Visibility Driven BVH Build Up Algorithm for Ray Tracing

Marek Vinkler, Vlastimil Havran, and Jiří Sochor.

Visibility Driven BVH Build Up Algorithm for Ray Tracing

Marek Vinkler[a,1,*], Vlastimil Havran[b], Jiří Sochor[a]

[a] *Faculty of Informatics, Masaryk University, Botanická 554/68a, 602 00 Brno, Czech Republic*
[b] *Faculty of Electrical Engineering, Czech Technical University, Karlovo náměstí 13, 121 35 Praha 2, Czech Republic*

Abstract

The minimization of traversal cost using surface area heuristic is extensively used to build high quality spatial subdivisions and bounding volume hierarchies for ray tracing. Despite the fair performance of trees built with the cost model, it is known that the underlying assumptions for surface area heuristics are not realistic. In this paper we show how the cost function of the surface area heuristic can be improved on using the assumed visibility of geometric primitives such as triangles. This way the build algorithm utilizes the exact or assumed visibility to construct more efficient BVHs by traversing smaller portion of the hierarchy. We show that by these inexpensive modifications to the cost function we can speed up the ray traversal by approximately 102% on average for path tracing of highly occluded scenes compared to standard surface area heuristics. Moreover, it is also possible to lower the construction time and memory usage by subdividing only those parts of the animated scene through which rays are expected to be traversed.

Keywords:
ray tracing, bounding volume hierarchies, BVH build algorithm, surface area heuristic

1. Introduction

Ray tracing is a fundamental operation used in rendering algorithms to synthesize images of a virtual scene. A crucial component influencing the performance is the quality of the acceleration data structures used to prune the search along the ray. In the rest of the paper we will deal only with hierarchical data structures that use a tree to encode the spatial regions or objects.

Many hierarchical data structures have been developed but only a few are commonly used in practice. *Kd-trees* offer the fastest traversal performance in general but they can require increased build times and higher memory usage. Another efficient data structure is the *Bounding Volume Hierarchy* (BVH) which performs comparably well to kd-trees. BVHs are considered to be more efficient for dynamic scenes as they can be refitted instead of a full rebuild and have a smaller memory footprint. BVHs are used throughout this paper but the findings could be applied to kd-trees as well.

The hierarchical data structures generally used are built up with the cost model with surface area heuristic (SAH) [1, 2]. Although data structures built in such a way perform reasonably well even for irregular distributions of objects, the assumptions for the underlying cost model are known to be unrealistic. In particular, the cost model with SAH expects a uniform distribution of rays in scene space and, more importantly, it assumes that rays do not intersect any objects. Both these assumptions are almost always violated when rendering any particular image.

In this paper we alleviate both these assumptions. We use the information about triangle visibility in the context of animation to suggest a new cost model that produces more efficient BVHs for scenes with high occlusion. Moreover, as we modify only the cost model the technique should be orthogonal to other methods for building high quality BVHs. The benefit of our method is visualized as colour coded workload in Figure 1.

The results of ray tracing BVHs built with our method are correct even if the information about the visibility is only approximate. This allows the method to be used in the context of animation where approximate visibility can be gathered during the rendering of the previous frame.

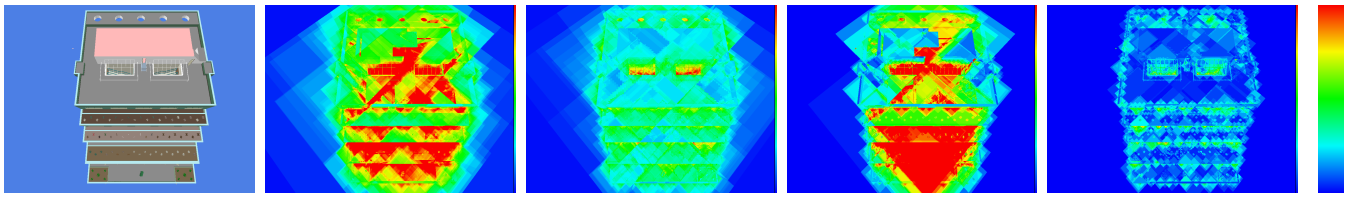
To show the hypothetical gains achievable by visibility driven build up, we compare BVHs built according to our heuristic with SAH based BVHs built over visible triangles only. The visible triangle BVHs are faster to traverse because they are built over the smallest possible number of triangles needed to get the correct result.

In section 2 we describe previous work. In section 3 we discuss background needed for the description of the algorithm in section 4. We present the results in section 5, followed by conclusion in section 6.

2. Related work

The building algorithms for BVHs in the context of ray tracing have been intensively studied in several directions. The CPU algorithms mostly focused on improving tree quality [3, 4]

*Corresponding author
Email addresses: xvinkl@fi.muni.cz (Marek Vinkler), havran@fel.cvut.cz (Vlastimil Havran), sochor@fi.muni.cz (Jiří Sochor)
[1] tel: 00420549495357



(a) (b) (c) (d) (e)

Figure 1: The visualization of the algorithm for primary rays using the scene Soda Hall rotated by 45 degrees: (a) rendered image with shading, (b) the number of traversal steps per ray for the reference (SAH) method, (c) the number of traversal steps per ray for our (OSAH) method, (d) number of intersection tests for the reference (SAH) method, and (e) number of intersection tests for our (OSAH) method. The colour of each pixel codes the amount of work done for that pixel, ranging from blue (the lowest value), via magenta to green, then to yellow and red (the highest value).

but some of them considered build times as well [5, 6]. Recently, the efficient storage of BVHs has also been researched in pursuit of realtime frame rates on massive models [7, 8]. On the other hand, GPU build up algorithms [9, 10, 11] targeted mainly the building times.

The final performance of ray tracing system is dependent not only on the BVH quality but also on the efficiency of its traversal. The bulk of the recent research targeted GPUs [12, 13] because of their superior floating point performance. Let us detail the most important papers related to ray tracing of BVH.

Efficient BVHs. The first top-down method of building up BVHs with spatial median was introduced by Kay and Kajiya [14]. Goldsmith and Salmon [1] showed that ray tracing performance fairly corresponds to the cost of the BVH and they suggested the insertion-based algorithm for BVH. Using this observation Ng and Trifonov [15] and Kensler [3] proposed to post-process BVH build up with a phase that improves the tree cost through series of subtree rotations.

Hybrid hierarchies. Another technique improving the cost was presented by Stich et al. [16]. They used a modified BVH build up algorithm that utilized spatial splitting similar to kd-trees in order to split large triangles causing extensive overlaps and increasing the cost. According to the observation of Goldsmith & Salmon [1] the improvement in tree cost is reflected in the traversal performance.

A similar method was proposed by Popov et al. [17]. Their kd-tree like build minimizes the overlap in BVH nodes and thus increases the performance. They also propose the assumption of non-terminating rays to be invalid and increasing the expected traversal cost of BVH.

Camera based optimization. The building of data structures using information about camera position was studied for kd-trees by Havran and Bittner [18] and for grids by Hunt and Mark [4]. The technique uses the surface area after perspective projection to the image plane instead of the world space surface area of the bounding box in the cost model. Such modified cost function leads to a more accurate estimate of the split cost for the current frame. This results in speedups at the cost of rebuilding the data structure for each camera position.

Bottom-up build. The hierarchical data structures for ray tracing are usually built in the top-down "Divide and conquer" manner. A bottom-up agglomerative clustering build up was presented by Walter et al. [19]. They showed how this approach can improve the data structures quality while keeping reasonable building times even for large data sets.

Dynamic data structures. Applications dealing with dynamic geometry are interested in both traversal and build times of data structures and different trade-offs may influence the winning strategy. Lauterbach et al. [5] presented a hybrid solution for interactive ray tracing of deformable objects. Based on a simple heuristic cheaper updating of node bounds and keeping the topology or full rebuild of the BVH is chosen in each frame. Similarly, Garanzha [6] proposed to combine cheap updates like refitting and subtree migrating with localized rebuilds to build the BVH for the current frame from the one for the previous frame. Two-level tree targeting realtime rebuilds on non-deformable models was described by Reichl et al. [20] combining BVH and kd-tree. Their shallow dynamic top level BVH assured low rebuild times while static kd-trees in the bottom level were responsible for keeping the tree quality.

Memory storage. As the complexity of scenes used in computer graphics increases, it is not always possible or viable to keep all the data in the main memory. To cope with this problem data have to be either compressed, loaded on-demand, or both. Eisemann et al. [7] decreased the memory footprint of the BVHs by expressing each node's bounding volume in only 8 bytes. They take advantage of the fact that each node shares three bounding planes with its parent. Just two bits are used per node in the scheme of Bauszat et al. [21]. Their method gets rid of children pointers by making the tree left-balanced and children bounding boxes are expressed implicitly by shrinking the parent box along the longest axis. The method offers low memory consumption at the cost of significantly worse traversal times. A hybrid method using SAH for the upper levels of the tree is proposed to mend this. BVH compression for massive models and out-of-core processing was studied by Kim et al. [8]. They focused on minimizing the hard drive access through compression of the data structure and lazy loading of only its traversed parts.

GPU build up algorithms. Parallel build up of BVHs on the GPU was investigated by Lauterbach et al. [9]. They implemented the median splitting strategy through fast radix sorting of Morton codes. To improve the quality of the final data structure, SAH based GPU build is used for lower levels of the tree.

The Hierarchical Linear BVH (HLBVH) of Pataleoni et al. [10] extends the approach of Lauterbach et al. [9] by reducing both the amount of computation and memory traffic. This is achieved by using the compress-sort-decompress approach of Garanzha and Loop [13] for sorting of primitives. They also proposed a hybrid algorithm, but unlike the previous method the SAH is used for the top levels of the hierarchy and HLBVH for the rest. Recently this build algorithm has been further optimized by Garanzha et al. [11]. They replaced the complex pipeline of the original paper with work queues as in Wald [22]. The result is faster and more memory efficient build of BVHs on the GPU. Hou et al. [23] discussed out-of-core build up of very large BVH trees on the GPU. They built the tree chunk by chunk on the GPU and assembled it on the CPU.

GPU traversal algorithms. Aila and Laine [12] studied efficiency of traversal on the GPU. A hardware aware implementation was compared to a simulator to asses its distance from the theoretical maximum. A completely different approach to GPU ray traversal algorithm was presented by Garanzha et al. [13]. They reformulated ray traversal as a sorting problem to extract ray coherence following the idea of Havran et al. in their ray shooting cache [24]. To lower the cost of sorting a compress-sort-decompress scheme is used. The authors claim speedup up to the factor of three over an optimized GPU depth-first ray tracing implementation.

Query driven approaches. Our method shares some resemblance with three other methods. First, Havran [2, Section 4.7] proposed a general cost model for building kd-trees that tries to get rid of all the unrealistic assumptions in SAH. However, this relatively complex method results in kd-trees that are not more efficient than kd-trees built with the cost model with SAH. Further, Bittner and Havran [25] studied how ray distribution information can be leveraged in building more efficient data structures but reported only mild improvements. Recently, Ize and Hansen [26] suggested to encode the knowledge whether the subtree rooted in the left child is cheaper or more expensive than the subtree rooted in the right child. This helps to compute shadow rays faster since the traversal of the shadow rays can be organized arbitrarily because finding *any occluder* along the ray path is sufficient. They report up to two times speedups.

Compared to the above discussed papers we only use the knowledge about expected triangle visibility in the context of ray tracing, that is finding the first object intersected along the ray path.

3. Background

In this section we give an overview of the cost model with SAH and the build method used in this paper. In the same manner we present the method of Stich et al. [16] which is also extended by our method.

3.1. Standard cost model with Surface Area Heuristic

The surface area cost model minimizes the traversal cost of a single random ray [1, 2]. Based on this model a Surface Area Heuristic (SAH) is developed which makes several assumptions about the properties of rays and their distribution:

1. All rays intersect the bounding box of the root node.
2. The rays are uniformly distributed.
3. None of the rays intersects an object.
4. The complexity of yet unconstructed parts of the hierarchy uses a linear estimate from the number of geometric primitives.

From these assumptions a standard cost model with SAH is derived:

$$C_{node} = C_{TS} + C_{IT}(p_L \cdot N_L + p_R \cdot N_R), \quad (1)$$

$$C_{leaf} = C_{IT} \cdot N, \quad (2)$$

where C_{TS} is the cost of performing the traversal step, C_{IT} is the average cost of ray-primitive intersection, p_L and p_R are the probabilities of traversing the left and right child respectively, N_L and N_R are the numbers of geometric primitives in the left and right child respectively, and N is the number of geometric primitives in a leaf. The terms p_L and p_R correspond to geometric probabilities of intersecting the bounding box of a child node when its parent is visited and they are computed as:

$$p_{\{L|R\}} = \frac{S_{\{L|R\}}}{S}, \quad (3)$$

where $S_{\{L|R\}}$ are the surface areas of left and right child's bounding boxes and S is the surface area of the parent's bounding box.

This cost model allows us to estimate the traversal cost of a ray in terms of the number of traversal steps and intersection tests provided the four simplifying assumptions above hold. During the top-down build the candidate split with minimal cost is found using Equation 1. The split with minimum estimated cost is then used to divide the primitives into child nodes. Simultaneously the cost estimate can be used to terminate further subdivision of the current node [2]. A leaf is created, if the cost of creating the interior node C_{node} is larger than the cost of a leaf C_{leaf}. The split made according to this method is referred to as *object split*.

BVH build

There are several BVH build methods. Our implementation of BVH building algorithm is carried out on the CPU and is based on a publicly available framework [12]. The build proceeds recursively top-down from the root node containing all the triangles (in fact the references to triangles).

For each node the best object split is found by sorting the triangles according to the centroids of their bounding boxes in each dimension. This linearly ordered array of triangles is then processed left to right and the cost is computed with Equation 1 for each possible split. The split with minimum cost is then found over all three axes.

After the best split position has been evaluated, its cost (Equation 1) is compared to the cost of a leaf (Equation 2). The current node is declared a leaf if the cost of the leaf is the smaller one and the number of triangles in the current node is less than a predefined maximum, in our case 8. Otherwise, an inner node is created and the triangles are distributed into its left and right child.

3.2. Spatial split for BVH

As observed by Ernst and Greiner [27], Dammertz and Keller [28], and Stich et al. [16], large triangles may hamper the performance of data structures built using the cost model with SAH. This is because they produce large overlaps of child nodes' bounding boxes and rays visiting this overlap may be forced to traverse the same spatial regions repetitively.

Stich et al. [16] suggest to solve this by splitting such a node with a kd-tree like split. The position of the splitting plane is chosen so that the SAH cost is minimized. This method can result in more references to a triangle which is spatially subdivided. This is called a *spatial split*. However, splitting a node with a spatial split need not be always beneficial. As references are duplicated, the SAH cost of BVH may also increase. Therefore, they solve this by choosing the split with the lower cost from spatial and object splits. Another side effect of spatial splitting is a significant increase in memory footprint and build time. The resulting data structure is denoted *Spatial BVH* (SBVH).

SBVH build

Existence of spatial splits requires changes of the building algorithm. We use the algorithm as implemented in the framework of Aila and Laine [12] which is based on a similar idea for kd-trees as the paper by Popov et al. [29]. The method uses the binning approach similar to Günther et al. [30], but instead of the centroids the left and right boundaries of the triangles in the selected axis are used. Each of the 32 bins in each dimension holds the counters of the triangle's left boundaries starting in this bin and right boundaries ending in this bin. This representation of reference spatial position allows to compute the number of triangles that span over the bins boundary.

The triangle's bounding box is then split along the bin boundaries updating each bin's bounding box with the corresponding part of its bounding box. After the binning is finished the bins are swept left to right in each dimension. The numbers of objects for the left child N_L and for the right child N_R are computed from the counters in bins. The triangles straddling the splitting plane are counted in both N_L and N_R. This allows to compute also the number of triangles split by a plane of the spatial split.

Finally a split with the minimal cost is found for spatial split and object split and the one with smaller cost is chosen to create the child nodes. If a spatial split is chosen, three new costs are computed for triangles that straddle the splitting plane. The first cost is for putting the whole triangle into the left child, the second one for putting it into the right child, and the third cost when the triangle is put into both children. The option with minimal cost is realized further improving the SAH cost of spatial splits. If a triangle straddles the splitting plane, two new tight bounding boxes are computed for its parts falling into the left and right child [31]. The termination criteria influencing the creation of leaves are the same as for the SAH method.

4. The Cost Model with Occlusion Heuristic

In ray tracing the data structure spatially organizes the regions of a scene in order to aid the ray in finding its closest intersection. Without the loss of generality, further in the text we will use only triangles as geometric primitives of a scene. The straightforward idea of our approach is that we could build more efficient data structures if we knew (or estimated well) which triangles are the closest along all the rays and which triangles are not visible. We suggest to modify the cost function so that we can compute the results for rays hitting visible triangles in a more efficient way. Nevertheless, the BVH built with our method allows to compute correctly also the results for rays hitting triangles that are not expected to be visible but with a higher cost than for the visible triangles. If the visibility assumptions are mostly valid, on average the algorithm will perform faster.

Before the computation of an image is finished, we do not know which triangles are visible. However, the proposed method is useful in two scenarios. First, when we compute the images for an animation where we yield a significant frame to frame coherence [32]. Then we can predict visible and invisible triangles with a high probability. Second, when we compute really many rays such as for path tracing, it will pay us back to get the approximate visibility information about triangles for several samples of the pixel. Then we can rebuild the data structure using this information as it is still more efficient in total (even with the time needed for rebuild). This is useful for rendering high quality images as in movie production etc.

With the information about approximate visibility of geometric primitives we create the cost function, where visible and invisible triangles are accounted for differently. We modify the cost function so that it separates potentially visible and invisible triangles in the hierarchy. The benefit of such an approach is that there are generally much fewer potentially visible triangles than invisible ones, if the scene contains *spatial regions with high occlusion*. This way the parts of the BVH tree containing potentially visible triangles are shallower. During the ray traversal for all the rays we can then compute the result in fewer traversal steps and ray object intersection tests.

To justify our approach with modified cost function we also tried to build a BVH from two subtrees, one for potentially visible and the other one for invisible triangles. The root of the resulting BVH has the BVH over potentially visible triangles as the left child and the BVH for potentially invisible triangles as the right child. Although this approach also yielded some benefit, indicating that visibility information can accelerate ray tracing, it was only marginal. In particular, on high occlusion scenes the two-subtree method improved the ray tracing performance for primary rays by less than 10% on average compared to the SAH. Our method, however, improves the performance approximately twice.

Any stack-based traversal algorithm for BVHs has two phases. In phase one, we find the first intersected object O along the ray path. In phase two, we have to also check for the ray intersection with all the triangles that could lie in front of the object O. This is usually implemented by a traversal stack

from which the nodes to be further visited are popped. Until explicitly tested the traversal algorithm does not know whether the found intersection is the closest one.

This presents another challenge; we not only have to find the closest intersection fast, but we also want to minimize the cost of traversal in phase two, after the closest intersection is found. When popping the nodes from the stack, ray-box and ray-triangle intersections in their subtrees are considered valid only if they are closer than the closest currently found ray-triangle intersection. We minimize the work in phase two by preventing nodes containing only invisible triangles to be closer than any visible triangle as detailed in the next section. Thus subtrees under these invisible nodes need not be traversed by rays hitting the visible triangles.

4.1. OSAH

In this section we describe in detail our technique that optimizes for visible triangles. Clearly, separation of visible and invisible triangles cannot be made if all the triangles in the current node are either visible or invisible. In these cases the standard cost model with SAH is used to build the subtree under the node (Equation 1). If the node contains both visible and invisible triangles, we use this modified cost model:

$$\begin{aligned} w &= 0.9 \\ p_L &= w\frac{N_L^V}{N_L^V + N_R^V} + (1.0 - w)\frac{S_L}{S} \\ p_R &= w\frac{N_R^V}{N_L^V + N_R^V} + (1.0 - w)\frac{S_R}{S} \\ C_{node} &= C_t + C_i(p_L \cdot N_L + p_R \cdot N_R), \end{aligned} \quad (4)$$

where N_L^V and N_R^V are the numbers of visible triangles in the left and right child respectively and w is the weight of importance for visible triangles. The other variables have the same meaning as in section 3.1, Equation 1 and 2. We call a split made according to Equation 4 an *OSAH split*.

The weight w is set to 0.9 to favour strongly the visible triangles. The reason for this choice is that our method targets scenes with high occlusion where visibility information can bring significant speedup. For scenes or viewpoints with less occlusion, setting the weight w to 0.5 can be more appropriate. On the average the method is not much sensitive to this weight as can be seen from graphs in section 5. Note that weight of 1.0 is not a good choice because the probability of ray-box intersection is then completely lost and cannot be used to break ties between splits with equally good visibility term.

For OSAH split it could be most advantageous if it puts all the visible references into one child and invisible references into the other child, supposed the bounding boxes of both children do not overlap. Obviously, this is not always possible, the bounding box over visible triangles might be the same as the bounding box of the whole node. Still, some candidate split with a minimum cost is chosen, even though it does not separate the child nodes well. To prevent the creation of such degenerated bounding boxes we first evaluate an additional *viability condition* before the OSAH split is chosen. The viability condition compares the best split computed with the OSAH cost model with the best split computed with the SAH cost model. The viability condition considers an OSAH split efficient only if it hides more triangles than the SAH cost model split. If the OSAH split is not viable, the split with the SAH cost model is used to create the child nodes. We use this viability condition:

$$N_I = \begin{cases} N_L & \text{if} \quad N_L^V < N_R^V \\ N_R & \text{otherwise} \end{cases}$$

$$N_I > \max(N_L^{SAH}, N_R^{SAH}), \quad (5)$$

where N_L^{SAH} and N_R^{SAH} are the numbers of triangles in the left and right child for the best split using the SAH cost model, respectively. In the most favourable case all visible triangles are put in a single child of the OSAH split and N_I thus counts only invisible triangles.

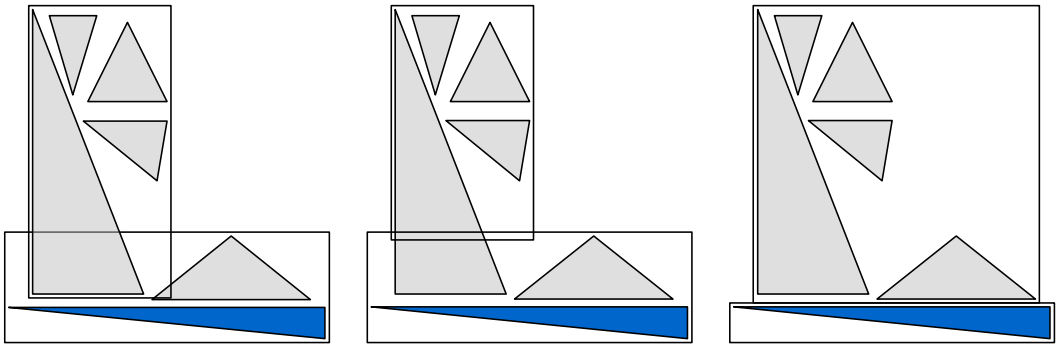
It is possible that the bounding volume of one child in the OSAH split will overlap some visible triangles from the other child as well. Then both children will have to be traversed by the rays hitting such visible triangles. To prevent this we use *spatial splits* as in the approach of Stich et al. [16]. Since we use triangle binning to compute the number of triangles to the left and right of the splitting plane, a visible triangle may be counted in both children, which puts a penalty to such a split. This helps to separate visible and invisible triangles and minimize the work done in phase two of the BVH traversal, when nodes from the traversal stack are popped to check if they can contain a closer triangle. As spatial splitting ensures that their ray-box intersection lies further along the ray than the visible triangles, such popped nodes can be discarded quickly.

We use spatial splits also for computing the SAH cost model split in the viability condition. Thus the spatial splits for the OSAH and SAH cost models can be computed at the same time. We are also able to perform a SBVH split in case the OSAH split is not viable. As the OSAH splits are meaningful only close to the root we limit their depth to $1/2 \cdot log_2(\#tris)$. Below this depth only SBVH splits are computed. To minimize the build time and memory footprint we use spatial splits only in nodes which are expected to be reached during traversal, e.g. in nodes containing visible triangles. An illustration of the splits chosen by the three BVH build algorithms is given in Figure 2.

Unlike for the SBVH split triangles straddling the splitting plane cannot be chosen to be put entirely into the invisible child even though this might lower the cost of such created child nodes. This is because this insertion might cause visible triangles to be overlapped by the enlarged invisible child's bounding box, which is what we are trying to avoid as explained above. On the other hand, these triangles can be freely put into the visible child as this will not cause visible triangles to be overlapped and overlapping of invisible triangles should not influence the traversal performance. Identically to the SAH and SBVH a leaf is created if its cost is smaller than the cost of splitting the node and the number of triangles in the node is less than the predefined maximum constant.

4.2. Modification for particular traversal algorithm

Finally, we modify the way child nodes are added to the hierarchy which is connected to the used traversal algorithm. In



(a) SAH (b) SBVH (c) OSAH

Figure 2: Visualization of different split types for BVH: (a) SAH, (b) SBVH, and (c) OSAH. Blue triangles are visible, light grey triangles are invisible. The black rectangles represent bounding boxes and have been slightly enlarged for clarity. Notice how OSAH can produce large bounding boxes but these are not traversed if the method's assumptions are met.

this traversal algorithm the left child is chosen to be traversed first, if bounding boxes of both children have the same distance along a ray. We can take advantage of this by making the child with more visible triangles the left child. Often, all the visible triangles are put in only one child and the nodes of the hierarchy with visible triangles are traversed before those with invisible triangles. Thus, the closest triangles are discovered faster and the traversal takes fewer steps. This feature is particularly beneficial for large axis-aligned occluding triangles (like walls) which are often overlapped by both child node's bounding boxes and the better traversal order cannot be deduced from the distances themselves.

5. Results

We have implemented the proposed method in the software framework accompanying the previous work of Aila and Laine [12] that utilizes CUDA, highly parallel computer architecture from NVIDIA. The ray tracer implementation builds the BVH on the CPU, the data structure is then transferred to the GPU, and the ray traversal code and image computation runs entirely on the GPU. We have extended the framework by path tracing, shooting 1,000 samples per pixel in our tests. For testing we have used a PC with Intel Core i7-2600, 16GB of RAM and NVIDIA GeForce GTX 580 running 64-bit version of Windows 7. All images in this paper were rendered at the resolution of 1024×768 pixels.

5.1. Reference method ABVH and VSAH

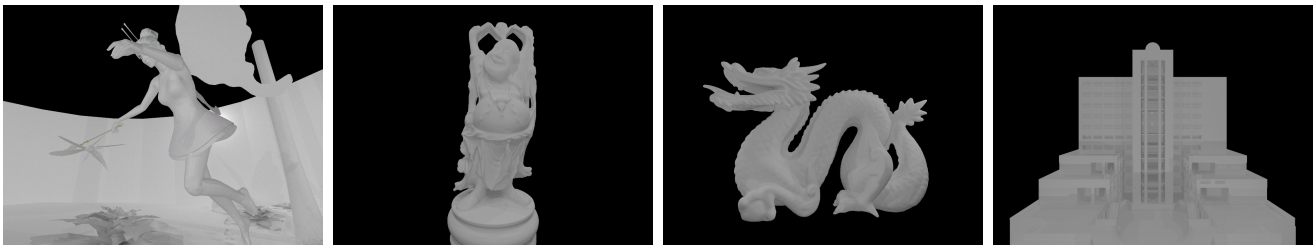
Our method for building BVH described in section 4.1, denoted OSAH, is compared with several previously known methods for building BVH, namely the standard BVH with the cost function with SAH as in Wald [22] (denoted SAH) and Spatial BVH as described by Stich et al. [16] (denoted SBVH). We also show results for two other reference methods denoted as ABVH and VSAH to make the comparison fair and also to show the potential of using the visibility information for building hierarchies.

The first new reference, the ABVH method, improves on the SBVH [16]. It computes the SBVH splits only in nodes containing visible triangles. It builds the visible parts of the tree with the SBVH while invisible parts are built with the standard SAH. The ray tracing performance of the method is nearly the same as for the SBVH but obviously it has almost always a lower build time. This method is a beneficial approximation to the full SBVH build using the visibility information, and it was not published in the former literature. It also enables us to show the difference between the utilization of spatial splits in the OSAH and the SBVH/ABVH. We want to remark that the ABVH build can be slower than the SBVH when many triangles are visible and the BVH for the two methods is very similar. Then the overhead of keeping the visibility information can slow down the build slightly. In our measurements this happened only for some cameras tested with path tracing (the increase of the time for building was 3% at most).

The other reference, the VSAH method, utilizes the visibility information more directly. It restricts the BVH building only over the visible triangles. This assumes exact knowledge of visibility, which is not possible by sampling, but it is still useful when taken as a reference. This method provides some upper bound for the performance improvement achievable using the visibility information alone. It also provides some lower bound on the construction time and memory usage. Because it is based on SAH and does not use spatial splits, it need not be the fastest of all of the compared methods, especially when many triangles are visible. We want to repeat here that this method is not practical as it implies that we know the exact triangle visibility beforehand. Note that for comparing all the methods the information about the visibility is precomputed for the same set of rays unless mentioned otherwise, i.e. it is exact.

5.2. Test Scenes

We report most of our results on three architectural models (Sibenik Cathedral, Sponza, and Soda Hall) and their alternative representation given by rotating the data by 45 degrees (as also used by Stich et al. [16]). These models and the Power



(a) Fairy Forest (b) Happy Buddha (c) Dragon (d) Mba1

Figure 3: Four low occlusion scenes used for measurements rendered with ambient occlusion.

Scene	#Tris	BVH Size [MB]				
		SAH	SBVH	ABVH	OSAH	VSAH
high occlusion						
Sibenik	80k	6.46	+22%	+18%	+18%	-50%
Sibenik_rot45z	80k	6.41	+32%	+28%	+26%	-49%
Sponza	76k	6.10	+30%	+24%	+22%	-26%
Sponza_rot45y	76k	5.88	+102%	+79%	+77%	-26%
Soda Hall	2.2M	170.91	+12%	+6%	+6%	-95%
Soda Hall_rot45z	2.2M	170.83	+25%	+12%	+11%	-95%
Powerplant	12.7M	1326.80	+19%	+19%	+18%	-95%
Powerplant_rot45z	12.7M	1278.34	+34%	+32%	+33%	-95%
low occlusion						
Fairy Forest	174k	14.00	+6%	+6%	+6%	-19%
Happy Buddha	1088k	90.81	+7%	+6%	+7%	-51%
Dragon	871k	72.84	+4%	+4%	+5%	-43%
Mba1	84k	6.54	+69%	+56%	+54%	-63%

Table 1: Statistics for all of our test scenes: number of triangles and average BVH memory consumption for five compared BVH building methods. The BVH sizes for ABVH, OSAH and VSAH are for path tracing where the number of visible triangles is the highest over all of our measured ray distributions.

Plant model represent scenes with high occlusion. Rendered images used for testing these scenes are in Figure 8. In addition to these highly occluded scenes, we have also tested our method on four other scenes, depicted in Figure 3. These four scenes (Fairy Forest, Happy Buddha, Dragon, and Mba1) feature little occlusion and therefore their potential for improvement using our method is small. Nevertheless, we show that our method performs similarly well as the reference method on these scenes.

The properties of all the scenes and the memory consumption of built BVHs are reported in Table 1. The visibility based methods have higher memory consumption than the method with standard SAH but the memory increase for the OSAH is almost always lower than for the SBVH. There are two reasons for this behaviour. First, only a small number of OSAH splits close to the root of the tree is often enough to separate the visible and invisible triangles. Second, there is much less visible geometry than invisible geometry. In our measurements between 1/1000 and 1/100 of the nodes were split with an OSAH split. Also the data given in Table 1 are for path tracing which has the most visible triangles of all the distributions and thus the highest BVH size for the OSAH and the VSAH.

5.3. Path Tracing

The summary results for path tracing, averaged over all cameras/viewpoints, are shown in Table 2. For each scene and method we report the average number of triangle intersection tests per ray N_{IT}, average number of traversal steps N_{TS}, the performance in $Mrays/s$, and the time to build the BVH in seconds. The measured build times for the visibility driven heuristics (ABVH, OSAH, and VSAH) include the GPU to CPU transfer of triangle visibility information. The time for allocation of visibility array on the CPU and the GPU/CPU transfer is insignificant compared to the time needed to build the BVH; it takes about 25.5ms for the Power Plant model. We represent triangle visibility in an integer array, if the transfer time have become a bottleneck we could lower the overhead by representing visibility as bits. The SAH is used as the reference method in our tables (i.e. the reference method has ratio +0%).

Detailed results for path tracing on the highly occluded scenes are given in Tables 3 and 4. For each scene four cameras (i.e. viewpoints) were chosen. The first viewpoint is an outside view of a detail of the model. The second one is an outside view of a complex geometry. The third and fourth viewpoints are interior views of the detailed and complex geometry. The images computed for these selected viewpoints are shown in Figure 8.

5.4. Discussion

As can be seen from the tables, for path tracing the BVHs built with the OSAH perform always faster than BVHs built with the reference SAH. The BVHs built with the OSAH are almost always faster even than the ones build with the SBVH. The speedup is obviously higher on scenes with high occlusion and on cameras located outside the scene. On cameras located inside the scene majority of rays traverse through the empty space (the invisible part of the BVH) and thus our method cannot help here. Also for the rotated scenes it is much harder to separate visible and invisible triangles and the speedup is usually low compared to SBVH. However, the performance of the SBVH comes at the cost of significantly increased build times, which are much less pronounced in our method. For the scenes with high occlusion, the OSAH method's average performance is 67% higher compared to the SAH and 17% with respect to the SBVH for path tracing. Also the memory consumption is a little lower than for the SBVH.

We have also studied the dependence of the method on the weight w used in Equation 4. This is shown for primary rays

Scene	N_{IT}					N_{TS}					Performance [Mrays/s]					T_{build}[s]				
	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH
Sibenik	0.84	-10%	-10%	-17%	+1%	9.13	-8%	-8%	-10%	-2%	88.92	+4%	+3%	+22%	+4%	0.61	+492%	+352%	+330%	-48%
Sibenik_rot45z	2.37	-29%	-29%	-30%	-3%	11.39	-14%	-14%	-13%	-0%	51.78	+20%	+19%	+28%	+3%	0.62	+534%	+416%	+387%	-47%
Sponza	1.34	-29%	-30%	-20%	-7%	12.09	-13%	-14%	-24%	-16%	348.64	+6%	+6%	+46%	+83%	0.58	+566%	+436%	+424%	-21%
Sponza_rot45y	5.80	-64%	-65%	-66%	-16%	19.64	-41%	-41%	-42%	-22%	142.97	+141%	+139%	+153%	+217%	0.57	+796%	+623%	+596%	-19%
Soda Hall	1.25	-25%	-26%	-26%	-2%	11.84	-4%	-4%	-16%	-6%	61.42	+15%	+14%	+20%	+4%	26.07	+85%	+48%	+22%	-96%
Soda Hall_rot45z	5.60	-66%	-66%	-66%	+9%	19.23	-38%	-38%	-41%	-8%	25.66	+94%	+93%	+93%	+8%	25.95	+119%	+61%	+33%	-96%
Powerplant	4.18	-68%	-69%	-69%	-30%	24.93	-44%	-44%	-48%	-12%	23.73	+121%	+123%	+129%	+19%	196.07	+38%	+42%	+27%	-96%
Powerplant_rot45z	23.52	-81%	-81%	-81%	-33%	37.41	-57%	-58%	-59%	-18%	7.11	+316%	+320%	+325%	+65%	190.20	+78%	+80%	+69%	-96%
Fairy Forest	2.32	-0%	-1%	-16%	-24%	12.66	-1%	-1%	-4%	+2%	44.28	+2%	+2%	+9%	+5%	1.58	+138%	+139%	+141%	-15%
Happy Buddha	0.37	-3%	-3%	-19%	+0%	5.19	-2%	-2%	-5%	-3%	68.69	+2%	+2%	+8%	+1%	11.08	+170%	+159%	+136%	-51%
Dragon	0.42	+0%	+0%	-12%	+0%	5.55	-1%	-1%	-2%	-0%	72.21	+1%	+1%	+6%	+1%	8.55	+182%	+175%	+157%	-42%
Mba1	0.82	-30%	-30%	-34%	-7%	8.72	-27%	-27%	-29%	-6%	40.05	+35%	+35%	+39%	+6%	0.60	+763%	+568%	+543%	-63%
total average	4.07	-64%	-64%	-65%	-21%	14.81	-31%	-31%	-34%	-11%	81.29	+35%	+34%	+55%	+64%	38.54	+72%	+67%	+53%	-93%
ratio average	+0%	-34%	-34%	-38%	-9%	+0%	-21%	-21%	-24%	-8%	+0%	+63%	+63%	+73%	+35%	+0%	+330%	+258%	+239%	-57%

Table 2: Summary results for path tracing and five BVH build methods, the new method is denoted OSAH. The average values over all the cameras are reported on the tested scenes; the number of ray-triangle intersection tests, the number of traversal steps, performance in Mrays/s, and the time to build the BVH are reported. SAH is used as the reference method (+0%). These data were measured for 1,000 samples per pixel. The total average is computed from all scenes and viewpoints as a mean value, the ratio average is computed from the averages over the viewpoints shown in this table for all the scenes.

Scene /	standard scene data (not rotated)																			
	N_{IT}					N_{TS}					Performance [Mrays/s]					T_{build} [s]				
camera	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH
Sibenik 0	0.26	-35%	-35%	-27%	+4%	3.90	-1%	-1%	-12%	-7%	217.62	+2%	+1%	+27%	+5%	0.62	+487%	+165%	+121%	-97%
Sibenik 1	0.30	-7%	-7%	-20%	-3%	4.05	-3%	-2%	-7%	-2%	107.39	+5%	+4%	+14%	+3%	0.61	+490%	+246%	+220%	-90%
Sibenik 2	1.42	-6%	-6%	-11%	+1%	13.22	-11%	-11%	-11%	-1%	13.70	+12%	+12%	+14%	+1%	0.61	+490%	+497%	+485%	-3%
Sibenik 3	1.39	-12%	-12%	-22%	+0%	15.36	-9%	-8%	-9%	-1%	16.96	+14%	+13%	+18%	+1%	0.62	+481%	+487%	+477%	-3%
average	0.84	-10%	-10%	-17%	+1%	9.13	-8%	-8%	-10%	-2%	88.92	+4%	+3%	+22%	+4%	0.61	+492%	+352%	+330%	-48%
Sponza 0	1.49	-11%	-12%	+37%	-11%	9.48	-5%	-6%	-58%	-79%	1298.70	+5%	+5%	+48%	+89%	0.58	+566%	+31%	+9%	-99%
Sponza 1	0.71	-37%	-38%	-41%	-15%	6.75	-19%	-20%	-21%	-1%	91.88	+24%	+23%	+30%	+2%	0.58	+566%	+571%	+560%	+5%
Sponza 2	1.67	-40%	-40%	-45%	-5%	15.95	-12%	-12%	-13%	+0%	2.59	+20%	+20%	+24%	+1%	0.58	+564%	+571%	+567%	+5%
Sponza 3	1.48	-32%	-32%	-39%	-3%	16.20	-17%	-17%	-18%	-1%	1.39	+21%	+20%	+22%	+2%	0.58	+566%	+571%	+562%	+5%
average	1.34	-29%	-30%	-20%	-7%	12.09	-13%	-14%	-24%	-16%	348.64	+6%	+6%	+46%	+83%	0.58	+566%	+436%	+424%	-21%
Soda Hall 0	0.45	-13%	-13%	-13%	+7%	9.06	-12%	-12%	-22%	-12%	79.30	+15%	+14%	+22%	+6%	26.08	+85%	+47%	+21%	-96%
Soda Hall 1	0.22	-27%	-27%	-27%	-5%	5.10	-10%	-10%	-15%	-8%	102.92	+13%	+12%	+16%	+7%	26.07	+85%	+64%	+37%	-93%
Soda Hall 2	2.62	-16%	-15%	-11%	+6%	18.66	+11%	+11%	-9%	-10%	7.94	-12%	-12%	+14%	+2%	26.08	+85%	+28%	+8%	-99%
Soda Hall 3	1.70	-41%	-45%	-50%	-16%	14.54	-15%	-16%	-21%	+4%	55.51	+22%	+23%	+27%	-6%	26.07	+85%	+51%	+21%	-96%
average	1.25	-25%	-26%	-26%	-2%	11.84	-4%	-4%	-16%	-6%	61.42	+15%	+14%	+20%	+4%	26.07	+85%	+48%	+22%	-96%
Powerplant 0	2.95	-60%	-60%	-61%	-6%	12.70	-23%	-23%	-26%	-7%	32.76	+81%	+81%	+86%	+17%	196.06	+38%	+43%	+28%	-95%
Powerplant 1	1.40	-64%	-64%	-64%	-1%	8.35	-27%	-27%	-27%	-6%	32.24	+103%	+102%	+103%	+20%	196.07	+38%	+43%	+25%	-96%
Powerplant 2	4.60	-65%	-65%	-64%	-12%	37.30	-44%	-45%	-53%	-6%	14.98	+189%	+195%	+224%	+10%	196.08	+38%	+40%	+23%	-97%
Powerplant 3	7.76	-75%	-75%	-75%	-56%	41.37	-52%	-53%	-53%	-21%	14.92	+178%	+185%	+185%	+28%	196.07	+38%	+43%	+32%	-96%
average	4.18	-68%	-69%	-69%	-30%	24.93	-44%	-44%	-48%	-12%	23.73	+121%	+123%	+129%	+19%	196.07	+38%	+42%	+27%	-96%
total average	1.90	-48%	-48%	-47%	-18%	14.50	-24%	-24%	-30%	-10%	130.68	+12%	+12%	+43%	+57%	55.83	+46%	+44%	+28%	-96%
ratio average	+0%	-33%	-33%	-33%	-10%	+0%	-17%	-17%	-24%	-9%	+0%	+36%	+37%	+54%	+27%	+0%	+295%	+220%	+201%	-65%

Table 3: Comparison of methods for highly occluded scenes with four cameras each (shown in Figure 8). The legend is the same as in Table 2.

Scene /	rotated version of scene data (Sibenik and Soda Hall 45^o in Z-axis, Sponza and Powerplant 45^o in Y-axis)																			
	N_{IT}					N_{TS}					Performance [Mrays/s]					T_{build} [s]				
camera	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH	SAH	SBVH	ABVH	OSAH	VSAH
Sibenik 0	0.87	-38%	-39%	-28%	-1%	4.96	-11%	-11%	-8%	-4%	120.35	+19%	+18%	+29%	+3%	0.62	+534%	+266%	+234%	-95%
Sibenik 1	0.82	-24%	-24%	-29%	-1%	5.14	-13%	-13%	-14%	-5%	68.76	+18%	+18%	+25%	+3%	0.63	+524%	+313%	+256%	-89%
Sibenik 2	4.27	-33%	-33%	-33%	-5%	15.58	-15%	-15%	-13%	+1%	8.19	+28%	+28%	+26%	+0%	0.62	+534%	+539%	+524%	-2%
Sibenik 3	3.52	-22%	-23%	-26%	+0%	19.88	-15%	-15%	-14%	+1%	9.81	+26%	+26%	+27%	-1%	0.63	+524%	+530%	+517%	-3%
average	2.37	-29%	-29%	-30%	-3%	11.39	-14%	-14%	-13%	-0%	51.78	+20%	+19%	+28%	+3%	0.62	+534%	+416%	+387%	-47%
Sponza 0	5.06	-68%	-68%	-66%	-58%	20.19	-56%	-55%	-60%	-80%	540.79	+141%	+139%	+153%	+230%	0.57	+798%	+75%	+16%	-99%
Sponza 1	5.56	-80%	-80%	-81%	-4%	11.55	-43%	-43%	-43%	-3%	29.51	+142%	+141%	+147%	+1%	0.57	+796%	+807%	+789%	+7%
Sponza 2	4.92	-37%	-37%	-41%	-3%	21.68	-29%	-29%	-30%	+1%	1.11	+80%	+79%	+84%	-1%	0.57	+796%	+804%	+795%	+7%
Sponza 3	7.65	-68%	-69%	-70%	-5%	25.12	-38%	-38%	-37%	-4%	0.47	+130%	+130%	+126%	+4%	0.57	+796%	+804%	+786%	+7%
average	5.80	-64%	-65%	-66%	-16%	19.64	-41%	-41%	-42%	-22%	142.97	+141%	+139%	+153%	+217%	0.57	+796%	+623%	+596%	-19%
Soda Hall 0	4.03	-76%	-76%	-73%	+0%	11.71	-30%	-30%	-29%	-6%	34.00	+94%	+93%	+86%	+6%	25.96	+119%	+65%	+34%	-96%
Soda Hall 1	2.27	-78%	-78%	-77%	-8%	7.69	-34%	-34%	-35%	-11%	44.98	+86%	+85%	+88%	+10%	25.95	+119%	+81%	+51%	-93%
Soda Hall 2	11.39	-62%	-63%	-65%	+15%	34.38	-37%	-38%	-45%	-11%	2.53	+111%	+114%	+165%	+9%	25.95	+119%	+32%	+10%	-99%
Soda Hall 3	4.69	-58%	-58%	-59%	+13%	23.16	-43%	-43%	-43%	-5%	21.13	+108%	+108%	+107%	+7%	25.95	+119%	+65%	+38%	-96%
average	5.60	-66%	-66%	-66%	+9%	19.23	-38%	-38%	-41%	-8%	25.66	+94%	+93%	+93%	+8%	25.95	+119%	+61%	+33%	-96%
Powerplant 0	11.40	-72%	-72%	-72%	-6%	21.14	-39%	-39%	-39%	-6%	8.83	+277%	+275%	+278%	+73%	190.20	+78%	+82%	+72%	-95%
Powerplant 1	6.55	-78%	-78%	-78%	-17%	13.35	-46%	-46%	-46%	-14%	8.87	+326%	+323%	+323%	+71%	190.21	+78%	+81%	+68%	-96%
Powerplant 2	45.77	-85%	-85%	-85%	-59%	62.89	-65%	-65%	-67%	-32%	4.49	+457%	+459%	+480%	+105%	190.20	+78%	+75%	+70%	-97%
Powerplant 3	30.35	-79%	-79%	-79%	-6%	52.27	-57%	-60%	-60%	-7%	6.24	+255%	+279%	+285%	+18%	190.20	+78%	+81%	+67%	-96%
average	23.52	-81%	-81%	-81%	-33%	37.41	-57%	-58%	-59%	-18%	7.11	+316%	+320%	+325%	+65%	190.20	+78%	+80%	+69%	-96%
total average	9.32	-73%	-73%	-73%	-22%	21.92	-44%	-44%	-45%	-14%	56.88	+113%	+112%	+123%	+140%	54.33	+86%	+80%	+67%	-96%
ratio average	+0%	-60%	-60%	-61%	-10%	+0%	-37%	-38%	-39%	-12%	+0%	+142%	+143%	+150%	+73%	+0%	+382%	+295%	+272%	-65%

Table 4: Comparison of methods for rotated variants of highly occluded scenes with the same viewpoints as in Table 3.

in graphs in Figure 4. The results show that there is not much dependence of the weight on average. We have set $w = 0.9$ for all other measurements to decrease the number of reported results.

5.5. Animation

In order to demonstrate that the method is beneficial even when using only estimated triangle visibility from the previous frame, we have also prepared a hundred-frame-long fly-through animation sequences. The camera looks at a single point in a scene while it is moving around the scene. The performance for the animations is summarized in graphs in Figure 5 for the three test scenes and their rotated variants.

Similarly we have prepared generic animations where 50 to 100 boxes are randomly placed inside the scene's bounding box and the vertices inside these boxes are randomly moved. [2] The results can be seen in Figure 6. In these walkthroughs and animations the performance of the OSAH method was very similar when the BVH used the visibility information from the current frame (i.e. exact visibility) and from the previous frame (i.e. only approximate visibility). This indicates that our method can be used with success when this approximate visibility assumption holds, irrespective if the scene is only a walkthrough or a fully animated scene (such as in movie production).

5.6. Different Ray Distributions

Finally, we present the average performance for each of the architectural models, measured for several ray distributions. The results are in graph in Figure 7. Note that for the shadow and ambient occlusion rays the hit triangle need not be the closest one along the ray. The visible triangles are thus not spatially localized, which is more challenging for our method. Nevertheless, even with these ray distributions our method performs better than the other methods. The graphs indicate that the performance is largely independent of the ray distribution imposed by rendering (not for random ones) and gracefully degrades to the performance of SBVH/ABVH as more geometry becomes visible.

Another important comparison of the data structure build methods is through the average speedups, computed as an arithmetic mean of the speedups for each scene and viewpoint. Table 5 shows the average ray tracing speedup and average increase in build times over the reference SAH method for the SBVH and our OSAH build methods. The comparison is taken for each of the measured ray distributions for highly occluded scenes.

Distribution of rays	Ray tracing speedup [%]		Build time increase [%]	
	SBVH	OSAH	SBVH	OSAH
primary	+87%	+102%	+331%	+62%
shadow	+62%	+73%	+340%	+123%
ambient occlusion	+46%	+59%	+326%	+107%
diffuse	+83%	+101%	+328%	+171%
path tracing	+89%	+102%	+338%	+236%

Table 5: Results summary: average speedups for ray tracing, and the increase of building time for the SBVH and OSAH methods for different ray distributions, where SAH is taken as a reference method. The values are computed over all high occlusion scenes (Sibenik, Sponza, Soda Hall, Powerplant and their rotated counterparts).

6. Conclusion and Future Work

We have introduced a new concept for improving the performance of the data structures that relies on the approximate visibility information over scene primitives. For this concept we have presented a novel technique for building higher quality data structures, bounding volume hierarchies (BVHs), by modifying the cost model. The BVHs built up using the proposed cost model speed up path tracing by 102% on average on highly occluded scenes compared to the SAH and 10% compared to SBVH. Although this is not a very high factor, this improvement is significant as we improve the constant behind already very efficient algorithm and not the asymptotic (i.e. logarithmic) behaviour of the algorithm. The proposed method is capable of improving the performance for other ray distributions as well with similar or better results. The technique is particularly beneficial for outside views of possibly deforming massive models with high occlusion.

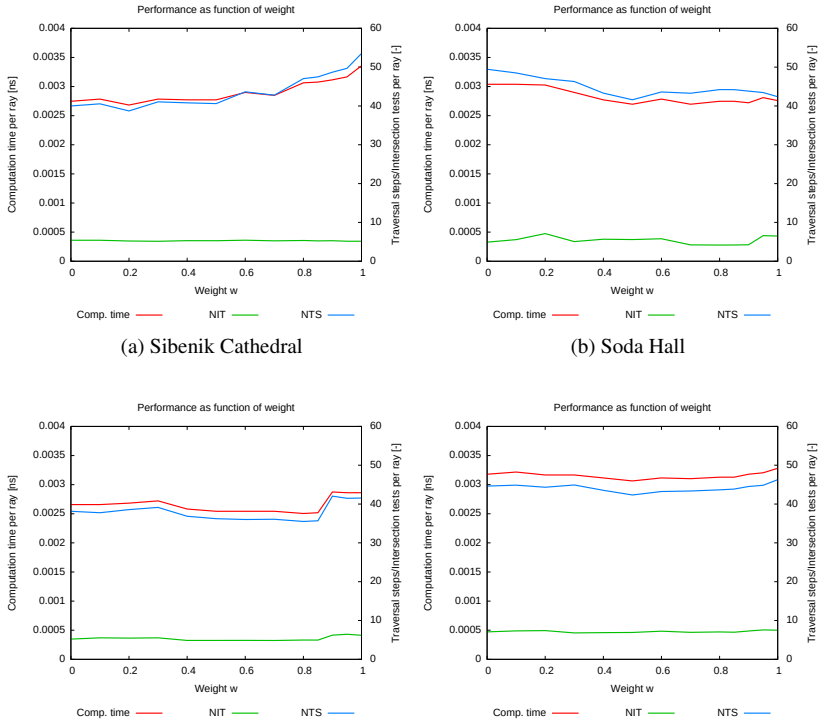
For future work there are several possibilities of using the visibility information in building algorithm for hierarchical data structures. Foremost we would like to implement the same cost model for building up kd-trees to judge the benefit or penalty imposed by front-to-back traversal. More sophisticated weight, cost functions, and viability conditions may further increase performance and decrease memory footprint and build times. Another perspective direction is to study lazy partial rebuilds of BVH with the OSAH like method for the animated scenes. This will require development of a heuristic evaluating the changes in visibility inside each node's subtree and deciding whether the subtree should be rebuilt or not.

Acknowledgements

We would like to thank the authors of the modelled scenes used in this work: Marko Dabrovic (http://www.rna.hr) for the Sibenik Cathedral and Sponza models, and Prof. Carlo Séquin for the Soda Hall model, UTAH animation repository for providing Fairy Forest scene, and Stanford repository for the two models Happy Buddha and Dragon. We would also like to thank Tero Karras, Timo Aila, and Samuli Laine for releasing their ray tracing framework.

This work was supported by Ministry of Education of The Czech Republic under research programs LC06008 and MSM 6840770014, further by the Grant Agency of the Czech Republic under research programs P202/10/1435, P202/12/2413, and P202/11/1883, and the Grant Agency of the Czech Technical University in Prague, grant No. SGS10/289/OHK3/3T/13.

[2] There are accompanying videos for the walkthroughs and animations for this submission.



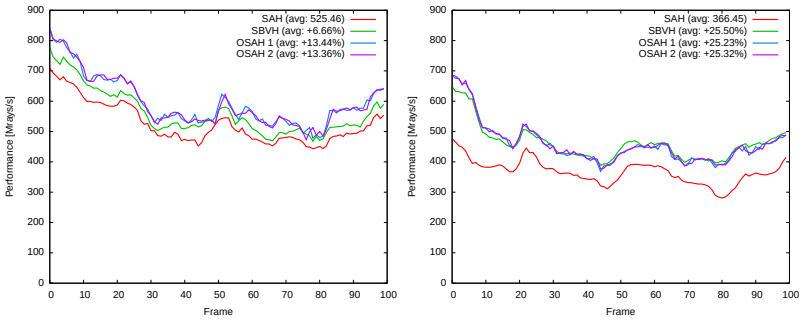
(a) Sibenik Cathedral

(b) Soda Hall

(c) Sponza

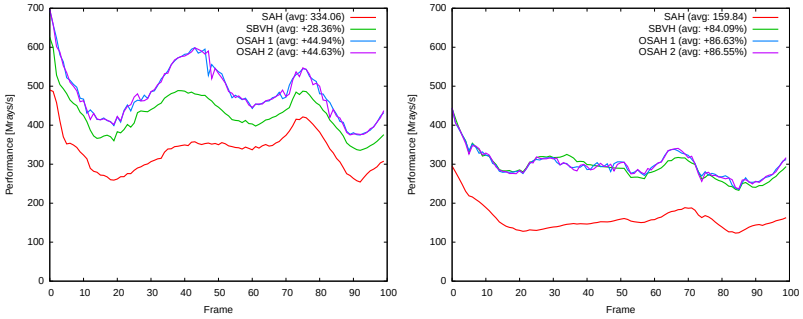
(d) Average

Figure 4: The performance of BVH built with the cost function with OSAH as function of weight in Equation 4, (a) Sibenik Cathedral, (b) Soda Hall, (c) Sponza, (d) average for these scenes and their rotated variants. The graphs show the average computation time, the number of traversal steps and intersection tests per ray for primary rays.



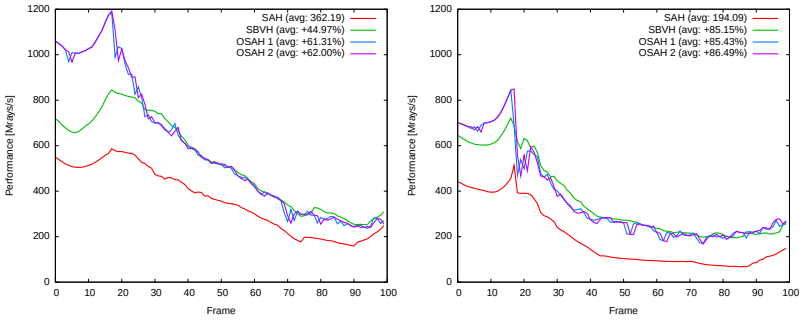
(a) Sibenik Cathedral

(b) Sibenik Cathedral rotated



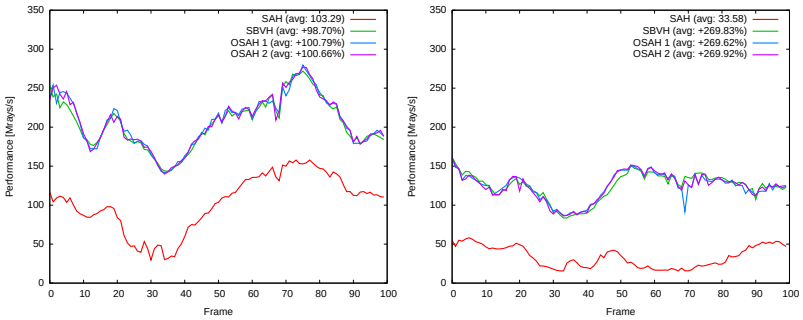
(c) Soda Hall

(d) Soda Hall rotated



(e) Sponza

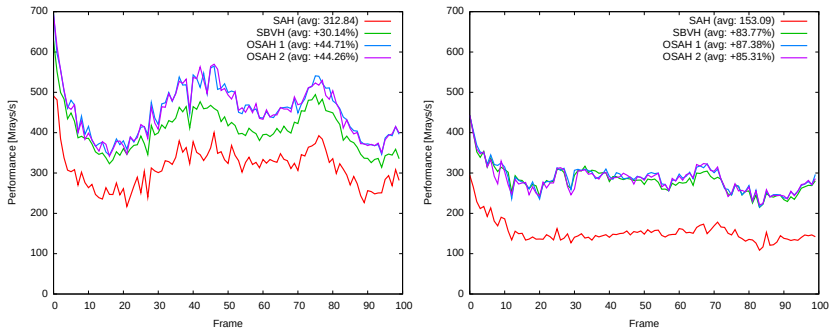
(f) Sponza rotated



(g) Powerplant

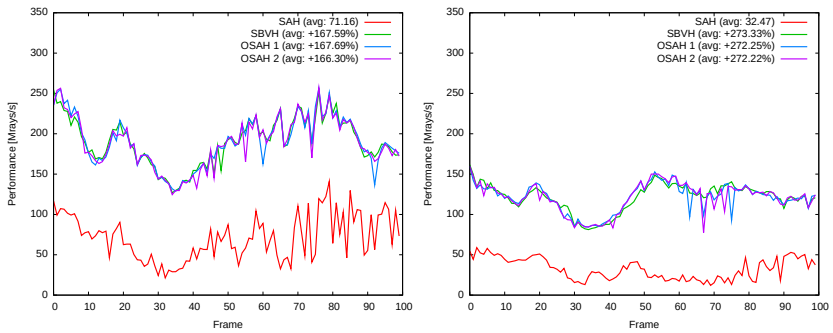
(h) Powerplant rotated

Figure 5: The performance of ray tracing for animation walkthrough with 100 frames using BVH built with SAH, SBVH and OSAH for (a) Sibenik Cathedral, (b) Sibenik Cathedral rotated 45 degrees in z-axis, (c) Soda Hall, (d) Soda Hall rotated by 45 degrees in z-axis, (e) Sponza, (f) Sponza rotated by 45 degrees in y-axis, (g) Powerplant, (h) Powerplant rotated by 45 degrees in y-axis. The method "OSAH 1" uses visibility information for the current frame while the method "OSAH 2" uses visibility information from the previous frame to build the BVH. Notice that the curves for "OSAH 1" and "OSAH 2" are almost identical, which shows that the approximate visibility from the previous frame can be used while preserving the performance. Average performance in Mrays/s over the entire length of animation is reported in brackets absolutely or relatively to the SAH method.



(a) Soda Hall

(b) Soda Hall rotated



(c) Powerplant

(d) Powerplant rotated

Figure 6: The performance of ray tracing for the animation of 100 frames with the deformable scenes made by the pertubation of a static scene. The legend is the same as in Figure 5. (a) Soda Hall, (b) Soda Hall rotated by 45 degrees in z-axis, (c) Powerplant, (d) Powerplant rotated by 45 degrees in y-axis.

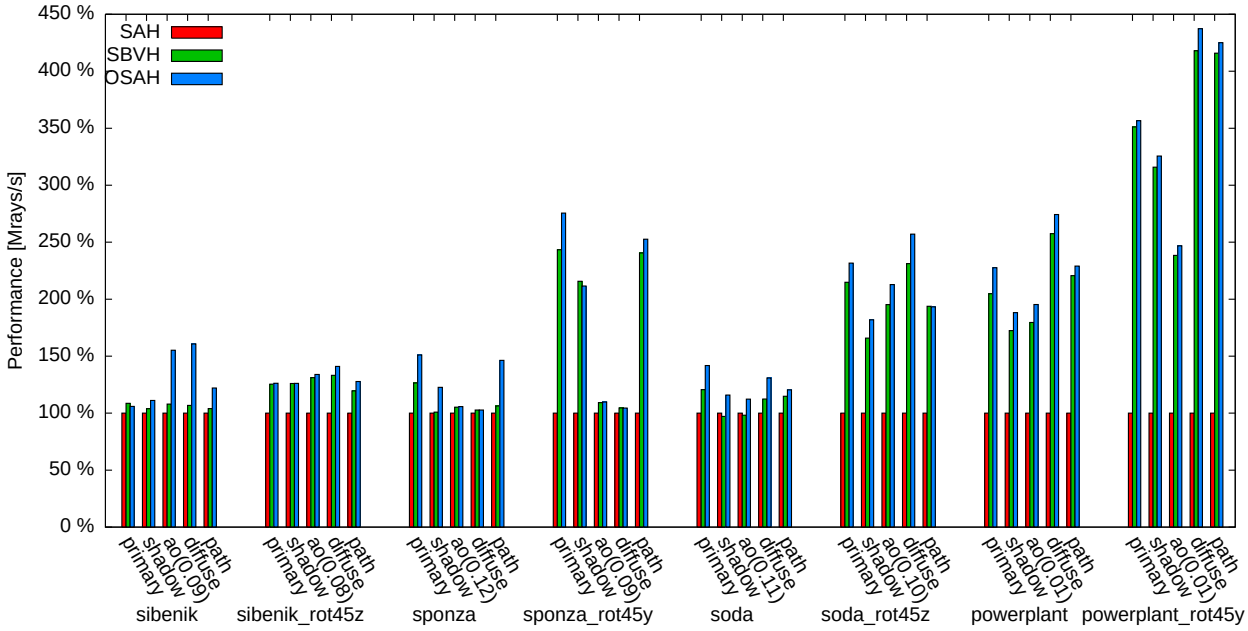
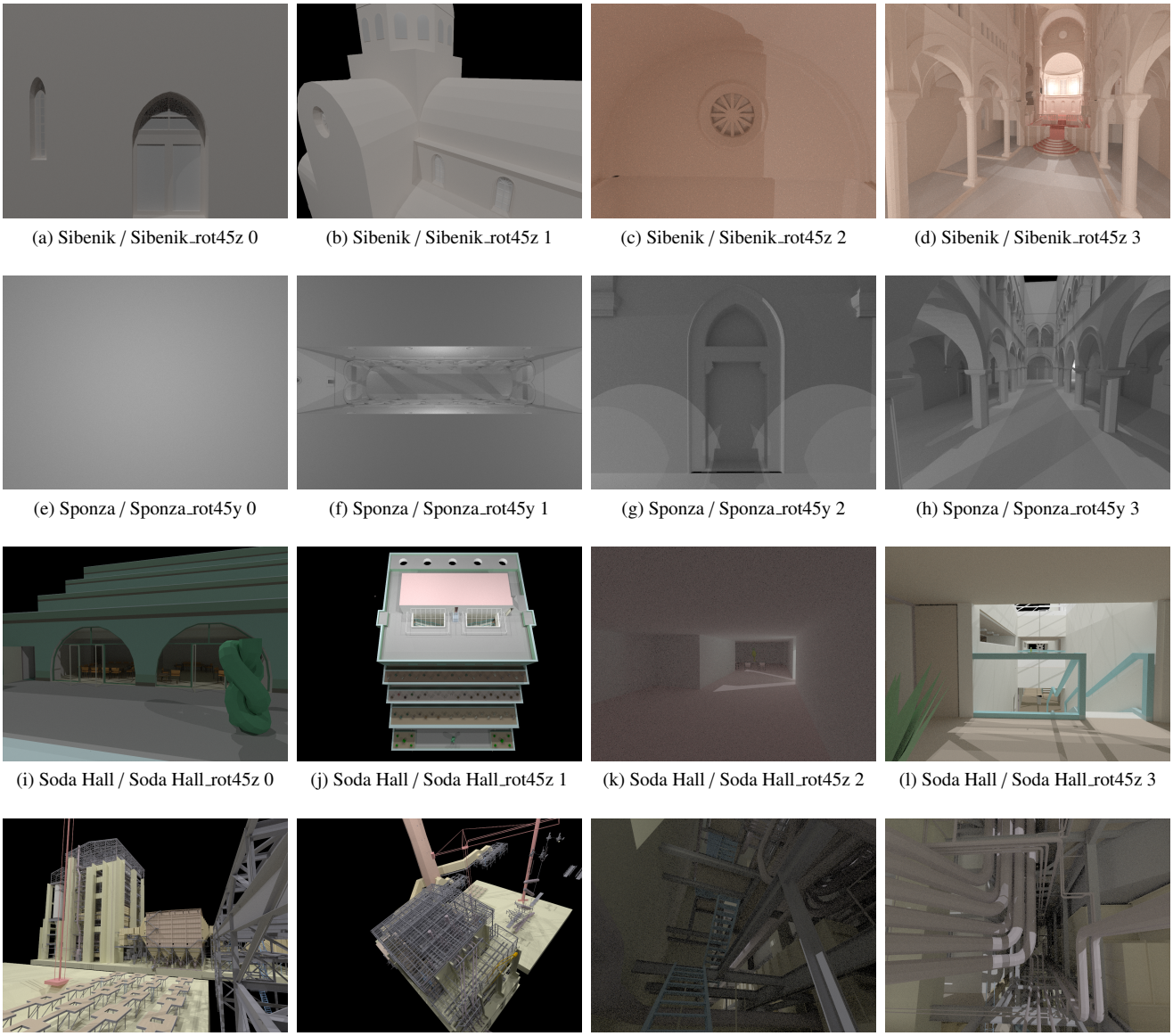


Figure 7: Relative performance in Mrays/s of our method (OSAH) and two reference methods (SAH and SBVH) in dependence of ray distribution. The cameras from Figure 8 were used for the measurements. The graphs are normalized to the performance of the reference SAH method (100%). *primary* distribution stands for camera rays. *shadow* for occlusion from 10 point light sources (5 outside, 5 inside). *ao* stands for ambient occlusion (numbers in brackets are ray lengths as ratios to scene diagonal), *diffuse* for one bounce diffuse interreflection. *path* stands for path tracing with 1,000 samples used per pixel. For *ao* and *diffuse* 32 rays were used to sample the hemisphere above hit point.



(a) Sibenik / Sibenik_rot45z 0 (b) Sibenik / Sibenik_rot45z 1 (c) Sibenik / Sibenik_rot45z 2 (d) Sibenik / Sibenik_rot45z 3

(e) Sponza / Sponza_rot45y 0 (f) Sponza / Sponza_rot45y 1 (g) Sponza / Sponza_rot45y 2 (h) Sponza / Sponza_rot45y 3

(i) Soda Hall / Soda Hall_rot45z 0 (j) Soda Hall / Soda Hall_rot45z 1 (k) Soda Hall / Soda Hall_rot45z 2 (l) Soda Hall / Soda Hall_rot45z 3

(m) Powerplant / Powerplant_rot45y 0 (n) Powerplant / Powerplant_rot45y 1 (o) Powerplant / Powerplant_rot45y 2 (p) Powerplant / Powerplant_rot45y 3

Figure 8: Cameras-viewpoints used for our measurements for the highly occluded scenes: Sibenik, Sponza, Soda Hall, and Powerplant.

References

[1] J. Goldsmith, J. Salmon, Automatic Creation of Object Hierarchies for Ray Tracing, IEEE Computer Graphics and Applications 7 (5) (1987) 14–20.

[2] V. Havran, Heuristic Ray Shooting Algorithms, Ph.D. Thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague (November 2000).

[3] A. Kensler, Tree Rotations for Improving Bounding Volume Hierarchies, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 73–76.

[4] W. Hunt, W. Mark, Ray-Specialized Acceleration Structures for Ray Tracing, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 3–10.

[5] C. Lauterbach, S.-E. Yoon, D. Tuft, D. Manocha, RT-DEFORM: Interactive Ray Tracing of Dynamic Scenes using BVHs, in: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 39–46.

[6] K. Garanzha, Efficient Clustered BVH Update Algorithm for Highly-Dynamic Models, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 123–130.

[7] M. Eisemann, C. Woizischke, M. Magnor, Ray Tracing with the Single-Slab Hierarchy, in: Proc. Vision, Modeling and Visualization (VMV) 2008, Konstanz, Germany, 2008, pp. 373–381.

[8] T.-J. Kim, B. Moon, D. Kim, S.-E. Yoon, RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies, IEEE Transactions on Visualization and Computer Graphics 16 (2) (2010) 273–286.

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

[10] J. Pantaleoni, D. Luebke, HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry, in: M. Doggett, S. Laine, W. Hunt (Eds.), Proceedings of the Conference on High Performance Graphics 2010, HPG '10, ACM SIGGRAPH/Eurographics, Saarbrücken, Germany, 2010, pp. 87–95.

[11] K. Garanzha, J. Pantaleoni, D. McAllister, Simpler and Faster HLBVH with Work Queues, in: Proceedings of the Conference on High Performance Graphics 2011, HPG '11, ACM SIGGRAPH/Eurographics, New York, NY, USA, 2011, pp. 59–64.

[12] T. Aila, S. Laine, Understanding the Efficiency of Ray Traversal on GPUs, in: Proceedings of the Conference on High Performance Graphics 2009, HPG '09, ACM SIGGRAPH/Eurographics, New York, NY, USA, 2009, pp. 145–149.

[13] K. Garanzha, C. Loop, Fast Ray Sorting and Breadth-First Packet Traversal for GPU Ray Tracing, Computer Graphics Forum 29 (2) (2010) 289–298.

[14] T. L. Kay, J. T. Kajiya, Ray Tracing Complex Scenes, SIGGRAPH Comput. Graph. 20 (1986) 269–278.

[15] K. Ng, B. Trifonov, Automatic Bounding Volume Hierarchy Generation Using Stochastic Search Methods, in: CPSC532D Mini-Workshop "Stochastic Search Algorithms", 2003, pp. 147–162.

[16] M. Stich, H. Friedrich, A. Dietrich, Spatial Splits in Bounding Volume Hierarchies, in: Proceedings of the Conference on High Performance Graphics 2009, HPG '09, ACM SIGGRAPH/Eurographics, New York, NY, USA, 2009, pp. 7–13.

[17] S. Popov, I. Georgiev, R. Dimov, P. Slusallek, Object Partitioning Considered Harmful: Space Subdivision for BVHs, in: Proceedings of the Conference on High Performance Graphics 2009, HPG '09, ACM SIGGRAPH/Eurographics, New York, NY, USA, 2009, pp. 15–22.

[18] V. Havran, J. Bittner, Rectilinear BSP trees for preferred ray sets, in: Proceedings of SCCG'99 (Spring Conference on Computer Graphics), Budmerice, Slovak Republic, 1999, pp. 171–179.

[19] B. Walter, K. Bala, M. Kulkarni, K. Pingali, Fast Agglomerative Clustering for Rendering, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 81–86.

[20] M. Reichl, R. Dünger, A. Schiewe, T. Klemmer, M. Hartleb, C. Lux, B. Fröhlich, GPU-based Ray Tracing of Dynamic Scenes, Journal of Virtual Reality and Broadcasting 7 (1).

[21] P. Bauszat, M. Eisemann, M. Magnor, The Minimal Bounding Volume Hierarchy, in: Proc. Vision, Modeling and Visualization (VMV) 2010, Siegen, Germany, 2010, pp. 227–234.

[22] I. Wald, On fast Construction of SAH-based Bounding Volume Hierarchies, in: Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing, IEEE Computer Society, Washington, DC, USA, 2007, pp. 33–40.

[23] Q. Hou, X. Sun, K. Zhou, C. Lauterbach, D. Manocha, Memory-Scalable GPU Spatial Hierarchy Construction, IEEE Transactions on Visualization and Computer Graphics 17 (4) (2011) 466–474.

[24] V. Havran, R. Herzog, H.-P. Seidel, Fast final gathering via reverse photon mapping, Computer Graphics Forum (Proceedings of Eurographics 2005) 24 (3) (2005) 323–333.

[25] J. Bittner, V. Havran, RDH: Ray Distribution Heuristics for Construction of Spatial Data Structures, in: H. Hauser (Ed.), 25th Spring Conference on Computer Graphics (SCCG 2009), ACM, Budmerice, Slovakia, 2009, pp. 61–67.

[26] T. Ize, C. Hansen, RTSAH Traversal Order for Occlusion Rays, Computer Graphics Forum (Proceedings of Eurographics 2011) 30 (2) (2011) 297–305.

[27] M. Ernst, G. Greiner, Multi Bounding Volume Hierarchies, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 35–40.

[28] H. Dammertz, A. Keller, The Edge Volume Heuristic - Robust Triangle Subdivision for Improved BVH Performance, in: Proceedings of the 2008 IEEE/Eurographics Symposium on Interactive Ray Tracing, 2008, pp. 155–158.

[29] S. Popov, J. Günther, H.-P. Seidel, P. Slusallek, Experiences with Streaming Construction of SAH KD-Trees, in: Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing, 2006, pp. 89–94.

[30] J. Günther, S. Popov, H.-P. Seidel, P. Slusallek, Realtime Ray Tracing on GPU with BVH-based Packet Traversal, in: Proceedings of the 2007 IEEE/Eurographics Symposium on Interactive Ray Tracing, IEEE Computer Society, Washington, DC, USA, 2007, pp. 113–118.

[31] V. Havran, J. Bittner, On Improving KD-Trees for Ray Shooting, Journal of WSCG 10 (1) (2002) 209–216.

[32] H. Hubschman, S. W. Zucker, Frame-to-frame coherence and the hidden surface computation: constraints for a convex world, ACM Trans. Graph. 1 (1982) 129–162.