHs [KA13, AKL13].

Since we make extensive use of Bonsai, a fast CPU based BVH construction algorithm presented by Ganestam et al. [GBDAM15], it is worth mentioning a few details about the algorithm. Initially, Bonsai employs a fast approximate partitioning routine, using triangle mid-points only, to divide the scene into smaller spatially coherent groups of triangles. The triangle groups are passed to a BVH builder, in Bonsai a fast sweep SAH routine is used, and in parallel constructed into *mini-trees*. Prior to the last stage where the mini-trees are considered as leaf nodes in a sweep SAH pass, a *pruning* algorithm is applied to each mini-tree. Thus tightening the bounds of each mini-tree and correcting potential flaws to the full BVH caused by the initial partitioning stage.

Domingues and Pedrini [DP15] improve upon the performance of GPU based BVH construction by using an agglomerative process that merges nodes in treelets based on selecting the minimum surface area of the bounding boxes. They build on previous treelet reordering work that searched exhaustively for the best treelet [KA13]. Their technique is based on an initial tree being built using the LBVH method [LGS*09]. Domingues and Pedrini don't present any results with triangle splitting. However, their BVH builder may as well be combined with the pre-splitting approach by Karras et al. [KA13], or for that matter, our approach as a preprocess.

Previous triangle splitting algorithms, although producing high quality BVHs, are either slow in construction [SFD09, PGDS09], or in need of hand tuning or manual decisions prior to construction [EG07, DHK08, KA13]. Motivated by the little, or none, *a priori* knowledge needed for BVH construction and the prospect of producing superb trees, we continue in the fashion of the recursive triangle split approaches. However, to reduce build times, we pair our splitting method with the Bonsai BVH algorithm.

3. Algorithm

Our SAH guided mid-point split partitioning can be either integrated with the Bonsai BVH construction algorithm or work as a stand alone triangle pre-split method for any other BVH algorithm. Both with Bonsai and as a sweep SAH pre-split pass, our split partitioning approach results in improved ray tracing performance and when paired with Bonsai, ray tracing performance is similar to that of Embree's SBVH based spatial split builder.

3.1. The Surface Area Heuristic

Although the Surface Area Heuristic [GS87, MB90] for BVH construction in recent years has been found not to correlate perfectly with improved ray tracing performance [FFD09, AKL13, GBDAM15], it is still an advantageous approach in the construction of high quality BVHs. With this in mind we will briefly explain the SAH cost equations and how the SAH cost is evaluated and minimized in a top-down BVH construction algorithm.

The total SAH cost of a BVH can be computed as

$$C_i \sum_{n \in I} \frac{A(n)}{A(root)} + C_l \sum_{n \in L} \frac{A(n)}{A(root)} + C_t \sum_{n \in L} \frac{A(n)}{A(root)} N(n), \qquad (1)$$

where the equation expresses the expected cost of a random ray traversing the BVH, however in such a way, that it does not terminate within the scene geometry. Internal nodes are in the set I and the set L represents leaf nodes. The operator A represents the surface area of a node's bounding box and N represents the number of triangles in a node. The constants C_i, C_l, and C_t represent the costs of traversing an internal node, a leaf node, and intersecting a triangle, respectively.

Using SAH to optimize a top-down BVH builder is done by at each level of recursion evaluating and minimizing the equation

$$E_r = C_i A(n) + C_l(A(n_l)N(n_l)) + A(n_r)N(n_r)), \qquad (2)$$

which represents the cost of proceeding with the recursion, where n_l and n_r are the potential left and right child nodes. The result is compared to

$$E_t = C_t A(n)N(n), \qquad (3)$$

which represents the cost of terminating the recursion and using the current node n as a leaf node in the BVH. If $E_t < E_r$ an SAH optimal leaf node is found and the recursion terminates. Otherwise the recursion continues with n_l and n_r as child nodes. In our triangle split algorithm we use Equation 2 in a similar way to how it is used in recursive BVH construction.

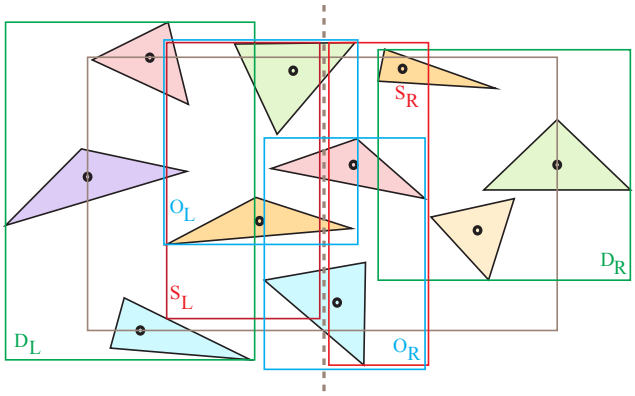


Figure 2: *The bounding boxes of the different sets of triangles used by our triangle split method. The brown box represents the mid-point bounds of all triangles and its spatial median is used to create the split plane. The blue bounds represent the overlap sets, the red bounds represent the split sets, and the green bounds represent the left and right disjoint sets.*

3.2. SAH guided mid-point split partitioning

In addition to computing the mid-point bounds used to find the split plane while partitioning, to know whether triangles should be subdivided or not, we need to compute the complete left and right bounding boxes. While partitioning, we accumulate six sets of different triangle categories. We create two completely *disjoint* sets of left and and right triangles, in relation to the mid-point split plane, where the two sets are denoted as D_L and D_R and $D_L \cap D_R = \emptyset$. We also store two *overlap* sets, O_L and O_R, of those triangles that have their mid-points to the left or right side of the split plane but with bounding boxes overlapping the split plane. The last two sets, the *split* sets, contain the left and right halves of subdivided triangles respectively, and are denoted as S_L and S_R. The triangles in each of the split sets are exactly the same as the triangles in the overlap sets, i.e. $O_L \cup O_R = S_L = S_R$. However, due to split clipping [HB02], the left and right subdivided triangles have different bounding boxes. This also implies that the split sets contain twice as many triangles as the overlap sets.

Figure 2 shows the bounding boxes and their associated groups of triangle indices created around the split plane. As in the Bonsai algorithm [GBDAM15], to avoid empty partitions, we use the mid-point bounds rather than full bounds when choosing the split plane. As a consequence of this, the split plane used to subdivide triangles isn't the spatial median of the complete bounding box of a partition, but instead chosen as the spatial median of the mid-point bounds. We choose the largest axis for partitioning.

Once the bounds of the six sets are computed we evaluate the benefit of splitting triangles by computing the SAH cost when using the overlap sets

$$C_O = A(D_L \cup O_L)|D_L \cup O_L| + A(D_R \cup O_R)|D_R \cup O_R|$$

and comparing the result to the SAH cost when using the split sets,

evaluated as

$$C_S = A(D_L \cup S_L)|D_L \cup S_L| + A(D_R \cup S_R)|D_R \cup S_R|,$$

where A is the surface area of a set, C_O is the SAH cost of keeping the original triangles, and C_S is the SAH cost of using the split triangles. Split partitioning is continued by choosing whichever set has the lowest SAH cost.

Our split partitioning algorithm continues the recursion until a treshold of triangle count has been reached. The threshold is based on the Bonsai algorithm's mini-tree size and can be set arbitrarily. However, we have found that for smaller scenes ($< 100,000$ primitives) a smaller threshold of 512 is necessary to perform enough triangle splitting, and for larger scenes ($> 4,000,000$ primitives), a value of 8192 as a threshold is enough to achieve a high quality BVH. We linearly interpolate the threshold value between its minimum and maximum values. Other than the mini-tree size threshold, our method does not require any other bias or tuning variables to guarantee enough splitting or to avoid over-splitting.

4. Implementation

Since we have implemented our splitting algorithm in Embree, we also make use of some of their design choices. The type *primitive reference*, denoted *primref*, is a structure of six floats representing a triangles bounding box and two integers used as references to the original mesh and triangle. Since the size of a *primref* is $(6+2)\cdot 32$ bits, it fits into a 256-bit AVX register, which is useful for fast bounding box computations. As an initial pass, all algorithms we present create an array of *primrefs* containing the original bounding boxes and indices of each triangle. It is the *primref* array that is used in BVH construction and triangle splitting. A split triangle is simply two *primrefs* referencing the same original triangle, but having different bounding boxes. To avoid confusion, in the following section we simply talk about triangles, although all operations are actually done on *primrefs*.

4.1. Triangle Splitting and Recursively Growing Memory

One of the most difficult tasks with a recursive algorithm that needs to expand memory wise, is efficient memory management. In practice, keeping the different sets (split, overlap, and disjoint) in separate arrays and then merging them when a decision whether to split or not is taken, although much easier to implement, is inefficient memory wise.

We perform a quick-sort style in-place partitioning with the split plane as the pivot. To minimize memory traffic, we track the index of any triangle that belongs to one of the overlap sets instead of making an actual copy of the triangle. When partitioning discovers an overlap triangle, it places that triangle to the left or right of the split plane depending on its bounding box midpoint, as is done with a non-split partitioning. Thus there is no need for any auxiliary storage of the overlap sets as it is with the split sets. Since we store the index of the overlap triangles, it is possible to later substitute an overlap triangle for a split triangle, if the SAH-cost calculation determines that it is more beneficial to keep the split sets of triangles than the overlap sets. The partitioning performed at each recursion is illustrated in Figure 3.

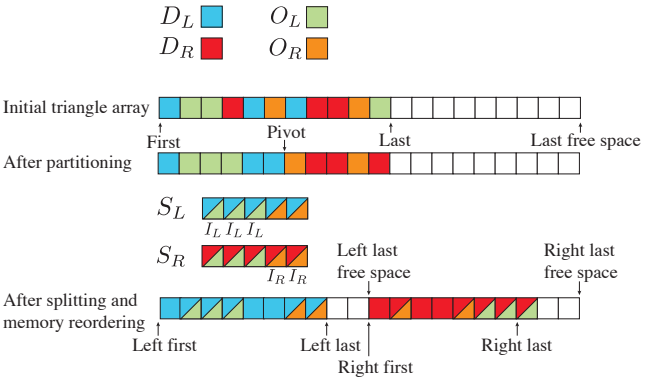


Figure 3: *Triangle array memory management in each recursion. The initial array contains disjoint left and right triangles (blue and red) and overlap triangles (green and orange). After in-place partitioning, the left side of the split plane contains $D_L \cup O_L$ and the right side its counterparts. Two separate arrays of temporary left and right split triangles contain their respective parts of split triangles. An indexed triangle, one that can be inserted where an overlap triangle currently resides, in the left split set S_L is colored blue and green and marked with I_L. An indexed triangle in the right set S_R is colored red and orange and marked with I_R. If the SAH metric finds the split sets favorable, indexed triangles are inserted where the overlap triangles are and a small segment of the right partition is moved as the last part of the diagram illustrates. Note that this is an extreme case were almost 50% of the triangles are split, and it is merely for illustrative purposes.*

A memory problem arises when it is decided that the split sets should be used. Since there are twice as many triangles in the split sets as there are in the overlap sets, and only half of the split triangles are indexed and can be inserted into the original triangle array. In the first level of split partitioning, before any recursive calls have been made, it is possible to simply append the non-indexed triangles of the right split set S_R to the end of the original partition. However, it doesn't solve the problem with the non-indexed left split triangles. There is no space to place them in-between the pivot and the triangles that belong to the right of the split plane. Thus, as many right triangles as there are non-indexed left split triangles have to be moved to the end of the original partition as well. Then there would be enough space for both the left and right split sets.

Due to the recursive fashion of our algorithm, it becomes more problematic with the succeeding recursions. In most cases, there won't be any empty space available at the end of the partition, and all triangles of one of the halves would have had to been moved to new memory every time the split sets are better than the overlap sets. This memory overhead can be solved by always rebalancing the available empty space relative to the partition sizes and making sure that enough empty space is created in each partition as the algorithm recurses.

Whether the split sets produce a lower SAH cost than the overlap sets or not, we always perform a rebalancing of empty space. In each recursion of the left and right partitions, they are rearranged to supply each side with a fraction of the available empty space

in proportion to the current size of the partitions. As an example, after splitting and partitioning, if there is space for 15 additional triangles at the end of the right partition but no space between the left and the right partitions, and the left partition is twice as large as the right partition, then after rebalancing there would be space for 10 additional triangles at the end of the left partition and 5 at the end of the right partition.

If there isn't enough space for both S_L and S_R after split partitioning, then the whole right set $D_R \cup S_R$ has to be moved to a new memory region, with an additional 20% extra padding space as well. The left side gets all the memory left behind by the right side.

We empirically found that the 20% additional space for the partitions works in a robust way for all of our scenes. It is rare that a partition grows with more than a few percent of its size.

In a pathological worst case scenario where all triangles in a scene need to be split, the size of the temporary split set arrays S_L and S_R would need to be as large as the entire scene, but such a scene would cause trouble to any split builder. We found that the actual worst case space needed for the split set arrays across all our scenes is 1% of the triangle count in size. Often they are much smaller than that. For San Miguel the largest temporary split set array is only 0.05% in size relative to the number of triangles of the scene.

Bonsai [GBDAM15] permits gaps between each final partition and some unused space at the end of the partitions will not affect other stages of the builder. However, when using our splitting algorithm as a pre-split pass, a compaction of the triangle array prior to BVH construction is necessary.

Task parallelism is implemented by recursively spawning new tasks as more partitions are created. We utilize data parallelism when swapping positions of triangles and when computing new bounding boxes.

The total memory allocations needed for our split partitioning algorithm additional to the initial triangle array are a dynamically growing S_L and S_R pair per thread and the dynamically growing memory of the triangle array. Our in-place growing memory partitioning scheme allocates at worst about twice as much additional memory than needed. As an example, the Power Plant scene is built with 19% additional split triangles but allocated space is increased by 39%.

4.2. Bonsai and 8-wide Trees

To maximize ray tracing performance on modern CPUs Embree utilizes 8-wide SIMD instructions (AVX) when ray tracing. However, to efficiently gain data level parallelism while traversing a BVH, Embree builds 4-wide or 8-wide trees for SSE or AVX hardware. We had to modify the Bonsai algorithm to also build 8-wide trees. The initial partitioning of Bonsai doesn't store any BVH node information and is not affected by the fact that the BVH nodes will have eight children. However, mini-tree construction and top tree construction must be adapted to build 8-wide trees. Since both the top tree and the mini-trees use the sweep SAH algorithm, they are modified in the same way. Just like in Embree's binned SAH builder, instead of creating a node once an SAH optimal pivot has been found, the two halves are placed in a priority queue. While the queue is smaller than the maximum number of children of a node, the element with the largest SAH cost in the queue is chosen for further partitioning. Once the queue has reached the maximum branching size, a BVH node is created and the children are further partitioned recursively.

One issue with the top tree in 8-wide BVH construction that doesn't exist for binary trees, is that it isn't guaranteed that the top tree gets precisely the maximum number of children (which are mini-trees) in its leaf nodes, causing partially filled nodes in the middle of the tree. Initially we thought it would be better to move the gaps to the leaves of the BVH and when a partially filled node was created, the empty space was propagated down the tree. However, we didn't see any tree quality improvement by moving the empty nodes from the middle of the tree to the leaf nodes. It was simply wasted CPU cycles, thus we decided to leave the partially filled nodes as they were.

Triangle splitting is integrated into Bonsai simply by exchanging the mini-tree partitioning with our split partitioning.

5. Results

Our main results have been generated using an Apple Macbook Pro laptop with a quad core Intel 4850HQ CPU. Our primary results are produced by running our triangle split implementations in the Embree ray tracing framework and using the path tracer available in Embree, in benchmark mode, for tree quality measurements. We compare several different construction algorithms with and without splitting and the results are shown in Table 1. For non-split algorithms we include a standard sweep SAH, SWEEPSAH, the original non-split version of Bonsai (BONSAI), and the binned SAH algorithm included in Embree, BINNEDSAH. Our new splitting algorithm is integrated into the Bonsai construction algorithm and denoted BONSAIS. To demonstrate the ability to use our splitting algorithm with other construction techniques, we apply it as a pre-split pass to sweep SAH, SWEEPPRE. We also compare to the two splitting algorithms available in the Embree. The first is a pre-split builder BINNEDPRE, and the second is a spatial split algorithm BINNEDS, based on SBVH [SFD09].

BVH performance comparisons are done using Embree version 2.7.1. It is worth noting that our implementation of SWEEPSAH is not optimized for build time, although, Ganestam et al. [GBDAM15] showed that SWEEPSAH can achieve competitive build performance.

In Table 1 build times are presented in milliseconds and rendering performance is the average time in milliseconds of 10 frames rendered using the path tracer supplied by Embree. A frame is simply a rendering pass with one sample per pixel and many frames need to be accumulated for a final image to converge. The more samples and longer time an image need to converge the more beneficial it is with a high quality BVH.

We have marked the two algorithms resulting in the best rendering performance in bold. If more than two algorithms share the best performance percentage, they are all marked. Among the al-

gorithms with the best rendering performance we also marked the one with the fastest build time.

Across our twelve test scenes, BONSAIS is always the fastest triangle split algorithm. Additionally, BONSAIS is among the two best performing algorithms 10 out of 12 times and in four occasions, Bonsai with integrated triangle splitting even out performs BINNEDS. SWEEPPRE is among the top performing algorithms in two occurrences, and tends to have a ray tracing performance in between BINNEDS and BINNEDPRE on scenes where triangle splitting is beneficial. BINNEDS improves rendering performance on all scenes but one compared to SWEEPSAH. However, on scenes that don't benefit much from splitting, the performance gain is small, and at the cost of significantly longer build times.

An example of a scene where triangle splitting does not improve rendering performance is Dragon. The reason is that Dragon only contains finely tessellated and evenly distributed triangles, and non-split builders like BINNEDSAH can easily minimize the SAH cost.

Figure 4 illustrates multicore performance scaling of the Power Plant scene with BONSAS and BINNEDSAH. Scaling measurements were performed on an Intel E5-2643V3 dual socket 12 core CPU. Although BINNEDSAH scales slightly better with an increasing number of CPU cores than BONSAIS, it is worth noting that BONSAIS build times are close to BINNEDSAH build times and ray tracing performance on the 12 core CPU using BONSAIS is 105*ms* per frame and with BINNEDSAH 162*ms* per frame. Since BINNEDSAH and BONSAIS present significantly better build performance than BINNEDS we omit BINNEDS in the multicore scaling comparisons.

Although it is easy to measure memory performance of BONSAIS it is difficult to make thorough comparisons in regards of build time memory usage to the BINNEDS algorithm. The source code of BINNEDS presents many small memory allocations and memory free instructions. We added a counter to see whether the actual number of allocations were significant or not and found that a large number of allocations were made. However we didn't have the means to measure the maximum memory allocated at any time. To further investigate memory usage we executed both BONSAIS and BINNEDS through the well known memory analysis tool Valgrind. Valgrind couldn't tell us what the largest amount of memory allocated at any time was either, but it could inform us about the total number of allocations made by both algorithms. In the tests we made, BINNEDS allocated 34× more memory in total compared to BONSAIS and for Arabic City BONSAIS allocated 735MB of memory and BINNEDS allocated 25GB of memory, not counting any free instructions.

5.1. Binary Trees

Since Embree only implements 4-wide and 8-wide BVHs we also present results on a subset of our scenes using a standard binary BVH. In this comparison we also use SWEEPSAH as a baseline but present only one additional algorithm, BONSAIS. The ray tracer used in the second comparison performs ray traversal on the GPU but computes shading on the CPU. The GPU calculations are done on an Intel Iris Pro 5200 integrated graphics processor.

In our Embree implementation, we noticed that the improved

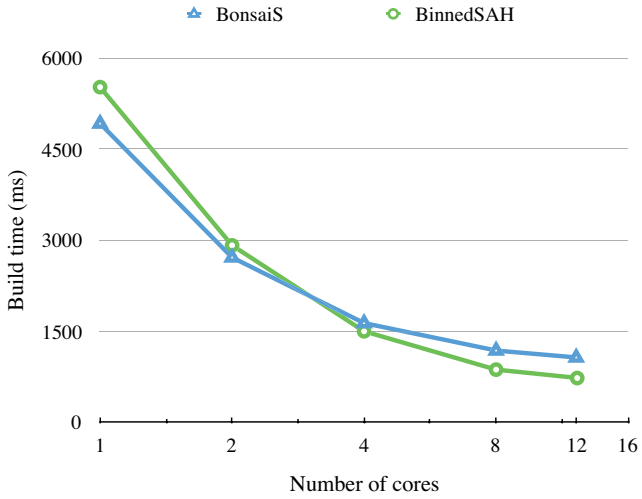
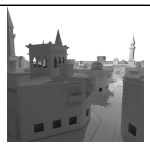


Figure 4: *Multicore performance scaling of the Power Plant scene. Although* BONSAIS *is close,* BINNEDSAH *presents slightly better multicore scaling. Scaling measurements were conducted on a dual socket 12 core CPU. Note the logarithmic x-axis scale.*

rendering times using triangle splitting didn't always match the expected rendering improvement we had seen prior to integrating our methods to Embree. One observation that partially explains the gap in rendering performance between the 8-wide Embree implementation and the 2-wide GPU implementation is that it may be less advantageous for wider trees to perform triangle splitting. We found that if we forced Embree to actually build binary BVHs, although the ray tracer would still use its 8-wide AVX implementation, on some scenes, the reduced rendering time benefit from triangle splitting compared to no splitting would differ with about 5% compared to the splitting benefit with 8-wide trees. This doesn't fully account for the discrepancy, where as an example Arabic City, in Table 1 using 8-wide trees BONSAIS performs at 84% of SWEEPSAH, but in Table 2 using 2-wide trees, the rendering time is reduced to 65% of SWEEPSAH. The rest of the discrepancy can then only be explained by using different hardware or a different ray tracer, or more likely, a combination of the two.

Due to this discrepancy we also present results of a sub set of our test scenes by ray tracing standard binary BVHs. We do this with our own GPU based ray tracer, with shading computations done on the CPU. The improved rendering times in Table 2 can also be put in relation to the rendering speed improvements reported by Karras et al. [KA13]. For Arabic City their SBVH implementation reduces rendering times to 62% compared to SWEEPSAH. The same comparison results in 71% for Sponza and 73% for San Miguel. The rendering times of the same three scenes of the fast triangle split builder by Karras et al. [KA13] are reduced by 74%, 73% and 77% respectively compared to SWEEPSAH. For Arabic City and Sponza the triangle counts are increased by 50% and for San Miguel the count is increased by 30%. On the same scenes, BONSAIS reduces rendering times to 65% for Arabic City, 75% for San Miguel, and 60% for Sponza, compared to SWEEPSAH, while having an adaptive split count that is increased by 34% for

	 Arabic City		 Crown		 Dragon		 Fairy Forest		 Italian City		 Kalabsha	
Triangles	416,236		4,868,924		7,349,978		174,117		382,029		4,542,705	
	Build	Trace	Build	Trace	Build	Trace	Build	Trace	Build	Trace	Build	Trace
SweepSAH	254	295 [100%]	4680	325 [100%]	7595	129 [100%]	117	299 [100%]	240	300 [100%]	4910	551 [100%]
Bonsai	34	291 [99%]	469	331 [102%]	664	131 [102%]	16	299 [100%]	31	292 [97%]	451	554 [101%]
BinnedSAH	33	302 [102%]	447	322 [99%]	**674**	**126 [98%]**	**14**	**296 [99%]**	31	297 [99%]	459	555 [101%]
SweepPre	**374**	**243 [82%]**	5414	322 [99%]	8087	134 [104%]	147	301 [100%]	322	239 [80%]	6098	550 [100%]
BonsaiS	53	247 [84%]	**535**	**316 [97%]**	827	132 [102%]	24	**297 [99%]**	**47**	**231 [77%]**	**765**	**513 [93%]**
BinnedPre	91	267 [91%]	980	317 [98%]	1577	134 [104%]	40	305 [102%]	77	254 [85%]	811	572 [104%]
BinnedS	404	**237 [80%]**	4130	**304 [94%]**	2205	**124 [96%]**	129	**289 [97%]**	316	**229 [76%]**	4646	**505 [92%]**

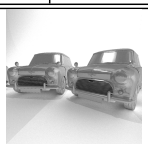
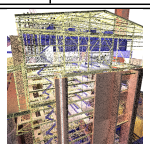
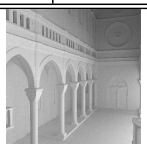
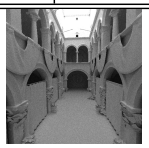
	 Mini		 Power Plant		 Sala		 San Miguel		 Sibenik		 Sponza	
Triangles	912,411		12,759,246		400,637		7,880,512		79,380		262,267	
	Build	Trace	Build	Trace	Build	Trace	Build	Trace	Build	Trace	Build	Trace
SweepSAH	719	593 [100%]	14347	617 [100%]	283	298 [100%]	8208	394 [100%]	49	222 [100%]	178	1210 [100%]
Bonsai	74	581 [98%]	1261	606 [98%]	34	298 [100%]	711	363 [92%]	9	225 [101%]	22	1137 [94%]
BinnedSAH	83	592 [100%]	1311	700 [113%]	35	332 [111%]	798	386 [98%]	7	222 [100%]	22	1393 [115%]
SweepPre	952	**548 [92%]**	17603	469 [76%]	359	296 [100%]	9301	359 [91%]	53	222 [100%]	213	1224 [101%]
BonsaiS	**106**	**538 [91%]**	**1881**	**437 [71%]**	**47**	**286 [96%]**	**835**	**339 [86%]**	**11**	**218 [98%]**	**29**	**1057 [87%]**
BinnedPre	166	617 [104%]	3625	606 [98%]	90	339 [114%]	2168	388 [98%]	14	229 [103%]	56	1244 [103%]
BinnedS	739	572 [96%]	13835	**441 [71%]**	342	**292 [98%]**	7118	**341 [87%]**	53	**213 [96%]**	168	**1113 [92%]**

Table 1: *BVH build time and rendering performance measurements across all scenes and algorithms. Build times are reported in milliseconds and ray tracing performance in milliseconds per frame. The reported frame time is the average of ten frames rendered with the path tracer available in Embree, set to benchmark mode. The reported build times are generated by taking the minimum build time when running each builder 20 times. In regards of both build times and rendering performance, lower is better. The two algorithms that result in the highest ray tracing performance are marked, and so is the fastest build time among those with the best rendering performance.*

	 Arabic City	 Crown	 Italian City	 Kalabsha	 San Miguel	 Sponza
SweepSAH	240 [100%]	191 [100%]	129 [100%]	455 [100%]	667 [100%]	630 [100%]
BonsaiS	155 [65%]	183 [96%]	86 [67%]	330 [73%]	499 [75%]	380 [60%]

Table 2: *Rendering performance comparison of* SweepSAH *and* BonsaiS *using binary BVHs and on a GPU based ray tracer. Measurements are reported in milliseconds per frame.*

Arabic City, 6% for San Miguel, and 23% for Sponza. Since the algorithms are designed for and executed on different hardware it is difficult make direct build time comparisons. However, BonsaiS and the triangle split builder by Karras et al. [KA13] exhibit similar performance improvements, but BonsaiS results in a significantly smaller increase of triangles.

5.2. Triangle counts

Table 3 presents the increase in triangle counts due to splitting triangles. Our splitting algorithm and the binned spatial split builder BinnedS both split triangles adaptively and thus find triangle splits that greedily minimize the SAH costs. The pre-split approach used in Embree depends on a pre-defined splitting budget and thus may split too much or too little in some scenes. Generally, BonsaiS creates fewer additional triangles than the two triangle split methods available in Embree. The pre-split approach in Embree tends to increase the original triangle count by close to 50%. This is because of the split budget that allows a maximum increase of 50%. The algorithm will continue to split large triangles until it is close to its split budget, even though it may not always further improve rendering performance. For the Power Plant model, BinnedS creates more than twice the number of additional triangles than BonsaiS, even so, rendering performance is identical. For San Miguel, the difference in triangle splits is even greater, where BonsaiS only adds 6% additional triangles compared to 45% with BinnedS but BonsaiS results in slightly better rendering performance. Again, we see how dragon isn't affected much by splitting and most of its split computations are discarded.

	BONSAIS	BINNEDPRE	BINNEDS
Arabic City	34%	47%	61%
Crown	9%	46%	34%
Dragon	2%	46%	1%
Fairy Forest	19%	46%	24%
Italian City	27%	48%	59%
Kalabsha	11%	22%	46%
Mini	24%	46%	42%
Power Plant	19%	47%	50%
Sala	19%	45%	44%
San Miguel	6%	50%	45%
Sibenik	19%	46%	29%
Sponza	23%	45%	25%

Table 3: *The triangle count increase due to splitting.* BONSAIS *represents the counts for* SWEEPPRE *as well, since they use the same splitting algorithm.* BINNEDSAH *with pre-split always stays close to a 50% increase, since this is the allowed splitting budget. Our method and* BINNEDS *are both adaptive split techniques, and reduce the probability of keeping unnecessary splits.*

	Split time (ms)
Arabic City	13 [25%]
Crown	151 [28%]
Dragon	294 [36%]
Fairy Forest	11 [46%]
Italian City	11 [23%]
Kalabsha	394 [52%]
Mini	30 [28%]
Power Plant	844 [45%]
Sala	17 [36%]
San Miguel	248 [30%]
Sibenik	4 [36%]
Sponza	8 [28%]

Table 4: *The time in milliseconds our triangle splitting approach takes when used as a pre-split pass. In brackets we show the time spent on split partitioning relative to full* BONSAIS *build times. On split friendly scenes, such as Italian City, the relative time spent on splitting is lower than on less split friendly scenes. This is inherent, since all triangles that cross the split plane have to be considered as a split candidate, even if the split never is used. However, if the number of triangles used by the BVH builder isn't increased, the build times are lower than they would be if the splits were used. Thus increasing the ratio between triangle split time and BVH build time.*

5.3. Splitting performance

In Table 4 we present the actual time our splitting algorithm takes as a stand alone pre-split approach and the percentage of the BONSAIS build time that is spent on split partitioning. When our method is used with Bonsai the splitting time is merged with the initial mini-tree partitioning. As an example, when using our triangle splitter as a pre-split on San Miguel, the extra build time added is 248ms. Since the splitting algorithm is integrated into BONSAI, and the original partitioning pass of BONSAI takes 133ms for San Miguel, the added build time for BONSAIS in relation to BONSAI is $248 - 133 = 155$ms. Any other extra build time added for BONSAIS and other algorithms using our split method would be from the fact that there are more triangles to process while building the BVH.

5.4. SAH cost

Table 5 shows the SAH cost of the competing algorithms. BONSAIS consistently produces low SAH costs, not always the lowest, but close. Table 5 also demonstrates that SAH cost doesn't necessarily correlate with rendering performance [FFD09, AKL13, GBDAM15]. This is clearly seen on Fairy Forest and San Miguel, where BINNEDPRE greatly increases the SAH cost without significantly reducing ray tracing performance compared to BINNEDSAH.

6. Conclusion

We have presented a new triangle split algorithm for fast BVH construction on multicore CPUs using contemporary SIMD extensions. Our triangle split approach paired with Bonsai is always the fastest triangle split builder among the presented algorithms and achieves a rendering performance similar to, and some times better than, the SBVH based spatial split builder available in Embree. We achieve fast build times by utilizing a parallel in-place partitioning scheme for recursively growing data, and improved BVH quality by employing an SAH guided triangle split technique while partitioning. Our algorithm reaches its full potential in regards of both build times and rendering performance when paired with Bonsai. We have also showed that our method works well as a pre-split pass prior to BVH construction, and can easily be added to existing BVH builders without any invasive procedures.

We consider our contributions as continued work towards a high quality real-time BVH construction.

As future improvement, our algorithm could benefit from a more sophisticated parallel approach in the early stages of partitioning. Rather than using only one thread in the first level of recursion, all available threads could work on the same partition until the number of partitions equals the number of hardware threads available.

Acknowledgements

Thanks to ELLIIT and the Intel Visual Computing Institute for funding. We would like to thank Veronica Sundstedt for the Kalabsha temple model and Timo Aila for the Italian and Arabic City models. We would also like to thank Martin Lubich, http://www.loramel.net, for the Crown model and Gilles Tran, http://www.oyonale.com, for the Mini model.

References

[AKL13] AILA T., KARRAS T., LAINE S.: On Quality Metrics of Bounding Volume Hierarchies. In *High-Performance Graphics* (2013), pp. 101–107. 2, 3, 8

	SweepSAH	SweepPre	Bonsai	BonsaiS	BinnedSAH	BinnedPre	BinnedS
Arabic City	23.23	17.21	22.43	17.10	23.83	20.38	**16.80**
Crown	7.91	**7.48**	7.77	7.77	7.93	7.79	7.58
Dragon	6.22	6.16	**6.02**	6.48	6.50	7.44	6.50
Fairy Forest	9.50	9.89	**9.42**	9.63	9.76	15.17	9.82
Italian City	17.94	13.83	17.49	**13.64**	19.05	17.27	13.66
Kalabsha	2.31	2.34	**2.24**	2.42	3.42	6.55	3.39
Mini	5.10	5.16	**4.94**	5.31	6.19	9.57	5.61
Power Plant	10.02	8.34	9.83	8.17	10.67	9.37	**7.32**
Sala	10.43	10.41	10.67	**10.00**	12.05	13.08	10.61
San Miguel	18.86	16.92	19.12	16.00	19.75	24.45	**15.99**
Sibenik	15.03	14.38	14.59	14.06	14.78	16.02	**13.45**
Sponza	22.53	22.25	22.61	**20.78**	24.59	29.26	21.65

Table 5: *SAH costs of all scenes and construction methods. For each scene, the builder with the lowest cost is marked with bold text.* BonsaiS *and* BinnedS *tend to have the lowest SAH costs, but no algorithm is consistently lowest. The C_i and C_t constants used to evaluate Eq. 1 are set to one and C_l is zero.*

[DHK08] Dammertz H., Hanika J., Keller A.: Shallow Bounding Volume Hierarchies for Fast SIMD Ray Tracing of Incoherent Rays. *Computer Graphics Forum, 27*, 4 (2008), 1225–1233. 2, 3

[DK08] Dammertz H., Keller A.: Edge volume heuristic - robust triangle subdivision for improved BVH performance. In *Proc. 2008 IEEE/EG Symposium on Interactive Ray Tracing* (2008), pp. 155–158. 2

[DP15] Domingues L. R., Pedrini H.: Bounding Volume Hierarchy Optimization through Agglomerative Treelet Restructuring. In *High-Performance Graphics* (2015), pp. 13–20. 2

[EG07] Ernst M., Greiner G.: Early Split Clipping for Bounding Volume Hierarchies. In *IEEE Symposium on Interactive Ray Tracing* (2007), pp. 73–78. 2, 3

[FFD09] Fabianowski B., Flower C., Dingliana J.: A cost metric for scene-interior ray origins. In *Eurographics Short Papers* (2009), pp. 49–52. 3, 8

[GBDAM15] Ganestam P., Barringer R., Doggett M., Akenine-Möller T.: Bonsai: Rapid Bounding Volume Hierarchy Generation using Mini Trees. *Journal of Computer Graphics Techniques, 4*, 3 (September 2015), 23–42. 2, 3, 5, 8

[GS87] Goldsmith J., Salmon J.: Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics & Applications, 7*, 5 (1987), 14–20. 3

[HB02] Havran V., Bittner J.: On Improving KD-Trees for Ray Shooting. In *Winter School on Computer Graphics* (2002), pp. 209–217. 2, 3

[KA13] Karras T., Aila T.: Fast Parallel Construction of High-quality Bounding Volume Hierarchies. In *High-Performance Graphics Conference* (2013), pp. 89–99. 2, 3, 6, 7

[Kaj86] Kajiya J. T.: The Rendering Equation. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)* (1986), vol. 20, pp. 143–150. 1

[KK86] Kay T. L., Kajiya J. T.: Ray Tracing Complex Scenes. In *Computer Graphics (Proceedings of SIGGRAPH 86)* (1986), vol. 20, pp. 269–278. 1

[LGS*09] Lauterbach C., Garland M., Sengupta S., Luebke D., Manocha D.: Fast BVH Construction on GPUs. *Computer Graphics Forum, 28*, 2 (2009), 375–384. 2

[MB90] MacDonald D. J., Booth K. S.: Heuristics for Ray Tracing Using Space Subdivision. *Visual Computer, 6*, 3 (1990), 153–166. 1, 3

[PGDS09] Popov S., Georgiev I., Dimov R., Slusallek P.: Object Partitioning Considered Harmful: Space Subdivision for BVHs. In *High-Performance Graphics* (2009), pp. 15–22. 2, 3

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering: From Theory to Implementation*, 2nd ed. MKP, 2010. 1

[SFD09] Stich M., Friedrich H., Dietrich A.: Spatial Splits in Bounding Volume Hierarchies. In *High-Performance Graphics* (2009), pp. 7–13. 2, 3, 5

[Wal07] Wald I.: On Fast Construction of SAH-based Bounding Volume Hierarchies. In *IEEE Symposium on Interactive Ray Tracing* (2007), pp. 33–40. 2

[Whi80] Whitted T.: An Improved Illumination Model for Shaded Display. *Communications of the ACM, 23*, 6 (1980), 343–349. 1

[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. 2