RDH: Ray Distribution Heuristics for Construction of Spatial Data Structures

Jiří Bittner*
Faculty of Electrical Engineering
Czech Technical University in Prague

Vlastimil Havran†
Faculty of Electrical Engineering
Czech Technical University in Prague

Abstract

Surface area heuristics is currently the most popular method for view independent construction of spatial hierarchies for ray tracing. We present a method which modifies the surface area heuristics by taking into account the actual distribution of rays in the scene. This is achieved by subsampling the rays to be cast and using these rays in order to estimate the probabilities of rays traversing through nodes of the constructed hierarchy. The main aim of our paper is to analyze the potential of taking the ray distribution into account. The results indicate that we can achieve a minor speedup of ray traversal compared to standard SAH. For large densely occluded scene we can also save the construction time and memory consumption of the hierarchy by not subdividing parts of the scene where no rays are traced.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—[Ray tracing]

1 Introduction

The fundamental task of ray tracing is to determine visibility along a given ray, i.e. to find the closest intersection of the ray with the scene. In order to find the intersection as fast as possible the search must be limited only to the proximity of the ray. This can be achieved by organizing the scene in a data structure such as regular grid, kD-tree, octree, or bounding volume hierarchy.

In particular two hierarchical data structures became very popular: kD-trees and bounding volume hierarchies. The major advantage of these hierarchies is their ability to efficiently adapt to irregular distribution of objects in the scene. The best known technique for this adaptation is the *Surface Area Heuristics* (SAH) originally proposed by Goldsmith and Salmon[Goldsmith and Salmon 1987] in the context of bounding volume hierarchies and then applied to kD-trees by MacDonald and Booth [MacDonald and Booth 1990]. The SAH uses a cost model which reflects the expected number of computed intersections as well as the expected number of traversal steps. This model is then used in a greedy optimization process which aims to minimize the cost of the constructed hierarchy. The core idea of the SAH is the evaluation of probability of ray visiting nodes of the hierarchy based on their surface areas. This evaluation assumes uniform distribution of rays and no occlusion.

In this paper we relief the assumption of uniform ray distribution made by the SAH. Instead we use explicit knowledge of ray distribution for a given image or a sequence of images. We aim at two main contributions: (1) We analyze the implications of the ray uniformity assumption made by the SAH. (2) We outline possible applications of the new heuristics which exploit spatial and temporal coherence of ray distributions.

*e-mail: bittner@fel.cvut.cz
†e-mail: havran@fel.cvut.cz

2 Motivation and Related Work

The most time consuming task of ray tracing is finding a closest intersection with the scene for every ray. This task is accelerated by data structures which limit the search for the intersection only to the proximity of the given ray. Data structures for ray tracing [Whitted 1979] have been investigated intensively over last three decades. There is a large body of literature on this topic which have been summarized in several thorough surveys [Slusallek et al. 2005; Chang 2004; Havran 2000; Glassner 1989].

The common principle of efficient data structures for ray tracing is to partition the scene into spatial cells and sorting the scene objects into these cells. When a ray is cast we identify the cells intersected by the ray and only objects referenced in these cells are tested for intersection with the ray.

There are two main categories of data structures used in ray tracing: *regular data structures* and *hierarchical data structures*. The most popular regular data structure are uniform grids [Fujimoto et al. 1986]. Ray tracing with uniform grids is efficient if the distribution of scene objects is also relatively uniform. This condition is often violated for practical scenes, which severely decreases the ray tracing performance.

The hierarchical data structures are commonly represented by a tree and possibly augmented by additional data structures. The major advantage of the hierarchies is their easy adaptation to the actual distribution of the objects.

The construction of the spatial data structures is analogous to spatial sorting [Samet 2006]. In particular, a top-down construction of a hierarchical spatial data structure is *Divide and Conquer* method, which is analogous to quicksort. In order to create an interior node of our data structure we have a set of objects residing in some spatial region. Our task is to distribute the objects into two or more child nodes. We use an algorithmic rule which prescribes how the objects should be distributed to these child nodes. This algorithmic rule corresponds to the pivot used in the traditional 1D quicksort. The final performance of the data structure heavily depends of the quality on the pivot selection.

If our pivot is formed by three planes perpendicular to main axes, then we create an octree [Glassner 1984]. If we allow that the spatial cells associated with children overlap, we create a bounding volume hierarchy. If we use a single splitting plane perpendicular to one coordinate axis we construct a kD-tree [MacDonald and Booth 1990]. In particular kD-trees, have proved very efficient in practice and therefore they are often the primary choice when implementing a ray tracing application.

The major factor influencing the quality of the constructed data structure is the positioning of the splitting plane in the spatial extent defined by an interior node. One possibility is to put the splitting plane in the middle (*spatial median*), another possibility is to balance the number of objects on the left and right side of the splitting plane (*object median*). Since our query is a ray passing through the scene until it hits an object, it is important to consider the geometric probability that a ray hits a spatial cell. This probability

is proportional to the surface area of a spatial cell assuming that distribution of rays in the scene is uniform [Solomon 1978]. The *surface area heuristics* (SAH) is a cost model which combines the geometric probabilities with the estimates of the traversal and intersection costs for a node being subdivided. According to the experiments [MacDonald and Booth 1990; Havran 2000] the SAH can lead to the performance of ray tracing to be by order(s) of magnitude higher than when we use spatial and object median approach.

We would like to note that the cost model based on SAH is useful also for other data structures that use spatial cells even if these cells overlap. This includes octrees [Whang et al. 1995], bounding volume hierarchies [Wald et al. 2007], and light-weight bounding volume hierarchies [Havran et al. 2006; Woop et al. 2006].

It was shown experimentally that the kD-trees constructed with SAH cost model are already very efficient in practice [Havran 2000]. However, Havran and Bittner [Havran and Bittner 1999] showed that it is possible to further improve the SAH by lifting the assumption that rays are distributed uniformly. They assumed that the rays are formed by a perspective, orthogonal, or spherical camera. In this paper we further lift the assumption that the set of rays is generated by a camera and propose a heuristics which allows for arbitrary ray distribution. Additionally, unlike previous techniques the proposed method considers occlusion in the evaluation of the cost model.

3 Overview

The method proposed in this paper extends the surface area heuristics by using explicit knowledge of ray distribution. The ray distribution is modeled using *representative ray set*. The representative ray set is generally a subset of rays cast in the current frame. Alternatively it is a subset of rays cast in the previous frame of an animation, assuming sufficient temporal coherence between these frames exists.

The representative ray set is used during the construction of the hierarchy by tracking all rays intersecting the current leaves of the subdivision. When the position of the splitting plane is being determined for a given leaf node, we use these rays to estimate the probabilities of rays intersecting the newly established parts of the leaf. As a result the new data structure is adapted to the actual ray distribution.

The paper is organized as follows: In Section 4 we review the SAH. In Section 5 we present the new method based on the ray distribution. Section 6 describes implementation details of the new method. Section 7 presents results and their discussion. Finally, Section 8 concludes the paper.

4 Surface Area Heuristics

Surface area heuristics is a greedy optimization method which aims to minimize the cost of the constructed spatial subdivision. In particular it optimizes the position of the splitting plane by minimizing the cost of traversal steps and ray-object intersections induced by the nodes created by the split.

The SAH uses the following cost function:

$$C = c_t + c_i(p_L^{SAH}|O_L| + p_R^{SAH}|O_R|), \tag{1}$$

where $p^{SAH}_{\{L|R\}}$ is the probability of rays intersecting the left and right children, respectively, $|O_{\{L|R\}}|$ is the number of objects in the left and right children, c_t is the traversal cost of interior node of the hierarchy (containing the splitting plane) and c_i is the cost of intersection computation. The crucial part of the SAH is the estimation of $p^{SAH}_{\{L|R\}}$, which is computed as follows:

$$p^{SAH}_{\{L|R\}} = \frac{S_{\{L|R\}}}{S}, \tag{2}$$

where $S_{\{L|R\}}$ are the surface areas of the left and right child, respectively, and S is the surface area of the subdivided node (see figure 1).

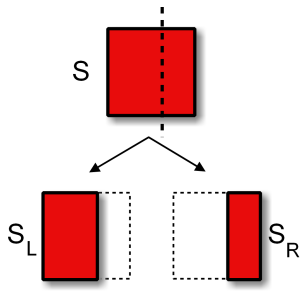

Figure 1: Subdivision of a node using a splitting plane induces two nodes with smaller bounding boxes. The SAH uses the ratio of the surface areas of the new boxes and the original box to estimate the probability of ray traversing to the corresponding nodes.

5 Ray Distribution Heuristics

The SAH uses surface areas to estimate the probability of ray traversing the left or right part of the tree. Our ray distribution heuristics evaluates these probabilities by taking into account the distribution of the actual rays which are traced.

5.1 Representing ray distribution

We represent the distribution of rays cast by the ray tracing algorithm using a *representative ray set* (RRS). The RRS can be significantly smaller than the actual set of rays to be cast as long as it describes the ray distribution reasonably well.

Let us denote the set of rays cast in frame i of an animation R^i and the set of rays used as RRS for frame i R^i_{RRS}. We consider the following possibilities for obtaining the RRS:

- $R^i_{RRS} \subseteq R^i$. The set of sample rays for frame i is a subset of rays cast in frame i. The ray distribution is described by subsampling the original ray set.
- $R^i_{RRS} \subseteq R^{i-1}$. The set of sample rays for frame i is a subset of rays cast in the previous frame $i-1$. The ray distribution is described by subsampling the ray set cast in the previous frame.
- $R^i_{RRS} \subseteq \bigcup_{\forall i} R^i$. The set of sample rays for frame i is the same for all frames and corresponds to a subset of the rays cast in all frames of the animation.

The first corresponds to undersampling in spatial domain assuming the rays cast for neighboring pixels are coherent. The second possibility also reuses the rays in temporal domain; rays cast in the previous frame are used as RRS for the current frame assuming the rays exhibit temporal coherence. The third possibility corresponds to a more aggressive undersampling in spatial and temporal domain, which however gives us a single ray set modeling the whole animation sequence.

5.2 Estimating traversal probabilities from RRS

For each splitting plane candidate we determine the rays which intersect the two fragments of the bounding box of the subdivided node (see Figure 2). Probability of a ray passing through the left and right fragments of the bounding box is then estimated as:

$$p^{RDH}_{\{L|R\}} = \frac{|R_{\{L|R\}}|}{|R|}, \tag{3}$$

where $|R_{\{L|R\}}|$ is the number of rays intersecting the left and right fragments respectively, and $|R|$ is the number of rays intersecting the whole box.

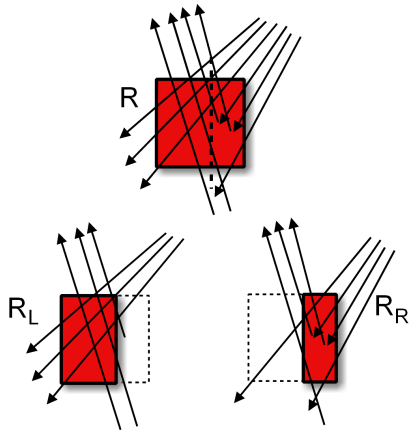

Figure 2: Illustration of the ray distribution heuristics.

From experiments we found out that using p^{RDH} directly in the cost function often leads to degenerated hierarchies. This happens due to the following two reasons: (1) the probability estimate is not perfectly accurate due to undersampling in RRS, (2) the greedy optimization based only on p^{RDH} has too strong focus towards RRS and thus is more prone to get stuck in a local minimum of the overall cost, i.e. the hierarchy degenerates. This happens for cases when majority of rays intersect the whole bounding box and hence the probabilities do not differ for wide range of a splitting plane position.

In order to solve these problems we blend the distribution of rays in *RRS* and uniform ray distribution using linear interpolation of of p^{RDH} and p^{SAH}:

$$p^{B}_{\{L|R\}} = w_r p^{RDH}_{\{L|R\}} + (1 - w_r) p^{SAH}_{\{L|R\}}, \tag{4}$$

where w_r is the weight of the ray distribution probability estimate. This weight aims to reflect the expectation that the estimate is correct for the given representative ray set. We used the following formula for computing w_r:

$$w_r = \alpha \cdot (1 - \frac{1}{1 + \beta * |R|}), \tag{5}$$

where α and β are user specified constants and $|R|$ is the number of rays intersecting the node to be split. This function gives more weight to *RDH* if $|R|$ is large and less weight if $|R|$ is small and we assume that these ray do not carry enough statistical information. For our experiments we used $\alpha = 0.9$ and $\beta = 0.1$. The weighting function of parameter $|R|$ is depicted in Figure 3.

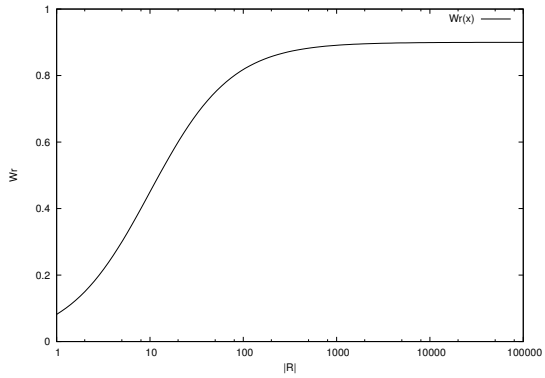

Figure 3: The weighting function w_r for $\alpha = 0.9$ and $\beta = 0.1$.

5.3 The cost function

The cost of a splitting plane candidate is computed similarly as in the SAH:

$$C = c_t + c_i(p^B_L|O_L| + p^B_R|O_R|), \tag{6}$$

where $p^B_{\{L|R\}}$ is the blended probability estimate of rays traversing the left and right child, respectively, c_t is the traversal cost and c_i is the intersection cost and $|O_{L|R}|$ is the number of objects intersecting the left and right children, respectively.

6 Implementation

This section describes several implementation details connected with the RDH.

6.1 Computing representative ray set

For testing the influence of the new heuristics we use the following strategy to compute the RRS. We first build a kD-tree using surface area heuristics. Then we subsample the actual set of rays to be cast using a regular pattern in the synthesized image. For example, using a 2x2 pattern we obtain a RRS with size of 1/4th of all rays to be cast. This subsampling method also handles secondary rays or shadow rays if these are traced. Note, that when using 1x1 pattern the RRS is equal to the actual ray set.

6.2 Efficient tracking of rays intersecting the box

For establishing the splitting plane we have to determine the set of rays intersecting left and right part of the corresponding bounding box.

We implemented a technique similar to the method for fast construction of kD-trees proposed by Wald and Havran [Wald and Havran 2006]. We use several data structures in order to make the algorithm efficient: First, we use a *ray segment*, which stores the entry and exit signed distances of a ray in the currently processed box, the reference to the whole ray, and the stack of the signed distances. Second, we use a *set of ray boundaries*, where each boundary contains a reference to the corresponding ray segment and a flag, whether the boundary is left, right, or lying on a splitting plane. Third, we use a *boundary index stack* to store which ray boundaries from the whole set of boundaries belong to the left, to the right, and to both of them. The boundary index stack hence stores 4 integers (leftmin, leftmax, rightmin, rightmax), which represent 3 intervals of ray boundaries when a splitting plane is placed: $\langle leftmin, leftmax \rangle$ for rays that belong only to the left box, $\langle leftmin, rightmax \rangle$ for rays that straddle the splitting plane, and $\langle rightmin, rightmax \rangle$ for rays that belong only to the right box. The data structures used in the algorithm are outlined in Figure 4.

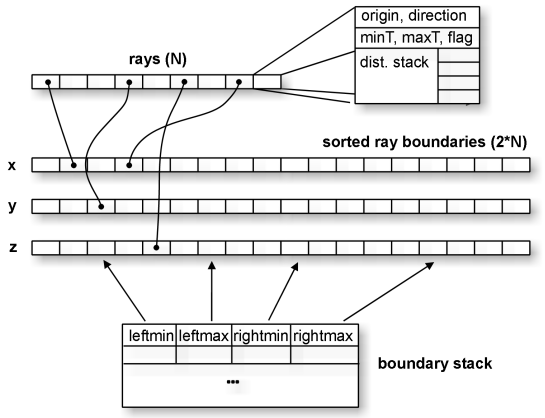

Figure 4: Data structures used for tracking the rays intersecting the currently subdivided node.

Before building the kD-tree with RDH we sort all three sets of initial ray boundaries, one set of ray boundaries for each axis. We assume that a kD-tree is constructed in a depth-first-search order, first constructing the left subtree of the current interior node. When a splitting plane position is decided and a left subtree should be constructed, we check the boundaries in the corresponding axis and mark all ray segments that have to be shortened (are cut by the splitting plane). The original ray boundaries are stored in the stack of signed distances associated with each ray segment data structure. For the marked rays we recompute the boundaries in a corresponding axis. All ray boundaries are then resorted by quicksort before we start to evaluate the RDH.

The number of rays intersecting the left and right fragments induced by the splitting plane is computed by counting the number of left and right boundaries on the left side and the right side of a splitting plane. The cost function for subsequent splitting plane candidates is evaluated incrementally by using the sweeping plane paradigm similarly to [Wald and Havran 2006].

When returning to an interior node after a left subtree has been constructed, we have to change the meaning of ray boundaries that lie on the splitting plane. These originally right boundaries for the left subtree now become the left boundaries for the right subtree to be constructed. The correct restore of boundary values is achieved by storing the depth of the node (together with the signed distance to the splitting plane) where the splitting plane has changed the boundary for each ray. The intervals of ray boundaries are also restored during return from both left child and right child.

In our experience this algorithm is relatively efficient as we store the information about original end of the ray segment only for the ray segments intersecting a splitting plane. The worst case space complexity for this algorithm is $O(N \cdot D)$, where the D is the maximum depth of a kD-tree and N is the number of rays. In practice the memory requirements are much lower since the case that all rays are always intersected by a splitting plane for all interior nodes from the root node to a leaf is very rare.

Another option how to evaluate ray distribution heuristics would be to use sampling of rays for preselected splitting plane positions similar to the approach of evaluating cost function with SAH [Hunt et al. 2006]. In that case we would need to test simultaneously both the rays from RRS and the objects. The method would have smaller accuracy as it limits the number of splitting planes tested, which was not desired for the analysis of our new heuristics.

7 Results

We implemented the ray distribution heuristics inside an optimized ray tracer. For the results we used a number of test scenes of various complexity and visibility characteristics. We tested ray casting with primary rays only and ray tracing including shadow and secondary rays. All images were computed in resolution of 513×513 pixels. The measurements were performed on a PC with 2.4GHz P4 CPU with 512KB L2 cache, 2GB RAM, and Linux OS.

7.1 Measurements overview

In our measurements we used 28 scenes in which we specified 52 different view points. For each measurement we evaluated the memory cost of the constructed kD-tree (M_{KD}), the number of intersections per ray (N_I), the average number of traversed nodes per ray (N_T), the construction time of the kD-tree (T_C), and the actual time of ray casting/ray tracing (T_R).

The tests were divided into three categories depending on the scene types and the rendering algorithm: (1) ray casting of low occlusion scenes, (2) ray tracing of low occlusion scenes, (3) ray casting of high occlusion scenes. Note that we did not test ray tracing for high occlusion scenes as we did not have meaningful definition of light sources for these scenes. In total we computed 78 measurements, where each measurement corresponds to a different image (either different scene, view point, or rendering algorithm). Several measurements are shown in Table 1. The scenes and their views correspond to snapshots shown in Figure 5. A visualization of the render cost functions for these views is shown in Figure 6.

In further tests we do not present the absolute values of the measurements, but use averages of the ratios of RDH and SAH methods for each measured parameter over all tests from the appropriate categories.

scene	method	M_{KD} [MB]	N_I [-]	N_T [-]	T_C [s]	T_R [s]
Ray casting, low occlusion scenes						
balls5	SAH	2.76	7.02	39.09	0.555	0.450
	RDH	4.04	5.60	38.67	1.677	0.444
Chevy	SAH	0.34	13.67	5.68	0.120	0.252
	RDH	2.14	3.78	8.66	0.489	0.196
jacks5	SAH	9.08	12.71	39.42	0.619	0.521
	RDH	8.10	11.17	33.65	1.569	0.488
rings17	SAH	11.14	13.87	74.56	1.006	1.027
	RDH	9.82	11.73	81.67	3.139	1.063
teapot40	SAH	8.91	4.19	29.26	0.859	0.346
	RDH	8.20	3.16	27.56	1.594	0.344
Ray tracing, low occlusion scenes						
balls5	SAH	2.76	16.27	30.91	0.465	2.974
	RDH	4.94	12.88	34.00	2.470	3.176
Chevy	SAH	0.34	37.48	8.54	0.131	2.296
	RDH	2.45	10.30	15.67	1.009	1.769
jacks5	SAH	9.08	21.95	57.33	0.619	3.620
	RDH	9.47	21.12	58.34	3.103	3.792
rings17	SAH	11.14	23.48	47.41	1.036	7.615
	RDH	11.62	21.59	56.93	5.693	8.626
teapot40	SAH	8.91	15.93	38.28	0.917	2.022
	RDH	9.36	11.82	39.36	2.579	2.002
Ray casting, high occlusion scenes						
arena v1	SAH	135.20	25.62	81.73	13.568	1.020
	RDH	126.68	19.11	61.80	17.660	0.825
arena v2	SAH	135.20	9.38	63.98	22.174	0.885
	RDH	109.61	6.51	56.27	21.267	0.494
Vienna v1	SAH	77.52	11.64	39.79	7.328	0.489
	RDH	77.98	8.64	33.95	10.691	0.419
Vienna v2	SAH	77.52	6.92	42.51	7.339	0.377
	RDH	63.43	5.14	32.12	9.773	0.315
Vienna v3	SAH	77.52	15.91	68.84	7.441	0.653
	RDH	72.13	8.44	52.81	10.026	0.489

Table 1: Experimental results for selected scenes and view points. The selection shows 5 representative scenes (views) for ray casting of low occlusion scenes, ray tracing of low occlusion scenes and ray casting of high occlusion scenes. M_{KD} is the memory cost of the constructed kD-tree, N_I is the number of intersections per ray, N_T is the average number of traversed nodes per ray, T_C is the construction time of the kD-tree and T_R is the time of ray casting/ray tracing.

7.2 Dependence on the size of RRS

In this test we analyzed the behavior of RDH in dependence on the size of RRS, i.e. number of rays used for RDH. We used different sampling patterns to obtain RRS and computed the ratios of measured values for RDH and SAH. The results are summarized in Table 2.

The results indicate that the RDH is surprisingly stable with lower number of samples. We expect that this behavior is also influenced by blending the ray distribution with the uniform ray distribution as described in Section 5.2. The information from rays is used in the higher levels of the tree whereas at the deeper levels the SAH is used with significant weight. The best results where achieved by using the actual rays to be cast as RRS (1x1 sampling pattern), but the results were quite stable even for lower sampling densities. In the remaining tests presented in the paper we used 4x4 subsampling as it exhibits reasonable performance rendering, while having a relatively low overhead on the construction time of the hierarchy.

RRS pattern	$\frac{M_{KD}^{RDH}}{M_{KD}^{SAH}}$	$\frac{N_I^{RDH}}{N_I^{SAH}}$	$\frac{N_T^{RDH}}{N_T^{SAH}}$	$\frac{T_C^{RDH}}{T_C^{SAH}}$	$\frac{T_R^{RDH}}{T_R^{SAH}}$
1×1	1.19	0.72	0.91	66.90	0.85
2×2	1.16	0.74	0.92	22.20	0.93
3×3	1.18	0.73	0.92	6.38	0.89
4×4	1.14	0.76	0.92	4.27	0.93
5×5	1.15	0.76	0.93	3.50	0.87
7×7	1.14	0.75	0.93	2.42	0.94

Table 2: Dependence of the method on the size of the RRS. The table shows a comparison of RDH and SAH for different sampling patterns of primary rays.

7.3 Behavior for different scenarios

In order to analyze the behavior of the method for different scenarios we grouped the tests according to scene types into two classes: ordinary scenes with low occlusion and complex architectural scenes with dense occlusion. In the first class we used ten common benchmark scenes for ray tracing generated (Standard Procedural Database [Haines 1987]) plus several other scenes. In the second class we tested two architectural scenes: city model and model of a sports stadium with furnished interiors. The results are summarized in Table 3.

scene	$\frac{M_{KD}^{RDH}}{M_{KD}^{SAH}}$	$\frac{N_I^{RDH}}{N_I^{SAH}}$	$\frac{N_T^{RDH}}{N_T^{SAH}}$	$\frac{T_C^{RDH}}{T_C^{SAH}}$	$\frac{T_R^{RDH}}{T_R^{SAH}}$
low occlusion	1.03	0.78	0.83	2.62	0.85
high occlusion	0.87	0.65	0.84	1.31	0.82

Table 3: Comparison of SAH and RDH on ray casting scenes with low occlusion and high occlusion.

7.4 Ray Casting vs. Ray Tracing

We compared the behavior of the method for ray casting (only primary rays) and recursive ray tracing. For the recursive ray tracing we casted shadow rays and secondary rays up to depth 3. The results of this test are summarized in Table 4.

method	$\frac{M_{KD}^{RDH}}{M_{KD}^{SAH}}$	$\frac{N_I^{RDH}}{N_I^{SAH}}$	$\frac{N_T^{RDH}}{N_T^{SAH}}$	$\frac{T_C^{RDH}}{T_C^{SAH}}$	$\frac{T_R^{RDH}}{T_R^{SAH}}$
ray casting	1.03	0.78	0.83	2.62	0.85
ray tracing	1.09	1.03	0.98	3.80	1.05

Table 4: Comparison of SAH and RDH when using ray casting and ray tracing algorithms.

The results indicate that on average we obtained 15% speedup for ray casting, but the performance of recursive ray tracing was actually decreased by 5% compared to the SAH.

In ray tracing the rays are more spread over the scene and thus they are closer to uniform distribution assumed by SAH. This explains why the RDH does not perform as good for this case as for ray casting. However, it is surprising that RDH doesn't provide greater speedup compared to SAH taking into account the fact that we consider the actual ray distribution including the occlusion. For ray casting the speedup might have some minor importance, however for ray tracing the current form of the method does not lead to practical results.

7.5 Ray based termination

In the last test we evaluated the influence of the ray based termination criterion on the constructed hierarchies. We used a threshold of one ray in order to decide whether to continue with the subdivision; i.e. if there is no ray intersecting the given box the subdivision is terminated for this node. The comparison of the RDH method and the RDH-R method which uses the ray based termination is shown in Table 5.

scenes	$\frac{M_{KD}^{RDH-R}}{M_{KD}^{RDH}}$	$\frac{N_I^{RDH-R}}{N_I^{RDH}}$	$\frac{N_T^{RDH-R}}{N_T^{RDH}}$	$\frac{T_C^{RDH-R}}{T_C^{RDH}}$	$\frac{T_R^{RDH-R}}{T_R^{RDH}}$
low occlusion	0.76	1.03	1.00	0.95	1.00
high occlusion	0.43	2.87	1.06	0.64	1.98

Table 5: Comparison of RDH-R with additional ray based termination criterion and the RDH with ordinary termination criteria.

We can observe that for low occlusion scenes the ray based termination saved about 25% of storage on average, while not increasing the average rendering time. For high occlusion scenes the memory savings where even more significant (57%), however the rendering time almost doubled on average. We can observe that we significantly increased the average number of intersections. This indicates that there have been parts of the scenes which were not covered by the sample rays, but were penetrated by significant number of actual rays that have been cast.

8 Conclusion and Future Work

We presented a method for changing the surface area heuristics by using knowledge about the actual distribution of rays in the scene. The kd-trees constructed using the proposed ray distribution heuristics achieve on average a slightly better performance than the traditional surface area heuristics for ray casting. The results indicate that for ray tracing the method actually does not provide any measurable benefit.

An important result of the paper is the observation that taking into account the actual ray distribution does not really help in the construction of the ray tracing hierarchy assuming the traditional greedy top-down construction algorithm. Thus, the actual practical application of the method to interactive ray tracing or ray tracing of animated sequences is a subject of future work. In the future we also want to study the possibility of using global optimization techniques together with the new heuristics.

Acknowledgements

This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research program MSM 6840770014 and LC-06008 (Center for Computer Graphics), and the Aktion Kontakt OE/CZ grant no. 2009/6. The Arena scene is a courtesy of Digital Media Production a.s.

References

CHANG, A. Y.-H. 2004. *Theoretical and Experimental Aspects of Ray Shooting*. PhD thesis, Politechnic University, USA.

FUJIMOTO, A., TANAKA, T., AND IWATA, K. 1986. ARTS: Accelerated Ray Tracing System. *IEEE Computer Graphics and Applications 6*, 4, 16–26.

GLASSNER, A. S. 1984. Space Subdivision For Fast Ray Tracing. *IEEE Computer Graphics and Applications 4*, 10 (Oct.), 15–22.

GLASSNER, A. 1989. *An Introduction to Ray Tracing*. Morgan Kaufmann.

GOLDSMITH, J., AND SALMON, J. 1987. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications 7*, 5 (May), 14–20.

HAINES, E. A. 1987. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications 7*, 11 (Nov.), 3–5. Available from `http://www.acm.org/pubs/tog/resources/SPD/overview.html`.

HAVRAN, V., AND BITTNER, J. 1999. Rectilinear BSP Trees for Preferred Ray Sets. In *Proceedings of SCCG'99 (Spring Conference on Computer Graphics)*, 171–179.

HAVRAN, V., HERZOG, R., AND SEIDEL, H.-P. 2006. On the Fast Construction of Spatial Data Structures for Ray Tracing. In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, 71–80.

HAVRAN, V. 2000. *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague.

HUNT, W., MARK, W. R., AND STOLL, G. 2006. Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic. In *2006 IEEE Symposium on Interactive Ray Tracing*, IEEE.

MACDONALD, J. D., AND BOOTH, K. S. 1990. Heuristics for ray tracing using space subdivision. *Visual Computer 6*, 6, 153–65.

SAMET, H. 2006. *Foundations of Multidimensional and Metric Data Structures*. Morgan Kaufmann.

SLUSALLEK, P., SHIRLEY, P., WALD, I., STOLL, G., AND MARK, B. 2005. SIGGRAPH 2005 Course on Interactive Ray Tracing #38.

SOLOMON, H. 1978. *Geometric Probability*. J.W. Arrowsmith Ltd.

WALD, I., AND HAVRAN, V. 2006. On building fast kd-trees for ray tracing, and on doing that in o(n log n). In *Proceedings of IEEE Symposium on Interactive Ray Tracing 2006*, 61–69.

WALD, I., BOULOS, S., AND SHIRLEY, P. 2007. Ray Tracing Deformable Scenes using Dynamic Bounding Volume Hierarchies. *ACM Transactions on Graphics 26*, 1.

WHANG, K. Y., SONG, J. W., CHANG, J. W., KIM, J. Y., CHO, W. S., PARK, C. M., AND SONG, I. Y. 1995. Octree-R: an adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics 1*, 4 (Dec.), 343–349. ISSN 1077-2626.

WHITTED, T. 1979. An improved illumination model for shaded display. *Computer Graphics 13*, 2 (Aug.), 14–14.

WOOP, S., MARMITT, G., AND SLUSALLEK, P. 2006. B-KD Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes. In *Proceedings of Graphics Hardware*.

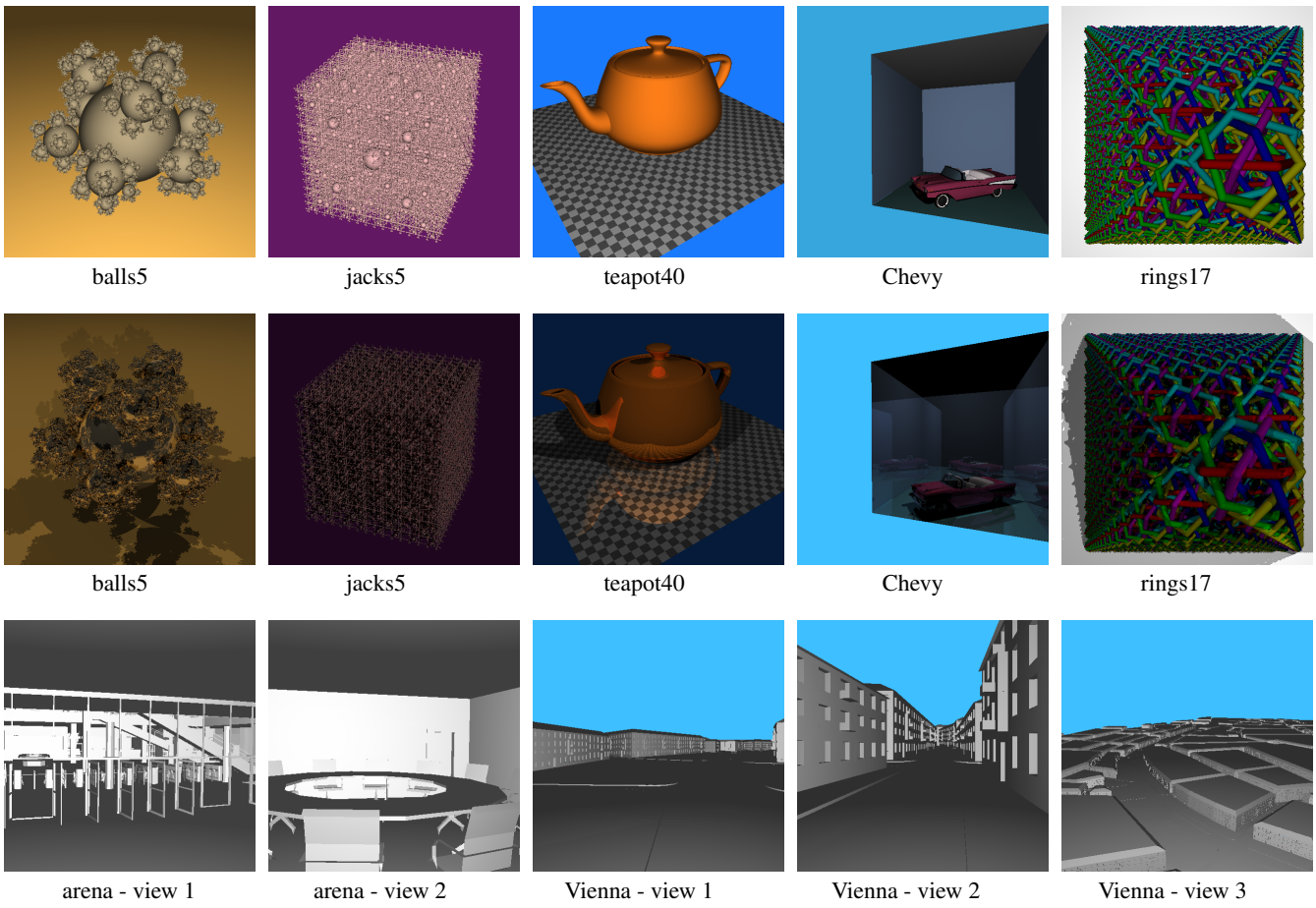

Figure 5: Snapshots of selected representative scenes and their view points. Top row: ray casting of scenes with low occlusion. Middle row: ray tracing of scenes with low occlusion. Bottom row: ray casting of scenes with high occlusion.

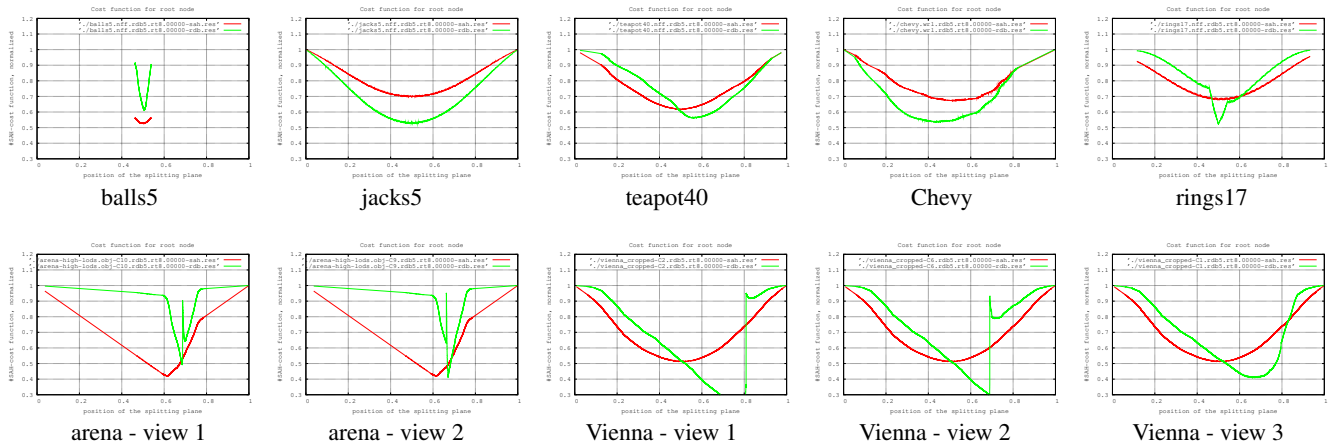

Figure 6: Comparison of the cost functions for root nodes of selected scenes. The red curve corresponds to SAH and the green curve to the RDH. Note that in most cases the RDH provides a steeper local minimum, which is slightly different from the position of the minimum determined by SAH.