Eurographics Symposium on Rendering - Experimental Ideas & Implementations (2015)
J. Lehtinen and D. Nowrouzezahrai (Editors)

Efficient visibility heuristics for kd-trees using the RTSAH

Matthias Moulin, Niels Billen, Philip Dutré

Department of Computer Science, KU Leuven, Belgium

Abstract

Acceleration data structures such as kd-trees aim at reducing the per-ray cost which is crucial for rendering performance. The de-facto standard for constructing kd-trees, the Surface Area Heuristic (SAH), does not take ray termination into account and instead assumes rays never hit a geometric primitive. The Ray Termination Surface Area Heuristic (RTSAH) is a cost metric originally used for determining the traversal order of the voxels for occlusion rays that takes ray termination into account. We adapt this RTSAH to building kd-trees that aim at reducing the per-ray cost of rays. Our build procedure has the same overall computational complexity and considers the same finite set of splitting planes as the SAH. By taking ray termination into account, we favor cutting off child voxels which are not or hardly visible to each other. This results in fundamentally different and more qualitative kd-trees compared to the SAH.

Categories and Subject Descriptors (according to ACM CCS): I.3.6 [Computer Graphics]: Methodology and Techniques—Graphics data structures and data types I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing

1. Introduction

High quality acceleration data structures for ray-tracing such as kd-trees aim to reduce the cost of tracing a ray. The standard heuristic for constructing these kd-trees is the Surface Area Heuristic (SAH) [MB90], which assigns to each candidate voxel a cost equal to the product of the cost of processing this voxel and the probability of actually processing this voxel. A top-down construction greedily minimizes this cost. In order to calculate these probabilities for each candidate voxel, the SAH assumes rays to be infinitely long, to start outside the scene's bounding box and the origins and directions of the rays to be uniformly distributed. Despite its popularity, the assumptions underlying the SAH usually do not hold in practice. In many scenes, most of the rays originate from inside the scene's bounding box and terminate when hitting a surface. By not taking ray termination into account, costs are overestimated, leading to a suboptimal ranking of the splitting planes and even to the decision of not splitting at all.

In this paper, we try to revise this assumption by revisiting the Ray Termination Surface Area Heuristic (RTSAH) [IH11], a cost metric originally used for determining the traversal order of the voxels for occlusion rays taking ray termination into account. We adapt this RTSAH to build view-independent kd-tree acceleration structures. Our build procedures have the same overall computational complexity and need to consider the same finite set of splitting planes as the SAH.

The contributions of this paper are as follows:

- We present a novel way of using the RTSAH for building kd-trees.
- We present two fast building algorithms for qualitative kd-trees based on the orthogonal projected and average projected surface area of the individual geometric primitives onto the splitting plane.
- We achieve reductions in intersection tests up to 47% for primary rays and up to 41% for shadow rays (when traversing the voxels in order) compared to the SAH.

2. Related Work

Acceleration data structures. Rays are intersected with the geometric primitives in the scene in order to determine which surfaces are visible and to resolve occlusion. The number of intersection tests can be kept between theoretical boundaries by using appropriate acceleration data structures (for a survey, see [WMG*09]).

The most well-known adaptive acceleration data structures are Bounding Volume Hierarchies (BVH) and kd-trees. BVHs and kd-trees are usually constructed using a greedy divide-and-conquer strategy for partitioning the three-dimensional space or the geometric primitives. Different traversal approaches exist for BVHs [HDW*11] and kd-trees [HH11].

Heuristics for building kd-trees. Heuristics are used for partitioning the three-dimensional space in a kd-tree. The best known and commonly used heuristic for generating qualitative kd-trees is the SAH. The SAH is originally introduced by Goldsmith and Salmon [GS87] for building BVHs and is adapted by MacDonald and Booth [MB90] for building kd-trees. The SAH assigns to each candidate voxel a cost equal to the product of the cost of processing this voxel and the probability of visiting this voxel. In order to calculate these probabilities for each voxel, the SAH assumes that:

1. the ray origins are uniformly distributed outside the scene's bounding box;
2. the ray directions are uniformly distributed;
3. the rays are infinitely long.

Several improvements have been made to the construction of the SAH. Havran et al. [HB02] introduced an efficient automatic termination criterion and clipping algorithm. Wald and Havran [WH06] encouraged cutting off large empty child voxels by reducing the expected cost of these splitting planes by a constant factor. This improvement usually results in slightly higher quality kd-trees. Hunt [Hun08] included mail-boxing, a ray-tracing optimization that removes redundant intersection tests. Wald and Havran [WH06] introduced a robust $\mathcal{O}(N \log N)$ build algorithm for kd-trees, where N is the total number of geometric primitives.

In addition, several fundamentally different heuristics are introduced to obtain a more accurate cost model since the assumptions of the SAH do not hold well in practice. Fabianowski et al. [FFD09] proposed the Scene Interior Ray Origins Heuristic which assumes ray origins to be uniformly distributed in the space inside the scene's bounding box. Havran and Bittner introduced build methods which take the actual distribution of rays into account for a fixed origin or direction [HB99] and by subsampling the rays [BH09]. Choi et al. [CCI12] proposed the Voxel Visibility Heuristic which considers the non-uniform distribution of rays by taking the occlusion of rays by geometric primitives into account. Reinhard et al. [RKJ96] and Havran [Hav00] considered ray termination inside a voxel by including a blocking factor. Vinkler et al. [VHS12] presented a visibility-driven modification of the SAH for building BVHs.

All these build heuristics use a greedy divide-and-conquer approach. When these local heuristics are extended to a global heuristic which considers the positioning of the splitting planes at all levels of the tree simultaneously, the problem of building the optimal kd-tree becomes NP-hard.

Traversal order for shadow rays. Shadow rays need not to be traced in front to back order to find the first intersection because their sole function consist of reporting occlusion. Ize and Hansen [IH11] introduced the RTSAH (based on Havran [Hav00]) for determining the traversal order for occlusion rays in BVHs and BSPs. Their heuristic assigns to each voxel a probability that a ray terminates upon traversal trough the voxel. When tracing a shadow ray, voxels with a higher termination probability are traversed first. For simplicity, non-empty leaf voxels are assumed to be completely opaque. The visibility of intermediate nodes is recursively obtained from its child nodes. Nah and Manocha [NM14] introduced the Surface Area Traversal Order metric which gives a higher traversal priority to the child voxel with the largest surface area.

Note that these algorithms, which change the traversal order, are orthogonal and fully compatible with our modified build heuristic.

Contribution. In this paper, we adapt the RTSAH to building kd-trees that reduce the cost of primary rays. By taking ray termination into account in the cost metric, we favor cutting off child voxels which are not or hardly visible to each other. This leads to a different ranking of the splitting planes which results in fundamentally different, more refined and more qualitative kd-trees compared to those obtained with the standard SAH.

For our RTSAH we introduce two new practical blocking factor approximations based on rasterization and the average projected surface area of the geometric primitives (not the axis-aligned bounding boxes).

3. Theoretical Framework

Hierarchical acceleration data structures that partition three-dimensional space such as kd-trees are often used to accelerate ray-tracing. To construct efficient acceleration structures, splitting planes that minimize the per-ray cost need to be chosen at each level of the tree. For computational efficiency, this is usually achieved by using appropriate (greedy) heuristics. In this section we briefly recall the SAH and introduce the modification of the original RTSAH [IH11] for building kd-trees.

3.1. Surface Area Heuristic

The SAH estimates the cost of splitting a voxel V with a splitting plane positioned in S into a left V_L and a right V_R child voxel as:

$$\mathcal{C}_\mathrm{SAH}(S : V \to \{V_\mathrm{L}, V_\mathrm{R}\}) = \mathcal{C}_\mathrm{t} + p_\mathrm{L} \cdot \mathcal{C}(V_\mathrm{L}) + p_\mathrm{R} \cdot \mathcal{C}(V_\mathrm{R}) \quad (1)$$

The heuristic consists of the cost of traversing the parent voxel, $\mathcal{C}_\mathrm{t}$, and the costs of visiting the child voxels, $\mathcal{C}(V_\mathrm{L})$ and $\mathcal{C}(V_\mathrm{R})$, multiplied by the probability of visiting these voxels, p_L and p_R. Figure 1 illustrates the partitioning of a voxel V.

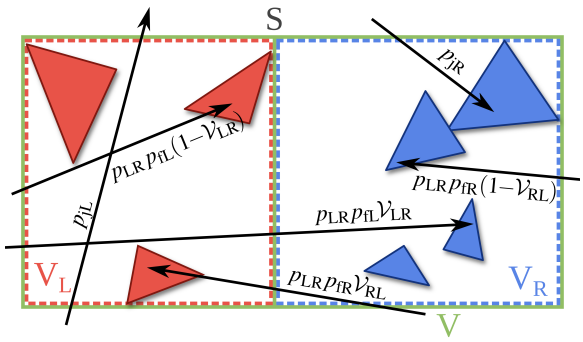


Figure 1: *The types of rays considered by the RTSAH with their corresponding probabilities: rays piercing just one child voxel (with probabilities p_{jL} and p_{jR}), rays piercing both child voxels while hitting (with probabilities $p_{LR}p_{fL}(1-\mathcal{V}_{LR})$ and $p_{LR}p_{fR}(1-\mathcal{V}_{RL})$) or not hitting (with probabilities $p_{LR}p_{fL}\mathcal{V}_{LR}$ and $p_{LR}p_{fR}\mathcal{V}_{RL}$) a geometric primitive inside the voxel that is pierced first. The RTSAH reduces to the SAH when $\mathcal{V}_{LR} = \mathcal{V}_{RL} = 1$.*

The cost $\mathcal{C}(V)$ of visiting a voxel is defined as the cost of testing a ray for intersection with all the geometric primitives contained in V. The probabilities p_L and p_R are the conditional probabilities of respectively piercing the left or right child voxel after piercing the parent voxel. When the ray origins and directions are uniformly distributed outside the scene's bounding box, these probabilities are exactly equal to:

$$p_L = p(V_L \text{ pierced} \mid V \text{ pierced}) = \frac{SA_{V_L}}{SA_V} \quad (2)$$

$$p_R = p(V_R \text{ pierced} \mid V \text{ pierced}) = \frac{SA_{V_R}}{SA_V} \quad (3)$$

where SA_{V_k} is the surface area of V_k.

The cost of not splitting a voxel is defined as the cost of intersecting all the geometric primitives contained in this voxel:

$$\mathcal{C}_{\text{no split}}(V) = \mathcal{C}(V). \quad (4)$$

A voxel will only be split into two child voxels when the cost of splitting is lower than the cost of not splitting.

3.2. Ray Termination Surface Area Heuristic

By not taking the presence of geometric primitives in the probabilities p_L and p_R into account, the SAH does not distinguish between rays piercing both child voxels while hitting or not hitting a geometric primitive inside the voxel that is pierced first. The RTSAH extends the SAH by distinguishing between these types of rays and by taking ray termination into account. The RTSAH cost function is defined as:

$$\mathcal{C}_{\text{RTSAH}}(S : V \to \{V_L, V_R\}) = \mathcal{C}_t + p_{jL} \cdot \mathcal{C}(V_L) + p_{jR} \cdot \mathcal{C}(V_R) + p_{LR}\Big(p_{fL}\Big(\mathcal{C}(V_L) + \mathcal{V}_{LR} \cdot \mathcal{C}(V_R)\Big) + p_{fR}\Big(\mathcal{C}(V_R) + \mathcal{V}_{RL} \cdot \mathcal{C}(V_L)\Big)\Big) \quad (5)$$

p_{jL}, p_{jR} and p_{LR} express the probability of piercing just the left child, just the right child and both child voxels respectively. When both child voxels are pierced, we distinguish the case where either the left or right child voxel is pierced first using the probabilities p_{fL} and p_{fR}. Rays which pierce both child voxels can potentially be terminated by a geometric primitive in the child voxel that is pierced first. The visibility probability, $\mathcal{V}_{LR}$ ($\mathcal{V}_{RL}$), expresses the fraction of rays, piercing the left (right) child voxel first and then the right (left) child voxel, that reach the splitting plane without being terminated by a geometric primitive inside the left (right) child voxel. The types of rays with their corresponding probabilities are illustrated in Figure 1.

We obtain expressions for the probabilities p_{jL}, p_{jR} and p_{LR} using Equation 2 and 3:

$$\begin{aligned} p_{jL} &= p(\text{just } V_L \text{ pierced} \mid V \text{ pierced}) \\ &= 1 - p_R \end{aligned} \quad (6)$$

$$\begin{aligned} p_{LR} &= p(V_L \text{ and } V_R \text{ pierced} \mid V \text{ pierced}) \\ &= 1 - p_{jL} - p_{jR} \end{aligned} \quad (7)$$

$$\begin{aligned} p_{fL} &= p(V_L \text{ pierced first} \mid V_L \text{ and } V_R \text{ pierced}) \\ &= \frac{1}{2} \end{aligned} \quad (8)$$

This is analogously for p_{jR} and p_{fR}. The visibility probabilities between the child voxels are defined as:

$$\mathcal{V}_{LR} = p(\text{no hit in } V_L \mid V_L \text{ pierced first, then } V_R) \quad (9)$$

$$\mathcal{V}_{RL} = p(\text{no hit in } V_R \mid V_R \text{ pierced first, then } V_L) \quad (10)$$

Note that the RTSAH cost becomes equal to the SAH cost when $\mathcal{V}_{LR} = \mathcal{V}_{RL} = 1$.

The cost of not splitting a voxel is the same as for the SAH (Equation 4). The empty space bonus [WH06] and its underlying motivation can also be applied to the RTSAH.

4. Practical Algorithm

In order to obtain a practical build algorithm based on the RTSAH, we need to evaluate the splitting cost (Equation 5) for a finite set of splitting planes.

The probabilities of a ray intersecting a single or both child voxels can be trivially calculated using Equations 6 to 8. However, the exact calculation of the visibility probabilities $\mathcal{V}_{LR}$ and $\mathcal{V}_{RL}$ requires the integration of the visibility over the hemisphere of incoming directions and the area of the splitting plane (see Figure 2a). Due to the dependence on the geometrical distributions in the child voxels no closed-form expression can be found for the visibility probabilities.

While the visibility probabilities can be calculated via Monte Carlo integration techniques, this is impractical due to the absence of an acceleration data structure during the construction phase. Furthermore, performing a Monte Carlo sampling of the visibility probabilities at each possible splitting plane would result in prohibitively large build times. The number of samples needs to be low when lots of geometric primitives overlap the voxel that must be split, which is the case for the first split decisions of the kd-tree. On the contrary, the choice of splitting plane is more important and decisive at these first split decisions and thus asks for an accurate estimator.

In order to maintain a tractable build time, we aim at a practical algorithm which for each split decision:

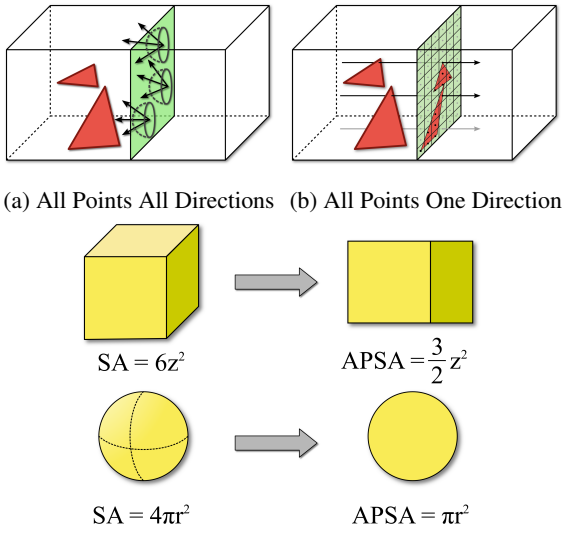
1. considers only a finite set of splitting planes including the best splitting plane according to our approximated RTSAH;
2. allows for an incremental calculation of the visibility probabilities.

Both conditions are hard to achieve due to the directional dependence of the visibility probabilities. Therefore, we suggest to eliminate this dependence by considering only a single direction: the orthogonal projection direction (section 4.1) and the average projected direction (section 4.2) onto the splitting plane. In section 4.3, we validate these approximations. In sections 4.4 and 4.5, we show that these approximations to the visibility probabilities lead to a practical build algorithm based on the RTSAH for which both conditions are satisfied.

4.1. All Points One Direction RTSAH

The All Points One Direction (APOD) RTSAH considers all the points on the splitting plane and ignores the directional dependence by considering only the direction orthogonal to the splitting plane. This corresponds to projecting the geometric primitives onto the splitting plane. The visibility is than equal to the area of the splitting plane which is not covered by the projected primitives divided by the total area of the splitting plane. This can be calculated efficiently using rasterization. To perform the rasterization, the splitting plane is partitioned into a finite number of square rasterization cells of which the center is called a rasterization point. A rasterization cell is covered by a geometric primitive if and only if the orthogonal projection of the geometric primitive covers the rasterization point of this cell. Figure 2b illustrates this idea.

The idea of calculating the visibility between two neighbouring child voxels with an orthogonal projection onto the splitting plane is similar to calculating the blocking factor of a child voxel in [RKJ96]. In contrast to [RKJ96], we do not use the axis aligned bounding boxes of the geometric primitives, but use the actual geometric primitives during rasterization. Furthermore, we implicitly avoid overlapping of geometric primitives by rasterizing and obtain both a cost model and build procedure.



(a) All Points All Directions (b) All Points One Direction

(c) Average Projected Surface Area

Figure 2: *The ideas behind the different approximations to the RTSAH. (a) shows the ground truth, where we integrate over the splitting plane's area and the hemisphere of incoming directions. (b) shows our APOD approximation, where we integrate over every point but only take the direction orthogonal to the splitting plane into account. (c) the average projected surface area of a convex solid.*

The rasterization can be seen as a special form of sampling, resulting in a biased estimator. The rasterization of geometric primitives is however computationally cheaper than testing random rays for intersection, reuses intersection information while shifting the splitting plane in the orthogonal direction and does not require a second kd-tree compared to [Hav00].

4.2. Average Projected Surface Area RTSAH

Cauchy showed that the average projected area of a three-dimensional convex solid is one fourth of its surface area (see Figure 2c). This property is used for obtaining the probabilities p_L and p_R of Equation 2 and 3.

The Average Projected Surface Area (APSA) RTSAH approximates the visibility probabilities as the fraction of the area of the splitting plane that remains after subtracting the average projected surface area of the geometry contained in a child voxel:

$$\mathcal{V}_{LR} \approx 1 - \min\left(\frac{SA_{G_{V_L}}}{4SA_S}, 1\right), \mathcal{V}_{RL} \approx 1 - \min\left(\frac{SA_{G_{V_R}}}{4SA_S}, 1\right) \quad (11)$$

Where $SA_{G_{V_k}}$ is the surface area of all the geometric primitives contained in voxel V_k and SA_S the surface area of the splitting plane. Note that when the sets G_{V_L} and G_{V_R} represent a convex solid fully contained in V_L and V_R respectively, the above approximations are exact.

4.3. Validation

To validate our approximations of the visibility probabilities, we constructed an artificial scene consisting of a variable number of stratified placed icosahedrons. The visibility probabilities in this context express the probability that a ray passes through the different planes of the scenes' bounding box. To obtain ground truth visibility probabilities, the scenes' bounding box got pierced by 100M rays which are uniformly distributed on the scenes' bounding sphere and are guaranteed to pierce the scenes' bounding box. For the APOD we used different numbers (16^2, 32^2, 64^2, 128^2, 256^2, 512^2, 1024^2) of rasterization points. The results are shown in Figure 3.

The APSA consistently results in the smallest difference between the approximated and exact visibility probabilities for the 6 planes of a scene's bounding box compared to the APOD. Note the slow linear increase of these underestimations compared to the exponential increase in the number of icosahedrons. The APOD which rasterizes onto 1024^2 points does slightly worse. Unfortunately, the APOD needs a small number of rasterization points to result in a fast build procedure. For these scenes this is not problematic since the other APOD approximations differ at most 5% with an exception for the approximation which uses 16^2 rasterization points. Considering only 16 rasterization points in 1D is not accurate enough given the number of geometric primitives in these scenes. Furthermore, the result becomes too sensitive to small shifts of the rasterization points. The APSA results on average in an underestimation of 12.7% in the scene with 2048 icosahedrons which is still a reasonable approximation given the difficulty of calculating the exact visibility probabilities for this scene.

We conclude that for these difficult (with regard to visibility) scenes, the APSA results in satisfactory approximations of the visibility probabilities. The APOD on the other hand is less accurate and depends on the number of rasterization points used. In the following section, we use at least 16 and at most 64 rasterization points in 1D depending on the number of geometric primitives that overlap the voxel that must be split.

4.4. Split candidates

For simplicity, we do not clip geometric primitives that overlap multiple voxels when rasterizing (APOD) or calculating surface areas (APSA). Therefore, the APOD and APSA RTSAH cost functions increase or decrease monotonously in the region between two consecutive geometric primitive boundaries along the same axis (illustrated in Figure 4a). Within such a region, the approximations of the visibility probabilities $\mathcal{V}_{LR}$, $\mathcal{V}_{RL}$ remain constant and p_{jL}, p_{jR} are both linear functions of the position of the splitting plane. This means that we only have to consider splitting planes at the boundaries of the geometric primitives in order to obtain the best splitting plane according to our cost metric (see Figure 4b). This is the same set of splitting planes considered by the SAH.

4.5. Build algorithm

We only have to make some small modifications to the build procedure for the SAH without changing the overall computational complexity. Obtaining p_{jL}, p_{jR}, p_{LR} is very similar to calculating p_L and p_R. The visibility probabilities are incrementally updated during a single pass through the splitting candidates for each primary direction. When moving the splitting plane from the left to the right, this requires incrementally decreasing $\mathcal{V}_{LR}$ and increasing $\mathcal{V}_{RL}$. The calculation of the visibility probabilities depends on the approximation used:

APOD RTSAH: For $\mathcal{V}_{LR}$ we need to keep track which rasterization points are not overlapped by geometric primitives contained in V_L (invariant). For $\mathcal{V}_{RL}$ we need to keep track how many geometric primitives contained in V_R cover each rasterization point (invariant). Therefore, all geometric primitives contained in V need to be rasterized onto the splitting plane prior to determining the costs of each split candidate.

APSA RTSAH: To calculate $\mathcal{V}_{LR}$ and $\mathcal{V}_{RL}$, we have to keep track of the sum of the surface area of all geometric primitives contained in the left and right child voxel respectively (invariants). This can be done incrementally by initializing $\mathcal{V}_{RL}$ with the sum of the surface area's of all the geometric primitives contained in voxel V.

The data structures that needs to guarantee these invariants, must be updated (incremented/decremented) appropriately when a geometric primitive starts overlapping the left child voxel or stops overlapping the right child voxel.

5. Experimental Results

We have implemented the APOD and APSA RTSAH in PBRT [PH10]. Our test scenes are shown in Figure 5. We compare the obtained kd-tree structures in 5.1 and the performance with regard to the number of intersection tests and traversal steps in Section 5.2 as opposed to the SAH.

5.1. Kd-tree properties

Table 1 summarizes the properties of the obtained kd-trees. The kd-trees obtained with the APSA RTSAH contain less geometric primitives per leaf as opposed to the SAH. Therefore leaves are traversed faster on average. On the other

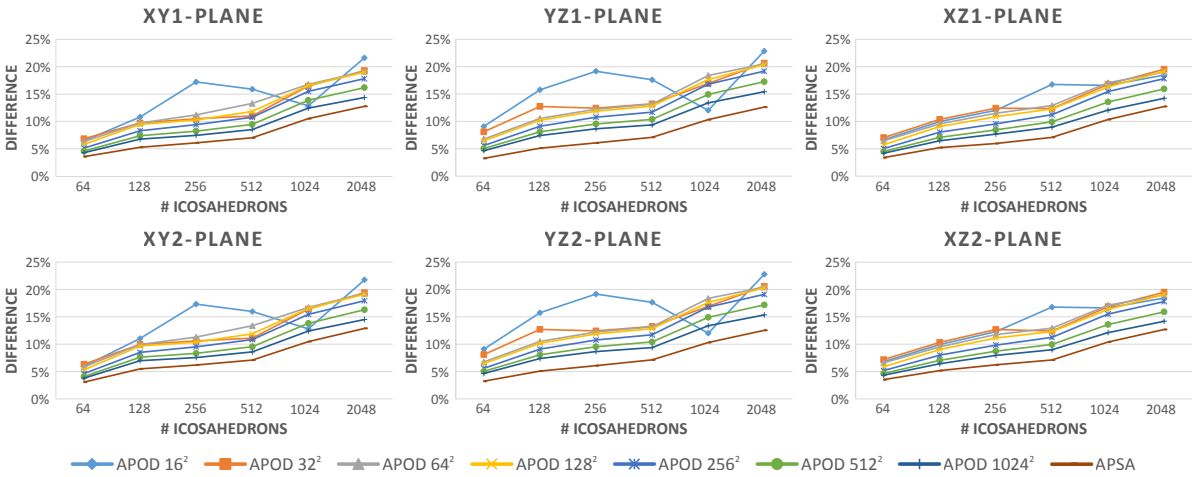
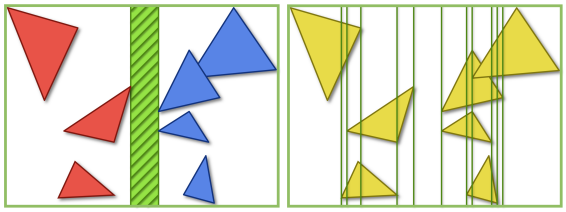


Figure 3: *The difference between the approximated and exact (100M rays) visibility probabilities for the 6 planes of the scene's bounding box. A positive (negative) difference corresponds to an underestimation (overestimation).*



(a) Region of no interest (b) Splitting planes

Figure 4: *(a) Shows a region (green) where the SAH, APOD and APSA RTSAH cost functions are monotonously increasing/decreasing. (b) The finite set of splitting planes (green) along a single direction considered by the SAH, APOD RTSAH and APSA RTSAH to obtain the optimal splitting plane according to their respective metrics.*

hand, the subtrees are more refined, resulting in 30 to 60% as many voxels and more duplicated references to geometric primitives for the same maximal depth. Therefore the average number of traversed voxels will be higher, more memory needs to be allocated, but also fundamentally better kd-trees can be obtained as opposed to the SAH.

We see similar results with regard to the number of (empty) leaves and geometric primitives for the kd-trees obtained with the APOD RTSAH. Only the `sponza` scene is less refined compared to the SAH. Despite the refinement, the number of geometric primitives per leaf as opposed to the APSA RTSAH is less ideal.

Both RTSAH build times are higher than the SAH due to the larger and deeper obtained kd-trees. Therefore, the APSA RTSAH results in minimal overhead compared to the SAH for the same number of split decisions. The APOD RTSAH does worse, because rasterizing at the highest levels of the kd-tree is computationally expensive. Our APOD or APSA RTSAH (CPU) build procedures are not optimized and do not cache surface areas or rasterization info for geometric primitives. These improvements can still result in a speedup, but are not tested.

5.2. Performance

Table 2 shows the difference in geometric primitive intersection tests, traversed voxels for different rays and the difference in total rendering time. As expected due to the properties of the kd-trees, more voxels need to be traversed for the APOD and APSA RTSAH due to the more refined subtrees compared to the SAH.

More importantly, we clearly see a large reduction in geometric primitive intersection tests for the APSA RTSAH. We obtain reductions up to 47% for primary rays for the `sibenik` and `sponza` scenes. In these scenes, ray termination is important to consider by a build heuristic since every ray will eventually hit some surface. We obtain a small increase in intersection tests for the `bunny` scene consisting of a single object and a floor plate. The APSA RTSAH needs less tests on the bunny, but performs slightly worse on the floor plate. Despite only focusing on exterior rays, we even see reductions to 41% for shadow rays (when traversing the voxels in order). This is due to the larger number of duplicated references to geometric primitives.

The APSA RTSAH generally outperforms the APOD RTSAH in more complex scenes (such as the `sibenik` and `sponza` scenes). By considering only the incoming normal direction for the APOD RTSAH, we note a difference between `sibenik1` (perspective camera) and `sibenik3`

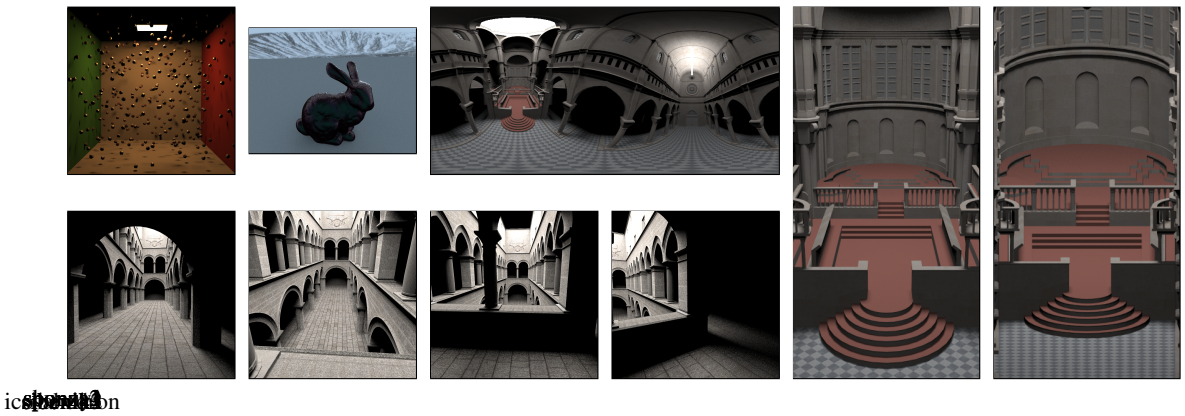


Figure 5: *Test scenes. The* `icosahedron` *scene consists of approximately* 7*k, the* `bunny` *scene of* 69*k, the* `sibenik` *scenes of* 80*k and the* `sponza` *scenes of* 66*k geometric primitives.

scene	build time			#leafs			#empty leafs		
	SAH	APOD	APSA	SAH	APOD	APSA	SAH	APOD	APSA
icosahedron	0.1s	3.3s	0.2s	65 k	87 k	94 k	6 k	6 k	6 k
bunny	0.6s	30.2s	2.1s	873 k	1.17 M	1.22 M	198 k	224 k	219 k
sibenik	0.7s	31.7s	2.5s	607 k	829 k	888 k	127 k	142 k	156 k
sponza	0.6s	16.3s	2.1s	502 k	452 k	823 k	130 k	109 k	176 k

scene	#references to geometric primitives			maximum #geometric primitives/leaf			average #geometric primitives /non-empty leaf		
	SAH	APOD	APSA	SAH	APOD	APSA	SAH	APOD	APSA
icosahedron	170 k	224 k	239 k	11	8	8	2.86	2.78	2.71
bunny	1.80 M	2.46 M	2.53 M	12	11	11	2.67	2.59	2.52
sibenik	1.97 M	2.87 M	2.95 M	81	72	36	4.10	4.18	4.03
sponza	1.64 M	1.66 M	2.59 M	57	54	44	4.41	4.84	4.01

Table 1: *Kd-tree structures obtained for the SAH, APOD and APSA RTSAH build heuristics for the test scenes.*

(orthographic camera). On the contrary, the APSA RTSAH results in similar gains independent of the type of camera used.

The false color images in Figure 6 show a more detailed comparison between the build heuristics. Here we can for example see that less intersection tests are performed near the pillars of the `sponza` scenes due to the inclusion of ray termination in our RTSAH approximations.

While we achieve significant reductions in intersection tests, the impact on the total rendering time is less pronounced. Our profiler showed that (depending on the scene and build heuristic) less than 35% of the time is spent on testing geometric primitives for intersection or traversing the acceleration data structure. Therefore, the reductions in rendering time are smaller.

6. Conclusions and further work

Our RTSAHs result in more qualitative kd-trees compared to the SAH by taking ray termination into account. Furthermore, our RTSAH build procedures have the same overall computational complexity and need to consider the same finite set of splitting planes as the SAH.

We plan to precisely investigate the differences between the acceleration data structures generated by the SAH and our RTSAHs and to further validate the underlying assumptions and simplifications. Furthermore, we want to use our approximated visibility probabilities to determine the traversal order for shadow rays as opposed to the (original) RTSAH introduced by Ize and Hansen [IH11]. Both the SAH and RTSAH do not take the actual ray distribution nor the presence of internal rays into account. The SAH for scene-interior ray origins introduced by Fabianowski et al. [FFD09] could also benefit from the ray termination of our RTSAHs.

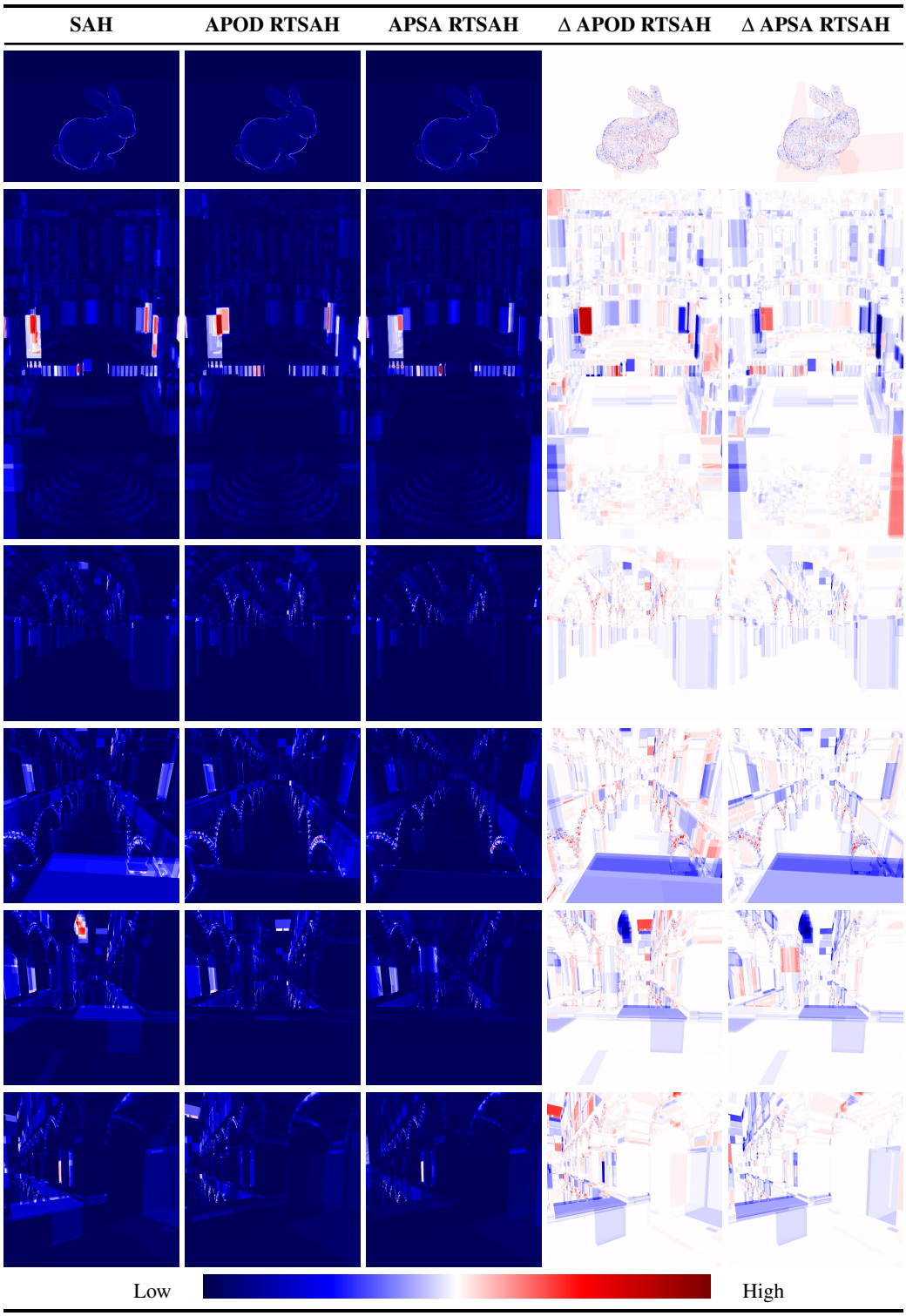


Figure 6: *False color images for the number of intersection tests for primary rays for the SAH, APOD and APSA RTSAH and the difference (blue (red) corresponds to a gain (loss)) between the SAH and RTSAH approximations. All scenes are rendered with a Whitted ray tracer [Whi80] using 512 samples per pixel. The scale is linear.*

scene	Δ intersection tests for primary rays		Δ intersection tests for shadow rays		Δ traversed voxels for primary rays		Δ traversed voxels for shadow rays		Δ total rendering time	
	APOD	APSA	APOD	APSA	APOD	APSA	APOD	APSA	APOD	APSA
icosahedron	-1.7%	-0.5%	-1.8%	-8.5%	1.2%	-0.6%	3.0%	-0.8%	-3,6%	-4,4%
bunny	-1.8%	0.9%	-1.5%	-3.7%	5.5%	2.6%	7.4%	2.7%	1.8%	-1.0%
sibenik1	-13.5%	-18.7%	-3.9%	-13.8%	13.4%	11.6%	15.5%	15.6%	-1.6%	-3.5%
sibenik2	-10.6%	-20.2%	-9.1%	-17.7%	8.0%	6.7%	12.3%	12.0%	-3.2%	-4.8%
sibenik3	-17.9%	-17.9%	-8.0%	-9.3%	13.7%	10.7%	14.3%	15.2%	0.4%	-0.1%
sponza1	-12.5%	-25.5%	-3.1%	-22.1%	8.2%	7.6%	-2.2%	10.6%	-3.9%	-4.8%
sponza2	-38.0%	-46.7%	-6.0%	-29.2%	3.0%	20.0%	-0.5%	14.4%	-7.4%	-7.6%
sponza3	-20.5%	-30.7%	-18.9%	-40.8%	4.6%	12.2%	1.3%	10.8%	-7.5%	-8.5%
sponza4	-12.9%	-27.4%	-14.8%	-33.8%	18.6%	7.1%	3.4%	10.4%	-0.4%	-3.1%

Table 2: *The difference in geometric primitive intersection tests, traversed voxels for primary and shadow rays and total rendering time for the APOD and APSA RTSAH relative to the SAH, where a negative (positive) percentage corresponds to a decrease (increase). All scenes are rendered with a Whitted ray tracer [Whi80] using 512 samples per pixel (on 8 logical threads).*

7. Acknowledgments

The scenes used in this paper are courtesy of PBRT [PH10]. Niels Billen is funded by the Agency for Innovation by Science and Technology in Flanders (IWT).

References

[BH09] BITTNER J., HAVRAN V.: Rdh: ray distribution heuristics for construction of spatial data structures. In *Proceedings of the 25th Spring Conference on Computer Graphics* (2009), ACM, pp. 51–58. 2

[CCI12] CHOI B., CHANG B., IHM I.: Construction of efficient kd-trees for static scenes using voxel-visibility heuristic. *Computers Graphics 36*, 1 (2012), 38 – 48. 2

[FFD09] FABIANOWSKI B., FOWLER C., DINGLIANA J.: A cost metric for scene-interior ray origins. *Eurographics Short Papers* (2009), 49–52. 2, 7

[GS87] GOLDSMITH J., SALMON J.: Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl. 7*, 5 (May 1987), 14–20. 2

[Hav00] HAVRAN V.: *Heuristic ray shooting algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University, Prague, 2000. 2, 4

[HB99] HAVRAN V., BITTNER J.: Rectilinear bsp trees for preferred ray sets. *Proceedings of SCCG 99* (1999), 171–179. 2

[HB02] HAVRAN V., BITTNER J.: On improving kd-trees for ray shooting. In *In Proc. of WSCG 2002 Conference* (2002), pp. 209–217. 2

[HDW*11] HAPALA M., DAVIDOVIČ T., WALD I., HAVRAN V., SLUSALLEK P.: Efficient stack-less bvh traversal for ray tracing. In *Proceedings of the 27th Spring Conference on Computer Graphics* (2011), ACM, pp. 7–12. 2

[HH11] HAPALA M., HAVRAN V.: Review: Kd-tree traversal algorithms for ray tracing. *Computer Graphics Forum 30*, 1 (2011), 199–213. 2

[Hun08] HUNT W. A.: Corrections to the surface area metric with respect to mail-boxing. In *Interactive Ray Tracing 2008, IEEE Symposium on* (2008), IEEE, pp. 77–80. 2

[IH11] IZE T., HANSEN C.: Rtsah traversal order for occlusion rays. *Computer Graphics Forum 30*, 2 (2011), 297–305. 1, 2, 7

[MB90] MACDONALD D. J., BOOTH K. S.: Heuristics for ray tracing using space subdivision. *Vis. Comput. 6*, 3 (May 1990), 153–166. 1, 2

[NM14] NAH J.-H., MANOCHA D.: Sato: Surface-area traversal order for shadow ray tracing. *Computer Graphics Forum* (2014). 2

[PH10] PHARR M., HUMPHREYS G.: *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010. 5, 9

[RKJ96] REINHARD E., KOK A. J. F., JANSEN F. W.: Cost prediction in ray tracing. In *Rendering Techniques* (1996), Springer, pp. 41–50. 2, 4

[VHS12] VINKLER M., HAVRAN V., SOCHOR J.: Visibility driven bvh build up algorithm for ray tracing. *Computers & Graphics 36*, 4 (2012), 283–296. 2

[WH06] WALD I., HAVRAN V.: On building fast kd-trees for ray tracing, and on doing that in o (n log n). In *Interactive Ray Tracing 2006, IEEE Symposium on* (2006), IEEE, pp. 61–69. 2, 3

[Whi80] WHITTED T.: An improved illumination model for shaded display. *Commun. ACM 23*, 6 (June 1980), 343–349. 8, 9

[WMG*09] WALD I., MARK W. R., GÜNTHER J., BOULOS S., IZE T., HUNT W., PARKER S. G., SHIRLEY P.: State of the art in ray tracing animated scenes. *Computer Graphics Forum 28*, 6 (2009), 1691–1722. 1