# On Fast Construction of Spatial Hierarchies for Ray Tracing

Vlastimil Havran Robert Herzog
Hans-Peter Seidel

MPI–I–2006–4–004 June 2006

**Author's Address**

Vlastimil Havran  Robert Herzog  Hans-Peter Seidel
Max-Planck-Institut für Informatik
Stuhlsatzenhausweg 85
66123 Saarbrücken
Germany
{havran, rherzog, hpseidel}@mpi-inf.mpg.de

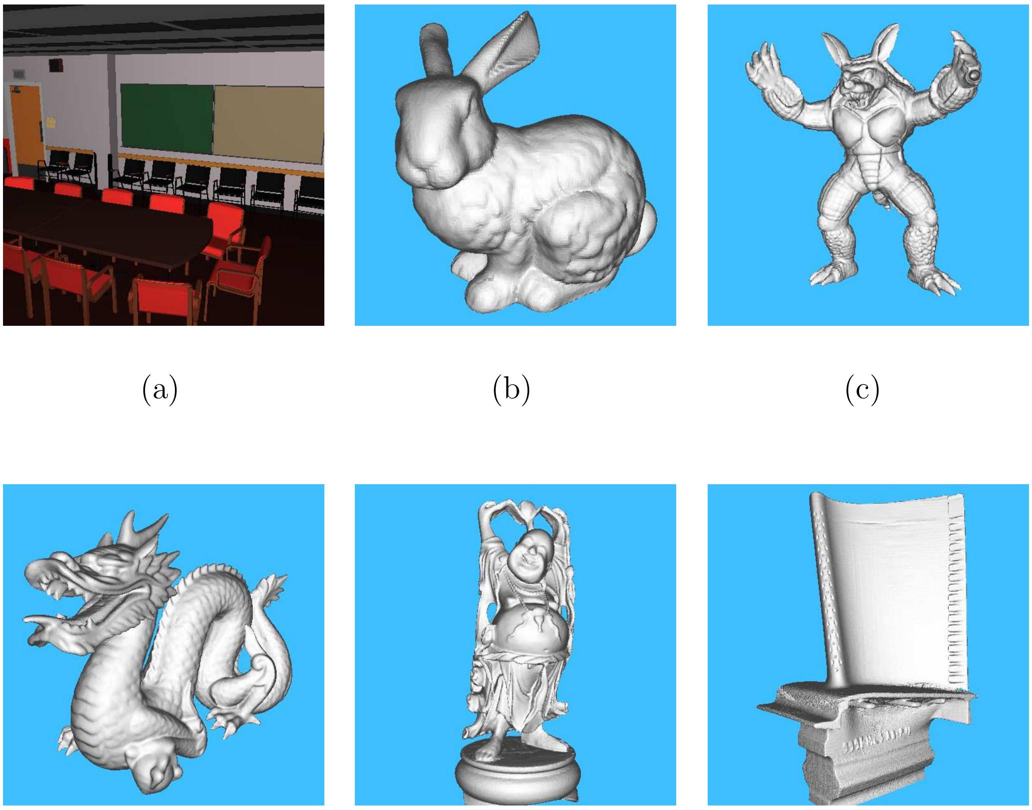
## Abstract

In this paper we address the problem of fast construction of spatial hierarchies for ray tracing with applications in animated environments including non-rigid animations. We discuss properties of currently used techniques with $O(N \log N)$ construction time for kd-trees and bounding volume hierarchies. Further, we propose a hybrid data structure blending between a spatial kd-tree and bounding volume primitives. We keep our novel hierarchical data structures algorithmically efficient and comparable with kd-trees by the use of a cost model based on surface area heuristics. Although the time complexity $O(N \log N)$ is a lower bound required for construction of any spatial hierarchy that corresponds to sorting based on comparisons, using approximate method based on discretization we propose a new hierarchical data structures with expected $O(N \log \log N)$ time complexity. We also discuss constants behind the construction algorithms of spatial hierarchies that are important in practice. We document the performance of our algorithms by results obtained from the implementation tested on nine different scenes.



## Keywords

Ray shooting, ray tracing, animation, time complexity, hierarchical data structures, spatial sorting, approximate algorithms



(a) (b) (c)

(d) (e) (f)

Figure 1.1: Six static scenes used for testing of our algorithm. (a) Conference Room (b) Bunny (c) Armadillo (d) Dragon (e) Buddha (f) Blade.

# 1 Introduction

Nowadays, thanks to the algorithmic progress in spatial *data structures* (DS) and increasing performance of computer hardware, we achieve interactive and real time ray tracing of primary and shadow rays [28] for static scenes. This requires efficient hierarchical DS that results in the logarithmic complexity of ray tracing, such as kd-trees [20, 11, 28] built with *surface area heuristics* (SAH). The preprocessing time for hierarchical DS has been shown to be $O(N \log N)$.

In practice, kd-trees have been used in interactive ray tracing for static walkthroughs [35]. So far only special settings of dynamic and semi-dynamic scenes [42, 41] where only a small portion of objects is moving or object instantiation is used, have been successfully addressed. Fully realtime or interactive preprocessing of spatial hierarchies for *non-rigid animations* also referred to as unstructured motion is a difficult problem. It was achieved only for a small number of objects ($\approx 10,000$ for year 2005). We lift these limitations using several techniques. First, we show how *spatial kd-trees* (SKD-trees) [23] can be used for ray tracing instead of classical kd-trees. Second, we show that it is convenient to combine the spatial kd-trees with bounding volumes (BVs), in a sparse way, resulting in a hybrid DS with $O(N \log N)$ preprocessing time and $O(N)$ storage.

We also lift the assumption on $O(N \log N)$ time complexity for preprocessing to $O(N \log \log N)$ by discretization in the space of splitting planes. Radix sort (bucket/distribution sort etc.) [15] that relies on the limited precision of input data achieves $O(N)$ time complexity for sorting instead of $O(N \log N)$ based on comparisons. Similarly, we can construct an efficient spatial hierarchy for ray tracing in a discrete setting with expected time complexity $O(N \log \log N)$ time instead of $O(N \log N)$. The construction assumes that the representation of axis aligned bounding boxes tightly encompassing the objects is restricted to $b$ bits. Further we assume that objects do not vary much in size and that distribution of objects is not highly skewed.

This paper is further organized as follows. In Chapter 2 we describe previous work on ray tracing dynamic scenes. In Chapter 3 we provide an algorithmic consideration for efficient algorithms. In Chapter 4 we recall the SKD-trees developed for databases and the DS built in $O(N \log \log N)$ time in the community of theoretical computer science. In Chapter 5 we describe the discretization for evaluation of SAH in one-dimensional setting. In Chapter 6 we describe the hybrid tree combining a spatial kd-tree with BVs. In Chapter 7 we describe how the discretization can be extended to three dimensions using a 3D summed area table. In Chapter 8 we describe briefly the modifications to the traversal algorithm. We present the experimental results in Chapter 9. We conclude the paper with a summary of contributions and future work.

# 2 Previous Work

In this chapter we briefly recall the previous work on ray tracing that relates to our paper.

## 2.1 Dynamic Data Structures for Ray Tracing

While there has been much effort devoted to optimization of ray tracing techniques for static scenes (surveys in [4, 11, 34, 2]), the DS for animated scenes have not been investigated in depth. In the early work Parker et al. [25] allow a few objects to be moved interactively. The ray tracing for a single ray is decomposed into two phases. In the first phase the intersection with static objects is computed using standard spatial DS. In the second phase the ray is intersected with dynamic objects. Then the results of the two phases are combined together taking the closest intersection. This method allows rendering of only a few simple dynamic objects in practice. Later, Reinhard et al. [27] show how to extend the grid-like structures duplicating the boundary of grids virtually in order to enlarge the spatial extent accessible by objects. More recently, Lext et al. [19] proposed a benchmark for ray tracing of animated scenes containing three data sets. Although they propose a classification of motion types and provide a very good motivation for such a benchmark, they do not describe any particular algorithm to solve the problem. Their classification of motion involves two types. First, *hierarchical motion* is generated from a hierarchical scene graph and hence it preserves some hierarchical spatial relationship among objects from frame to frame. Second, *unstructured motion* as more general case corresponds to non-rigid data animation where fewer or no assumptions can be made. The proposed benchmark data containing three scenes addresses different issues for the two types of motion and several challenges that new DS designed for ray tracing of animated scenes should solve.

Wald et al. [42] motivated by the algorithm of Lext and Möller [17] show a technique with two-level DS based on kd-trees exploiting the modeling hierarchy. The bottom level contains single individual objects consisting of many primitives. Such objects have their own locally precomputed kd-trees and can be animated in the global space by a single transformation matrix with optional instantiation over basic object data. The upper level kd-tree is built over bottom level objects enclosed by boxes for every frame. If the number of objects at the upper level is small (in order of hundreds to thousands), then the upper-level kd-tree construction achieves interactive rates. However, this solution is not suited for unstructured motion of individual object primitives. In the same year, Szecsi et al. [37] propose an interesting extension to the "sequential" method of Parker et al. They suggest to construct two kd-trees, one kd-tree over static data and one kd-tree over dynamic objects. Instead of traversing the trees in sequential order and later combining results, they propose to traverse the two trees simultaneously by a special traversal algorithm over two trees. Another paper by Adams et al. [1] focuses on the use of spatial and temporal coherence for efficient update of bounding sphere hierarchy for deforming point-sampled surfaces. A discussion on several important issues and motivation for dynamic DS was presented in [41].

## 2.2 Collision Detection and Visibility

Another related field that deals with animated geometric data is collision detection and general visibility algorithms. The focus of so far presented techniques has been the update of DS (mostly BVHs) from frame to frame. For a survey of techniques on collision handling we refer to the excellent tutorial [40]. The most relevant techniques in visibility deal with the update of dynamic data structures [5, 36]. More recently, Shagam and Pfeiffer [32] propose to use dynamically updated octrees in the context of visibility culling. We do not discuss the details of these techniques that are only related to the update of DS and not to the construction of the hierarchies from the scratch as presented in this paper.

# 3 Algorithmic Motivation

Below we briefly discuss algorithmic issues of spatial hierarchies relevant to unstructured motion. If we have $N$ moving objects with unstructured (=mutually independent) motion, then the lower bound to construct or reconstruct DS is $\Omega(N)$. This can be achieved for grid-like DS such as uniform subdivisions. The preprocessing time for grid-like DS is $O(N \cdot P)$, where $N$ is the number of objects, and $P$ is the average number of references per one object in grid cells. Since a single object can reside in more than one cell, for large objects it can result in high preprocessing time and hence it is highly dependent on the scene. More importantly, for skewed distributions the performance of grid-like structures is not competitive with kd-trees built with SAH [38, 12]. This could perhaps be alleviated by recursive grids [13]. However, it seems to be generally difficult to predict the memory requirements and hence the preprocessing time of recursive grids [4, 12].

In collision detection the dynamic update of bounding volume hierarchies (BVHs) is addressed, such as dynamic collision of unstructured motion [16]. This is efficient for collision detection where the query domain corresponds to an expanding sphere. For ray tracing however, where the query is formed by a line, different problems can be expected for dynamic updates. The dynamic updates are localized on lower levels of the hierarchy similarly to collision detection, at best only in leaves, where objects are moved to new positions. The changes are propagated upwards in the hierarchy. By repetitive local updates the global structure can become potentially less and less efficient, since upper levels of the hierarchy do not reflect the changes as much as efficiently as the construction from the scratch. Therefore, the performance of updated DS can degrade with repetitive updates due to the lack of update in the higher levels of the hierarchy. The moment when a subtree rooted at a particular node need to be rebuild due to efficiency reasons has to be recognized. This requires to keep the auxiliary information in nodes about hierarchy rooted in the nodes to decide on the right moment to rebuild a hierarchy completely. This in principle can cause stalls from time to time during rendering if such an update is located close to the root node.

Second problem for unstructured motion is the time complexity of such a rebuilding algorithm. For hierarchical (i.e. structured, organized) motion or scenes with only a small part of moving objects the update for one object (object primitive) in a hierarchy can be computed in $O(\log N)$ time. This is advantageous only for scenes where we need to update only small number of objects. If the number of moving objects is $P$ then hence $P \ll N \longmapsto P \log N \ll N \log N$. If we move majority of objects independently then the update of DS starts to be a serious bottleneck since $P \approx N$. It could perhaps be alleviated by multiple processing of objects to be updated. However, such algorithm has to be designed very carefully, since updating a hierarchy without knowing new positions of the neighboring objects makes the problem rather difficult. A trivial algorithm that updates all objects one by one by locally rebuilding the hierarchy (including deferred rebuilding) for each change has the time complexity $O(N \log N)$. This has the same time complexity as rebuilding the hierarchy from the scratch. However, the algorithm with updating the hierarchy does not guarantee the same efficiency as the algorithm completely rebuilding DS for every frame.

Based on the analysis above we argue that one viable algorithm for unstructured motion is an algorithm that can construct the DS very efficiently from scratch for every frame of the animation. Such a solution, if it exists, avoids bookkeeping the data in the nodes of the tree to be updated and guarantees good performance for every frame of the animation. Clearly, it can also be used for static scenes to decrease the preprocessing time. Therefore in order to overcome an algorithmic complexity we address the fast construction of DS while acceptably decreasing their performance during searching by a discretization of the problem setting.

# 4 Algorithmic Background

In this chapter we describe algorithmic preliminaries developed in the field of databases and theoretical computer science. We believe that below described concepts are not well known in the computer graphics community. We consider necessary to review them to give better motivation for our algorithm design.

## 4.1 SKD-trees and Related Data Structures

*Kd-trees* were designed by Bentley [3] as underlying DS for efficient indexing of multidimensional point data. A tree is constructed recursively, having interior nodes and leaves embedded in the axis-aligned splitting planes.

In order to address efficient indexing of non-point data Ooi et al. [23] have proposed an extension to kd-trees called *spatial kd-trees* (SKD-trees). Several other extensions to kd-trees were proposed in several papers from other authors, we refer to surveys [7, 24]. The proposal of SKD-trees is similar to BVHs [29, 14] and R-trees [10], but SKD-trees are more memory efficient. Instead of implementing a hierarchy by representing a single splitting plane in an interior node as in kd-trees, the interior nodes of SKD-trees contain *two splitting planes*. It subdivides the original region into two either overlapping or disjoint subregions. The concept of a SKD-tree node is shown in Figure 4.1. The closest concept to SKD-trees is Kay and Kajiya's method of slabs [14] that are used to represent BVs in BVHs.

During the subdivision step every object is fully contained in the spatial extent in one of the two child nodes. Every interior node describes both axis-aligned splitting planes and references to children. The tree is constructed recursively in a top down fashion. The recursion is terminated by a construction of a leaf node containing the reference to a single object residing in the spatial extent of the leaf. We base our ray tracing algorithm on SKD-tree nodes and combine them with BV nodes.

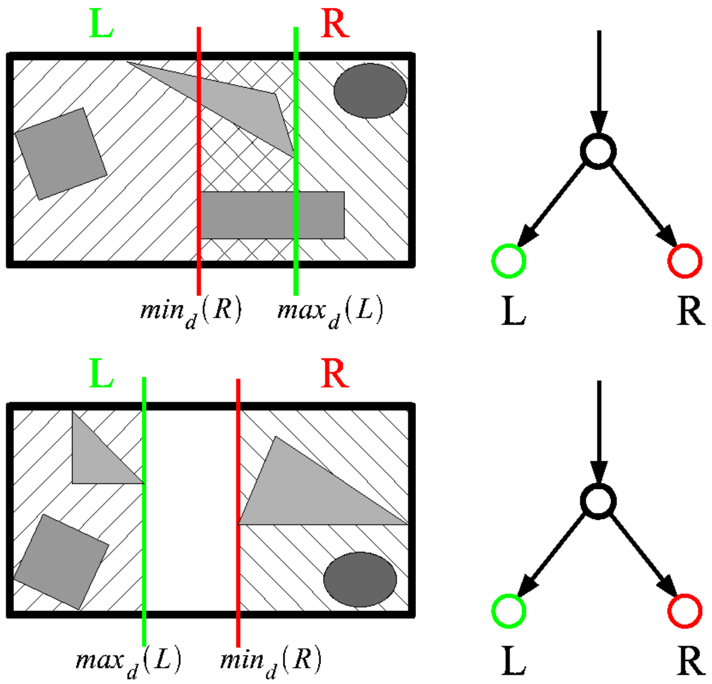



Figure 4.1: An organization of splitting planes inside SKD-node. Two splitting planes in the node defines spatial extents of two children (left) the child nodes overlap (right) the child nodes are disjoint.

We would like to mention that there are many variants to kd-trees and BVHs (and hence R-trees [10]) generalizing the concept of a hierarchy in various ways. For ray tracing two basic concepts are used. The first one allows *overlapping of spatial regions* (BVHs, R-trees, SKD-trees etc) by building up the hierarchy over the data, often referred to as *object hierarchies*. The second concept forms strictly disjoint spatial regions, using for example a set of subdividing hyperplanes (kd-trees, octrees, grid-like data etc.). This is often referred to as *spatial subdivision*. A few hybrid methods combining the two concepts were also proposed. Since this is a very broad topic, we refer to excellent surveys [30, 7, 24].

## 4.2 Fast Construction of Spatial Hierarchies

Motivated by the linear time complexity of multipole algorithm in computational physics Reif, Tate, and Xu [26, 39] have addressed an important related problem, namely fast construction of spatial decompositions for point data to solve closest pair, $k$-nearest neighbor, and $n$-body problems. Assuming that the input *point data* are represented with limited precision, namely $O(c \log N)$ for all coordinates of a single point ($c = const$) in $D$-dimensional space, they propose a method for construction of spatial subdivisions in $O(D^2 N \log\log N)$ time. Since we have been motivated by their approach, we describe their method briefly to show the differences later. In the first step a complete regular *support tree* of height $h$ (so all leaves at the depth $h$, spatial median, cyclic order of axes for splitting planes) is constructed. In the second step all the input points represented in a finite representation are mapped to the leaves of the support tree in constant time. In the third step, the partial tree is processed by moving from the leaves to the root merging empty leaves until each leaf contains at least a single point. This bottom-up merging continues until the whole tree is created. For leaves containing more than one point the algorithm recurses and new partial tree is supported and linked to the parent partial tree. By carefully selecting the height of the supporting tree for every level of recursion and performing the search in the tree by bitwise operations in constant time for each point, the time complexity is shown to be $O(N \log\log N)$. For details, experimental results, and the related work on this subject we refer to the excellent exposition in [26, 39].

# 5 SKD-Trees Constructed with Discretized SAH Computation in 1D

In this paper we focus on SKD-trees described in Chapter 4.1. Obviously, for SKD-trees we could use the spatial median strategy similar to other data structures [14, 44, 33]. However, it has been shown on kd-trees that spatial median results in inferior performance of ray tracing for skewed distributions compared to the methods based on the SAH cost model [20, 11]. Therefore we apply the cost model based on SAH to construct efficient SKD-trees. While a precise construction algorithm with time complexity $O(N \log N)$ could be used, motivated by the method of Reif and Tate [26] we discretize the evaluation of a cost function along the axis to be subdivided.

In this chapter we describe the construction of SKD-tree based on the cost model using surface area heuristics. This model was described by Goldsdmith and Salmon [8]. They use integral geometry measure to estimate the probability of rays intersecting a spatial region such as a box. This is usually referred to as surface area heuristics (SAH). Using the cost model based on SAH they construct BVH in a randomized way inserting objects one by one, changing the shape of the constructed hierarchical tree on the fly. It has been shown independently by experiments that this method leads to inferior performance compared to top-down construction [21, 22, 12].

A geometric probability of shooting rays has been applied later with success to kd-trees [20]. Such a kd-tree is constructed in top-down fashion where a cost model based on SAH is evaluated to select the splitting plane position in interior nodes. This method has been shown to produce efficient kd-trees for ray tracing scenes with uniform as well as skewed distribution of objects [20, 11]. An efficient construction algorithm based on the plane-sweep paradigm with time complexity $O(N \log N)$ for kd-trees has been described in [11, page 80] and detailed in [43].

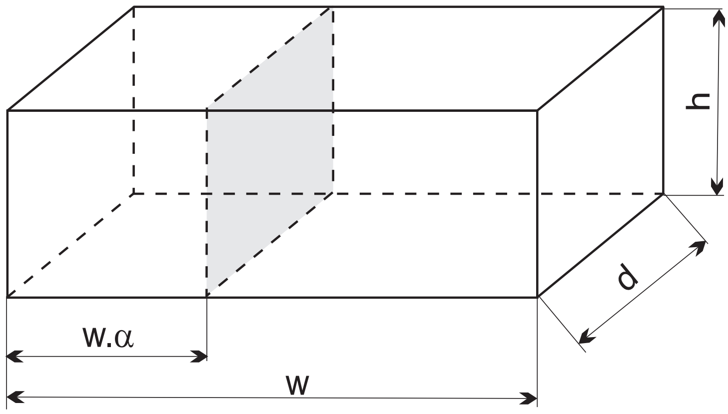



Figure 5.1: A spatial extent of the kd-tree node of width $w$, height $h$, and depth $d$ split into two spatial regions at $w \cdot \alpha$.

Below we consider a geometry depicted in Figure 5.1. To create an interior node of a kd-tree we assume a box of size $w \times h \times d$ containing $N$ objects which is subdivided along $w$ by a splitting plane at the position $\alpha \cdot w$, $\alpha \in (0, 1)$. If $N_L(\alpha)$ and $N_R(\alpha)$ is the number of objects on the left respectively right of the splitting plane, and $N_S(\alpha)$ objects are straddling the splitting plane ($N = N_L(\alpha) + N_R(\alpha) + N_S(\alpha)$), we can formulate the *quality of a subdivision step* $q(\alpha)$ for a splitting plane at $\alpha$ by computing the ratio of the cost after and before subdivision in the following way [11, page 68]:

$$\begin{aligned}
SA &= (w \cdot (h + d) + h \cdot d), \\
C_L(\alpha) &= (N_L(\alpha) + N_S(\alpha)) \cdot (w \cdot \alpha \cdot (h + d) + h \cdot d)/SA, \\
C_R(\alpha) &= (N_R(\alpha) + N_S(\alpha)) \cdot (w \cdot (1 - \alpha) \cdot (h + d) + h \cdot d)/SA, \\
q(\alpha) &= (C_L(\alpha) + C_R(\alpha))/N,
\end{aligned}$$

where $C_L(\alpha)$ and $C_R(\alpha)$ is a linear estimate of the cost of the left subtree and right subtree to be constructed, respectively. A typical graph of $q(\alpha)$ for large $N$ is shown in Figure 5.2. The local greedy heuristics with the cost model based on SAH can be used for all hierarchical DS constructed in a top-down fashion. This was used for example to optimize octrees [45]. Recently, Mahovsky [21, page 58] proposed to use the cost model also for BVH, but neither experimental nor theoretical results are presented.

We have investigated if a cost function can be computed in a discretized setting. We have found out to be convenient for a larger number of objects. This has two reasons. First, the cost function shows a single global minimum for a large number of objects as discontinuities are smoothed away due to a large number of overlapping objects. Second, the cost function is relatively

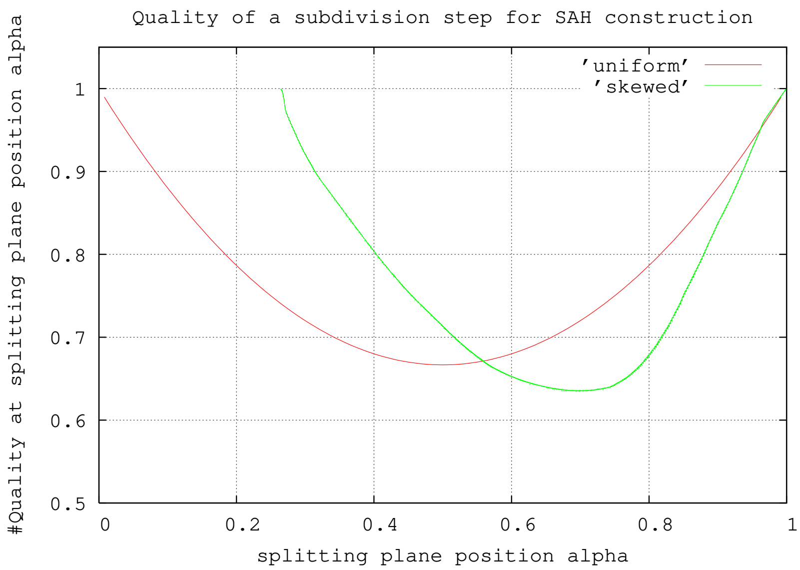



Figure 5.2: A graph with the typical behavior of $q(\alpha)$ representing the cost reduction for subdivision at position $\alpha$ for a large number of objects (N=100,000) based on real data. Although in principle the cost function is discontinuous, for many small objects the discontinuities are smoothed away. (red) Uniform distribution. (green) Skewed distribution.

smooth and flat around the global minimum of the function because of its integral form. In the opposite for a small number of objects the cost function shows strong discontinuities and the selection of a splitting plane has to be handled more carefully.

To implement discretization we create $M$ buckets subdividing the spatial extent along the width of the box $w$. Each bucket contains the representation of three axis aligned bounding boxes $A$, $A_L$, and $A_R$ and the number of objects in the bucket. In the first step we initialize all bounding boxes and counters in all buckets. In the second step every object is one by one inserted into a bucket $i$ based on the *centroid of the bounding box of the object* updating $A(i)$ ($i \in \langle 0, M-1 \rangle$). The use of the box centroids is necessary since objects are non-point data and we need to assign them to buckets in a unique way. In the third step we carry out a plane-sweep from right to the left to compute the minimum box on the right of each bucket $b$ as $A_R(b) = \bigcup_{m=b}^{M-1} A(m)$, so $A_R(b) = A(b) \bigcup A_R(b+1)$ and $A_R(M-1) = A(M-1)$. In the fourth step we compute plane-sweep operation from left to the right to compute the minimum box on the left for bucket $b$ as $A_L(b) = \bigcup_{m=0}^{b} A(m)$, so $A_L(b) = A(b) \bigcup A_L(b-1)$ and $A_L(0) = A(0)$.

This allows to compute in $O(M)$ time a tight axis aligned box for all the objects assigned to the bucket $b$ in range $\langle 0, b \rangle$ and $\langle b, M-1 \rangle$. The number of objects $N_L(b)$ associated with all the buckets on the left of bucket $b$ is summed during the plane-sweep operation from left to right of bucket $b$.

Then we can evaluate the cost function in for $M-1$ positions. We computed a spatial extent (box) on the left of a bucket $b$ in $A_L(b)$ and the number of objects in $N_L(b)$. We also computed a box on the right of $b$ in $A_R(b)$ and the number of objects on the right as $N - N_L(b)$. Then we evaluate the cost function $C(b)$ based on SAH as follows:

$$\begin{aligned} C_L(b) &= N_L(b) \cdot SA_L(b)/SA, \\ C_R(b+1) &= (N - N_L(b)) \cdot SA_R(b+1)/SA, \\ C(b) &= C_L(b) + C_R(b+1) \end{aligned} \tag{5.1}$$

where $SA_L(b)$ is the surface area of the tight box $A_L(b)$ and $SA_R(b)$ is the surface area of the box $A_R(b)$. We evaluate the cost function for $M-1$ positions. We select such $b_m$ that results in the minimum value of the estimated cost $C(b_m)$. Spatial extents for the two children along axis $a$ to be subdivided create two values for the SKD-tree node: $Max_a(A_L(b_m))$ for the left subtree and $Min_a(A_R(b_m))$ for the right subtree.

The construction of a SKD-tree is carried out recursively in a top down fashion, performing in each step distribution of objects to buckets and computing the split into two child nodes. This results in $O(N \log N)$ time complexity for construction. Even if it is equivalent to $O(N \log N)$ algorithm for kd-trees [11], there are different multiplicative factors hidden in big O notation. First, for SKD-trees there is no fragmentation during splitting, so the number of references to objects is strictly $N$. For kd-trees the number of references to objects is $r \cdot N$ ($r \geq 1$), according to our experimental evaluation approximately in range $1.8 - 3.5$ in dependence on the scene and used termination criteria. This influences the number of evaluations of the cost function. Second, $O(N \log N)$ algorithms for kd-trees require to split three lists to six lists based on the position of the splitting plane plus evaluation of a cost model. This is by factor of two to three times more costly than a simple bucketing described above.

Clearly, the discretized evaluation of a cost function results in a smaller precision. It means that less efficient spatial hierarchies are constructed due to discretization. We show that this trade-off is acceptable if our concern is total running time of the application (construction of DS + rendering based on ray tracing using the data structures). Second source of inefficiency is overlapping of SKD-nodes. We have to traverse all spatially overlapping nodes to ensure that we find the nearest object. More traversal steps could result in decreased performance. This is partially compensated by a smaller height of SKD-trees compared to kd-trees and by single references to objects in the leaves of hierarchy.

# 6 H-trees = SKD-trees + BVs

During experiments with SKD-trees described above we have found out that SKD-nodes do not separate empty space sufficiently. The performance of ray tracing with a tree containing only SKD-nodes and leaves is far behind the performance of kd-trees, by factor about 1000% for some scenes we tested. Below we propose the combination of SKD-trees and BVHs called H-tree. Its efficiency is then comparable to kd-trees.

## 6.1 H-trees Description

The storage of traditional BVH nodes requires to store axis aligned bounding box (six values). This has two disadvantages. First, storing the whole box for every interior node is relatively memory expensive (a single interior node takes 32 Bytes if pointers and floating point values are represented by 4 Bytes). Second, the ray traversal algorithm for BVHs requires to compute up to six ray plane intersections to determine if a ray intersects the box. This intersection test is relatively costly (kd-trees require to compute a single ray-plane intersection and to evaluate only two conditions). During our study of pure SKD-trees for ray tracing we have found out that a pure SKD-tree node does not bound enough tightly the objects associated with the nodes. Therefore we propose a hybrid DS combining SKD-tree nodes and bounding volume nodes in a hybrid hierarchy. Our bounding volume nodes are in principle equivalent to cutting off empty spaces in kd-trees. When empty space is cut away, the ray traversal algorithm can proceed such spatial regions more efficiently.

We distinguish between two current spatial extents associated with a node $\nu$ represented by a box. First, we define the *enclosing spatial box* $B^E(\nu)$ as the minimum box enclosing all the objects associated with $\nu$. Second, during traversal a *traversal box* $B^T(\nu)$ is formed by limiting the spatial extent of the node by traversing all the interior nodes on the path from root to node $\nu$.

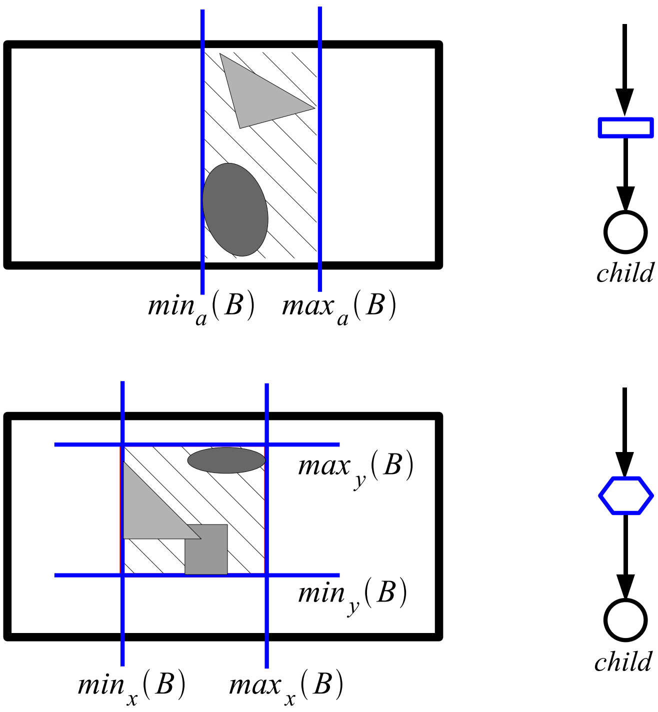



Figure 6.1: Bounding volume nodes used in our H-tree depicted in 2D space, (top) BV2: two-plane bounding node, (bottom) BV6: six-plane bounding node.

Obviously, $\forall \nu : B^E(\nu) \in B^T(\nu)$ forms a necessary condition for any correct spatial hierarchy. Based on relation between $B^E(\nu)$ and $B^T(\nu)$ and assuming axis-aligned planes to be used in the bounding primitives, we can enumerate the options for *bounding nodes* representing a bounding volume (BV):

- BV1: a single plane splitting node as in a kd-tree, used to cut away an empty space on either side of the plane.
- BV2: a two-plane bounding node defining a spatial extent of objects along one axis. This is complementary to the definition of the spatial tree node and corresponds to slabs [14] as infinite region between two parallel planes. The concept of the node is shown in Figure 6.1 (top).
- BV6: a six-plane bounding node (traditional axis-aligned bounding box), shown in Figure 6.1 (bottom).
- BVO (others): we can form a bounding volume node by any combination of six axis-aligned bounding planes. In total having one, two,

three, four, five, or six axis aligned planes (this includes options in above mentioned categories BV1, BV2, and BV6).

In total we could have 63 types of bounding nodes as it can be shown by variational calculus. However, using all the nodes would complicate the algorithm implementation. Such approach though theoretically viable is likely to have small practical merit. In order to keep both construction and traversal algorithm simple, we have implemented only variants BV2 and BV6 simultaneously. As a result, we have an H-tree that can contain these nine types of nodes:

- a leaf node containing a pointer to a single object,
- a six-plane bounding node – traditional bounding box node, a single child is referenced in DFS order,
- a two-plane bounding node – a bounding box node with two planes, in total three nodes (one for each axis $x, y, z$),
- a SKD-tree node, in total three nodes (one for each axis $x, y, z$),
- a link node from a current address to another address to be used for a fixed size memory allocator, if we allocate the memory in chunks of fixed size.

## 6.2 Memory Layout of H-trees

The important aspect is the layout of nodes in the memory of a computer. Such alignment in main memory, L2 cache, and L1 cache plays an important role for the performance in current modern processors [28] and [11, page 125].

A design of H-tree can use depth-first-search (DFS) ordering of nodes in the memory. It allows to align the nodes on the 16 Bytes boundary assuming 32-bit computer architecture. The SKD-tree nodes and leaves take 16 Bytes, two-plane nodes 16 Bytes, and six-plane nodes 32 Bytes. Since the maximum number of nodes is known in advance, we could preallocate the maximum possible size required by the tree $N \cdot 16 \cdot (1+1+2)$ for $N$ objects. We have $N$ leaves $(1 \times 16)$, $N-1$ interior SKD-nodes $(1 \times 16)$, and at most $N$ six-plane bounding nodes $(2 \times 16)$. To minimize wasting of unused allocated memory we allocate memory subsequently in chunks of fixed size such as 16 KBytes. We use link nodes from current address to the beginning of a new chunk when the memory in the current chunk is exhausted.

## 6.3 H-trees Construction

The construction of an H-tree is started by a precomputation of a tight bounding box over all objects ($B^E(root)$). We set the traversal box to $B^T(root)$ and initialize it with infinite size. Then we start building up a tree with termination at leaves containing single objects. The type of the interior node to be created is chosen by evaluating the cost model described below. This model determines if it is the most advantageous to insert a six-plane bounding node (automatically for root node), a two-plane bounding node, or most frequently a SKD-tree node (also automatically for the child of the root node). Since the expected traversal cost of a node is estimated in a similar way as for kd-trees, we omit the notation of a node $\nu$ to keep formulas simpler.

- For the SKD-tree node the cost induced by a two plane separation is:

$$\begin{aligned} C_{SKD} &= C^T_{SKD} + C_{IT}/SA(B^T) \cdot \\ &\quad \big(N_L(b) \cdot SA(B^T_L(b)) + N_R(b) \cdot SA(B^T_R(b+1))\big), \end{aligned} \tag{6.1}$$

  where $C^T_{SKD}$ is the estimated cost to traverse the node and $B^T_L(b)$ and $B^T_R(b+1)$ is a traversal box of the left or right child respectively induced by objects in buckets $\langle 0, b\rangle$ and bucket $\langle b+1, M-1\rangle$. Further, $C_{IT}$ is an estimated cost for ray object intersection and $SA(B^T)$ denotes surface area of $B^T$. The difference to Eq. 5.1 is the different surface area of tight boxes (Eq. 5.1) and traversal boxes (Eq. 6.1). Further, the estimated cost of traversing the node $C^T_{SKD}$ is also plugged into the formula.

- For two-plane bounding node (BV2) we compute the cost as follows:

$$C_{2PN} = C^T_{2PN} + C_{IT} \cdot SA(B^T_{2PN})/SA(B^T) \cdot N, \tag{6.2}$$

  where $C^T_{2PN}$ is the estimated cost of traversing the node with two planes and $B^T_{2PN}$ is a new traversal box after bounding with the two planes.

- For six-planes bounding node (BV6) we compute the cost:

$$C_{6PN} = C^T_{6PN} + C_{IT} \cdot SA(B^T_{6PN})/SA(B^T) \cdot N, \tag{6.3}$$

  where $C^T_{6PN}$ is the estimated cost of traversing the node with six planes and $B^T_{6PN}$ is a new traversal box (= tight box) after bounding with the six planes.

The traversal costs $C^T_{SKD}$, $C^T_{2PN}$, and $C^T_{6PN}$ and an average intersection cost $C_{IT}$ are obtained from measurements before construction, creating random nodes and traversing them by a set of random rays as proposed in [4].

# 7 AH-Trees in expected $O(N \log \log N)$ via Approximation

In this chapter we describe the basic version of the construction algorithm to decrease the complexity from $O(N \log N)$ to expected $(N \log \log N)$. Our technique is based on discretization similar to sorting by radix sort that instead of $O(N \log N)$ allows sorting in $O(N)$ time, assuming limited precision of the input data representation. We call the resulting data structures *AH-trees*.

Below we briefly discuss the motivation for AH-trees. The discretization of the cost function computation along one axis described in Chapter 5 does not help us to reduce the algorithmic complexity of preprocessing, even if the reduction of constants behind big O notation is important in practice. We explain the reasons behind $O(N \log N)$ complexity required by sorting necessary in the construction of spatial hierarchies. First, a key concept of a spatial hierarchy is sorting and it takes $\Omega(N \log N)$ [15]. Assuming limited precision of the data representation radix sort allows time complexity in $O(N)$ time. This could be used in the preprocessing phase to sort all object boundaries for all three axes [43, 11]. The rest of the algorithm splits three lists containing object boundaries into six lists for one subdivision step. This avoids repetitive sorting for each interior node, however, we still need for splitting $O(N)$ time for every interior node with $N$ objects. Then the time complexity remains $O(N \log N)$ for a tree of depth $O(\log N)$, even if presorting took $O(N)$ time.

Below we show how it is possible to overcome $O(N \log N)$ by discretization, first for limited size of objects and second for arbitrary size of objects.

## 7.1 AH-trees for Limited Size of Objects

To overcome the algorithmic complexity issue we address the problem similarly to Reif and Tate [26, 39] using limited precision also for position of splitting planes *by discretization of the whole 3D domain*. However, instead of constructing a tree from bottom to top as they do, we use construction in a top-down fashion. It allows us to decide on a splitting plane at the discrete positions using the cost model based on SAH. Another issue to be solved is handling of large objects, since the method of Reif and Tate is designed only for point data. We propose a solution in the next section.

First, we compute a tight box over the scene objects. Then we create a 3D grid of arbitrary resolution preferably creating cubic-like cells. All the cells of the 3D grid have the same size. If the size of the cell is $C$ $(c_x, c_y, c_z)$, we then define a "small object" as one whose bounding box $B$ of size $(b_x, b_y, b_z)$ completely fits by size in $C$ (i.e. $b_x \leq c_x \wedge b_y \leq c_y \wedge b_z \leq c_z$). This allows us to limit a spatial extent of the object as shown in Figure 7.1.

In each cell of the grid we store the following data: list of objects assigned to the cell, the 'object count' assigned to the list, and 'aggregate count'. We distribute all objects into grid cells based on the position of the centroids of objects. Then we construct a 3D summed area table over 'object count' storing the precomputed prefix sum to 'aggregate count' of cells. The precomputation for $N$ objects and $M \times M \times M$ buckets is computed in $O(N + M^3) = O(N)$ time, since we take $M = const \cdot \sqrt[3]{N}$.

The use of 3D summed area table makes a difference to the method of Reif and Tate [26], since they in fact use preallocated regular octree. The 3D summed area tables allows us to evaluate a number of objects on the left and on the right for discrete positions of the splitting plane in constant time. However, two spatial extents associated with the children can overlap by the size of one bucket along the split axis. As we bound the size of the object to the size of the cell, the maximum size of the box tightly enclosing all the objects inside the cell is also limited. The cell box is extended for axis $a$ on both sides by $c_a/2$ for all three axes as depicted in Figure 7.1.

The discretization allows us also to implement the evaluation of the cost function in a discrete setting. We know the number of objects on the left and on the right for a selected bucket. We can compute the minimum of a cost function in expected $O(\log \log M)$ time by *interpolation-binary search* [31]. Briefly, interpolation-binary search consists of one phase of interpolation search [15] and one phase of binary search. We evaluate the cost model until the bucket with minimum cost is found. Another feature discretized is tight axis aligned bounding boxes for the left and right children. Although we cannot compute exact tight boxes for objects in constant time, we can

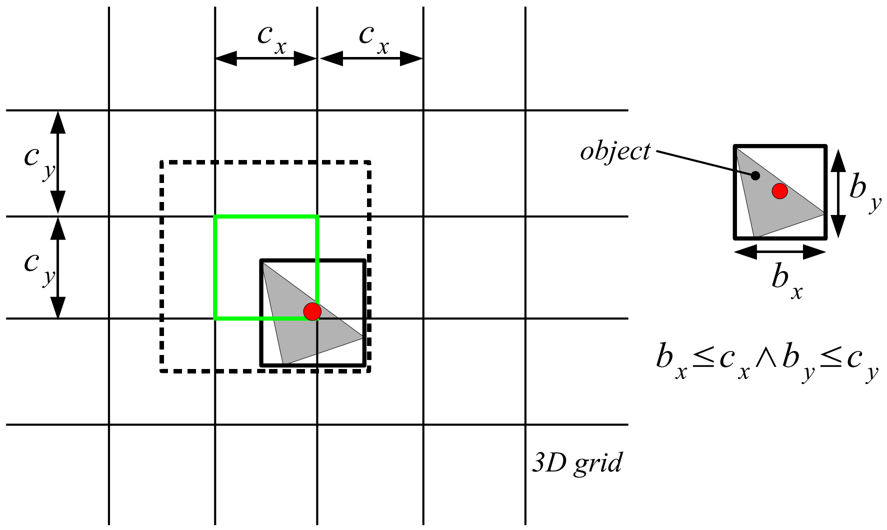



Figure 7.1: The concept of maximum bounding boxes for objects in a cell. Since the size of the objects is limited and objects are distributed according to their centroids, the traversal box is extended by half the size of the grid cell size. The box is depicted by dashed line.

easily detect if the box can be shrunk by the extent in any of the six bounding planes. This is important for scenes with skewed distributions that contain much empty space.

The construction algorithm proceeds recursively until a single cell is accessed. If this is a cell with only a single object, we create a leaf. Otherwise, based on the number of objects in a single cell we either recurse the algorithm with the discretization or use the algorithm described in Chapter 6. In practice for low number of objects ($\approx 100$) we use the algorithm in Chapter 6.

Below we discuss the time complexity of the AH-tree construction algorithm. In the analysis we relate $M$ to $N$ by formula $M = \sqrt[3]{N}$, resulting in $M^3$ cells. We also assume that the precision of the objects is limited by $c \log N$. Each level of the discretization to a 3D grid over $N$ objects resolves the tree construction in $3 \cdot \log \sqrt[3]{N}$ bits in all coordinates of object positions (hence object centroids). Then we need only a constant number of grid levels (independent of $N$) to address the limited resolution $c \log N$, including worst case distributions. Further, we have assumed in this section that the objects fulfill the condition of "small objects" above. Under these assumptions we can compute the time complexity as follows. We create a tree with exactly $N$ leaves. The number of interior nodes is also $O(N)$ (if bounding nodes are inserted, then we have at most $2 \cdot N - 2$ interior nodes).

To determine splitting positions for SKD-nodes closest to the root, we need $O(\log\log\sqrt[3]{N}) = O(\log\log N)$ steps. In total we carry out $N$ searches of splitting plane positions. By summing $N$ terms with the $\log\log N$ term we get $O(N\log\log N)$ complexity (some constants are left in big O notation). Recall that the factor $\log\log N$ is in practice limited to 6 for $N \leq 2^{64}$. If we abuse the big O notation as discussed in [26], we could say that the construction under the condition of small objects for every level of the grid hierarchy is $O(N)$.

## 7.2 AH-Tree for All Object Sizes

In this section we describe an extension of the basic algorithm described above to handle arbitrary objects. We utilize a method introduced by Guenther and Gaede [9] developed in the context of spatial databases. They propose a concept called *oversize shelves* of extra storage space for large spatial objects attached to interior nodes of a tree to avoid an excessive fragmentation in spatial subdivision schemes. The oversize objects are put to extra (ternary) child nodes during the insertion to the tree. Guenther and Gaede propose this method for a cell tree, but the idea of oversize shelves can be used in principle for any hierarchical DS.

Our DS are not created incrementally and the shape of the tree is not known in advance. We extend the construction algorithm of our AH-tree with oversize objects by two phases: *location phase* and *oversize shelves construction phase*. We insert all the objects based on the centroids to corresponding cells. In a cell of a grid we mark oversize objects. When we access a single cell $C$ to process a node $\nu$ (we create a leaf, or recurse, or create H-tree without discretization), we process all the oversize objects associated with $C$ in two phases as described below.

*Location Phase:* For every oversize object $O_c$ we traverse up the tree to the root starting at the node $\nu$. We insert $O_c$ to a temporary list $L(\nu_{insert})$ associated with the first SKD-tree node $\nu_{insert}$ we found on the traversal path from $\nu$ to the root where the bounding box of $O_c$ is fully contained in the traversal box $B^T(\nu_{insert})$.

*Oversize Shelves Construction:* During traversing up the tree from SKD-tree node $\nu$, after both child subtrees are constructed, we check every node whether the list $L(\nu)$ of oversize objects is empty. If the list contains some objects, we create a new tree $\mathcal{T}$ over all objects in $L(\nu)$ in the traversal box $B^T(\nu)$. We link $\mathcal{T}$ as a ternary child of $\nu$. In practice, we have to extend the nodes to allow ternary child nodes[1].

[1] This still fits to 16 Bytes for a SKD-node if implemented efficiently.

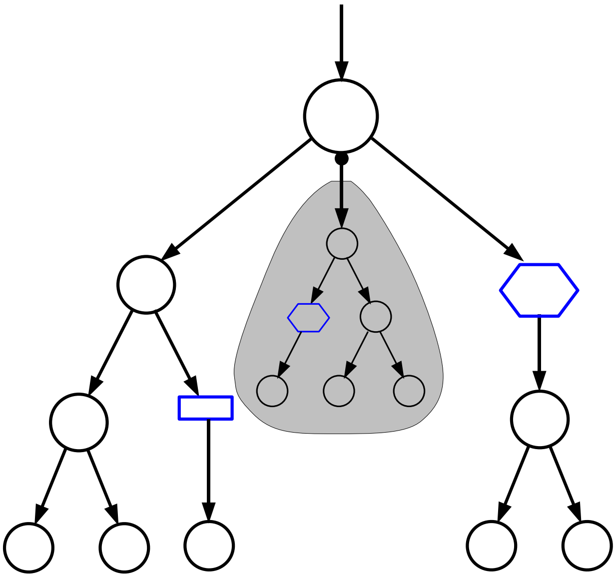


Figure 7.2: Oversize shelves concept: a part of H-tree with a single ternary child for oversize objects.

A part of the created AH-tree with ternary nodes containing oversize objects is shown in Figure 7.2. We show that the oversize shelves concept leads to correct DS. Each object $O_i$ found in a cell $C$ is used in one of three cases:

- to create a leaf with one object, if a cell contains only a single object,
- in a recursion that constructs a AH-tree rooted at $\nu$ for all objects found in the cell,
- in the list $L(\nu_{insert})$ of oversize objects of the node $\nu_{insert}$ on the path from $\nu$ to the root node.

If a list $L(\nu)$ containing oversize objects associated with a node $\nu$ is not empty, we construct a tree. We use recursively either the same algorithm described in this section ( 7) (if the number of objects is high and the discretization pays off) or an algorithm described in Chapter 5 (if the number of oversize objects is small, which is typical case). The construction algorithm for oversize objects, is also fully recursive, since the tree $\mathcal{T}_2$ constructed over oversize objects in tree $\mathcal{T}_1$ can also invoke a construction of tree $\mathcal{T}_3$ for oversize objects in any node of $\mathcal{T}_2$.

The discretization method for oversize objects has to be implemented carefully. It may happen that the boxes of the objects only overlap and the discretization does not separate objects, since the centroids of the objects are located in a single cell. When we detect such a case, we resort to $O(N \log N)$ algorithm described in Chapter 5. This however happens rarely for densely occupied scene regions, where all the objects span across the whole bounding box of the scene. As a result the complexity of this algorithm stays $O(N \log N)$ for the worst case, but in practice it shows expected $O(N \log \log N)$ for realistic input geometric data that overlap in acceptable way we refer to [6].

# 8 Traversal Algorithm for H-tree and AH-tree

The design of SKD-tree and H-tree nodes requires changes to the ray traversal algorithm. We use an interval clipping algorithm along a ray implemented with a stack similarly to kd-trees. A SKD-tree node traversal forms two (disjoint or overlapping) intervals along the ray from which the closest child is traversed first. The bounding nodes only clip the current interval along a ray by two or six planes. The SKD-tree nodes with ternary children first traverse the subtree constructed over oversize objects, since there is a higher probability of ray object intersections.

# 9 Results

We have implemented several ray tracing algorithms. We are concerned with the construction times of the DS and the performance of the ray tracing. As a reference we use to standard kd-trees constructed with SAH and automatic termination criteria implemented following course notes [35] and [11]. For these kd-trees we do not use split clipping [11, page 71] in order to minimize its construction time. For the same reason we evaluate the cost model only for a single axis. It is important to recall that our reference kd-tree construction algorithm is highly optimized. If we compare its performance to the recent publication [43] using the same scenes that our algorithm is typically twice faster than the one in [43]. This is also since we do not use split clipping. The performance of ray tracing with kd-trees is comparable to [43, 35] for shooting individual rays. For testing we used a PC equipped with AMD Athlon(tm) 64 X2 Dual Core Processor 3800+, but for both construction and rendering we exploit only a single core and single thread.

Table 9.1 shows the achieved experimental results. We use six static scenes shown in Figure 1.1 and three benchmark scenes for animated ray tracing (BART [18]). Three scenes have highly skewed distribution of objects that concerns positioning and sizes of objects, namely scene “Conference Room”, “Robots”, and “Museum”.

The first row of results refers to kd-tree. The second row denoted ’UG’ refers to uniform grids with the number of voxels five times higher than the number of objects. The third row denoted ’BVH-Med’ is the bounding volume hierarchy constructed with the spatial median in top-down fashion similarly to [33]. However our implementation is highly optimized (by factor from 3 to 5 compared with [33]). The fourth row denoted by ’BVH-SAH’ is bounding volume hierarchy extended by discrete evaluation of SAH in 1D as described in Chapter 5. The fifth row denoted ’H-tree’ follows Chapter 6 and the last row denoted ’AH-tree’ follows Chapter 7.2.

From the speedups for the construction time we conclude that construction of H-trees is by a factor of 4.3 (from 2.4 to 11.7) faster than for kd-trees, the more complex scene the better speedup. If we consider the total time needed to get the image (construction + rendering), we achieve average speedup about 4.0 (from 1.12 to 6.23). The performance of ray tracing with H-trees is comparable to kd-trees, H-trees are on average by 3% faster (speedup from 0.62 to 1.30).

The approximation of sorting on higher level for AH-trees leads to even faster construction times in the detriment to the performance of ray tracing for highly skewed distribution of objects. While for relatively uniform distribution of objects in the scene the performance of ray tracing is comparable to H-trees and kd-trees, for highly skewed distributions of objects (scene conference room, robots, kitchen) the performance drops by factor 0.23. The performance of ray tracing and the construction time make clear trade-off; we cannot make faster weighted sorting required in the construction of spatial hierarchy than $O(N \log N)$ while keeping the same performance of traversal algorithm through this hierarchy.

The construction time for H-trees has low variance over all frames of the animation for BART scenes. Our method then allows us to render the images by ray casting (only primary rays) on current PC hardware with dual core processors approximately at 3 to 5 frames per second, when one core is used to construct data structures and the other one ray traces an image.

We have been positively surprised by the results of Kay and Kajiya's algorithm redesigned in [33]. There are two reasons for its good performance. First, we have highly optimized the algorithm and its implementation of both the construction and traversal code given in [33], by about 300 to 500%. Second, some scenes shown in Table 9.1 have relatively uniform distribution of object primitives. In this case SAH-based construction is equivalent to spatial median based construction, but it is about twice faster to construct a hierarchy with a spatial median than that with the cost model based on SAH. This is justified by the number of operations we have to carry out that tightly corresponds to the source code.

| Method | $N_{NS}$ [$\times 10^3$] | $N_E$ [$\times 10^3$] | $N_{ER}$ [$\times 10^3$] | $N_{IT}$ | $N_{TS}$ | $N_{ETS}$ | $T_C[s]$ | $T_R[s]$ | $T_T[s]$ | $s(T_C)$ | $s(T_R)$ | $s(T_T)$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Static Scenes | | | | | | | | | | | | |
| | Scene - Conference Room - 17,936 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 80.2 | 29.3 | 70.9 | 11.99 | 49.74 | 10.32 | 0.12 | 0.81 | 0.93 | 1.00 | 1.00 | 1.00 |
| UG | 179.3 | 89.6 | 75.2 | 25.88 | 63.28 | 63.28 | 0.07 | 0.95 | 1.03 | 1.71 | 0.85 | 0.90 |
| BVH-Med | 35.9 | 17.9 | 17.9 | 10.40 | 89.84 | 10.40 | 0.02 | 1.15 | 1.17 | 6.00 | 0.70 | 0.79 |
| BVH-SAH | 35.9 | 17.9 | 17.9 | 5.27 | 60.86 | 5.27 | 0.05 | 0.93 | 0.97 | 2.40 | 0.87 | 0.96 |
| H-tree | 42.1 | 17.9 | 17.9 | 5.50 | 54.41 | 5.50 | 0.05 | 0.78 | 0.83 | 2.40 | 1.03 | 1.12 |
| AH-tree | 41.7 | 18.0 | 18.0 | 34.51 | 141.60 | 34.51 | 0.03 | 1.62 | 1.65 | 4.00 | 0.50 | 0.56 |
| | Scene - Bunny - 69,451 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 663.8 | 241.6 | 408.4 | 5.58 | 34.21 | 6.30 | 0.76 | 0.35 | 1.11 | 1.00 | 1.00 | 1.00 |
| UG | 663.8 | 345.3 | 245.1 | 16.42 | 32.65 | 32.65 | 0.26 | 0.38 | 0.64 | 2.92 | 0.92 | 1.73 |
| BVH-Med | 138.9 | 69.4 | 69.4 | 2.62 | 35.58 | 2.62 | 0.08 | 0.40 | 0.48 | 9.50 | 0.88 | 2.31 |
| BVH-SAH | 138.9 | 69.4 | 69.4 | 2.36 | 33.36 | 2.36 | 0.20 | 0.38 | 0.57 | 3.80 | 0.92 | 1.95 |
| H-tree | 153.3 | 69.4 | 69.4 | 2.80 | 26.65 | 2.80 | 0.20 | 0.30 | 0.49 | 3.80 | 1.17 | 2.27 |
| AH-tree | 153.3 | 69.4 | 69.4 | 3.57 | 37.85 | 3.57 | 0.11 | 0.39 | 0.50 | 6.91 | 0.89 | 2.22 |
| | Scene - Armadillo - 345,944 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 2891.1 | 1076.7 | 837.4 | 1.94 | 29.04 | 5.30 | 3.73 | 0.30 | 4.03 | 1.00 | 1.00 | 1.00 |
| UG | 3447.8 | 1723.9 | 833.5 | 11.78 | 58.29 | 58.29 | 1.51 | 0.47 | 1.98 | 2.47 | 0.64 | 2.04 |
| BVH-Med | 691.9 | 345.9 | 345.9 | 1.25 | 24.35 | 1.25 | 0.44 | 0.30 | 0.75 | 8.48 | 1.00 | 5.37 |
| BVH-SAH | 691.9 | 345.9 | 345.9 | 1.13 | 23.60 | 1.13 | 1.07 | 0.29 | 1.36 | 3.49 | 1.03 | 2.96 |
| H-tree | 783.2 | 345.9 | 345.9 | 1.19 | 17.39 | 1.19 | 1.07 | 0.23 | 1.30 | 3.49 | 1.30 | 3.10 |
| AH-tree | 777.8 | 345.9 | 345.9 | 1.29 | 29.02 | 1.29 | 0.54 | 0.33 | 0.86 | 6.91 | 0.91 | 4.69 |
| | Scene - Dragon - 844,037 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 4426.7 | 2213.3 | 3827.2 | 11.20 | 50.93 | 9.62 | 9.65 | 0.63 | 10.28 | 1.00 | 1.00 | 1.00 |
| UG | 4260.3 | 4260.3 | 3834.4 | 33.20 | 94.79 | 94.79 | 3.50 | 0.94 | 4.44 | 2.76 | 0.67 | 2.32 |
| BVH-Med | 1688.0 | 844.0 | 844.0 | 5.90 | 55.36 | 5.90 | 1.10 | 0.71 | 1.81 | 8.77 | 0.89 | 5.68 |
| BVH-SAH | 1688.0 | 844.0 | 844.0 | 5.49 | 51.61 | 5.49 | 2.62 | 0.68 | 3.29 | 3.68 | 0.93 | 3.12 |
| H-tree | 1921.3 | 844.0 | 844.0 | 6.43 | 44.75 | 6.43 | 2.60 | 0.55 | 3.14 | 3.71 | 1.14 | 3.27 |
| AH-tree | 1895.7 | 844.0 | 844.0 | 8.26 | 64.84 | 8.26 | 1.25 | 0.75 | 2.00 | 7.72 | 0.84 | 5.14 |
| | Scene - Happy Buddha - 1,051,739 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 2675.6 | 1542.8 | 8114.4 | 28.18 | 25.04 | 4.50 | 14.73 | 0.72 | 15.45 | 1.00 | 1.00 | 1.00 |
| UG | 5306.6 | 5306.6 | 10613.1 | 38.12 | 38.60 | 38.60 | 5.79 | 0.79 | 6.58 | 2.54 | 0.91 | 2.35 |
| BVH-Med | 2103.5 | 1051.7 | 1051.7 | 8.38 | 44.54 | 8.38 | 1.37 | 0.68 | 2.05 | 10.75 | 1.06 | 7.54 |
| BVH-SAH | 2103.5 | 1051.7 | 1051.7 | 8.13 | 43.20 | 8.13 | 3.13 | 0.67 | 3.80 | 4.71 | 1.07 | 4.07 |
| H-tree | 2335.4 | 1051.7 | 1051.7 | 9.19 | 39.53 | 9.19 | 3.11 | 0.54 | 3.65 | 4.74 | 1.33 | 4.23 |
| AH-tree | 2314.6 | 1051.7 | 1051.7 | 11.56 | 56.51 | 11.56 | 1.50 | 0.72 | 2.22 | 9.82 | 1.00 | 6.96 |
| | Scene - Blade - 1,765,388 triangles, 500×500 pixels | | | | | | | | | | | |
| kd-tree | 14984.3 | 7492.2 | 3388.4 | 1.98 | 33.28 | 5.72 | 16.45 | 0.34 | 16.80 | 1.00 | 1.00 | 1.00 |
| UG | 8862.8 | 8862.8 | 5369.6 | 13.93 | 76.42 | 76.42 | 6.85 | 0.64 | 7.50 | 2.40 | 0.53 | 2.24 |
| BVH-Med | 3530.8 | 1765.4 | 1765.4 | 1.89 | 33.28 | 1.89 | 2.31 | 0.43 | 2.75 | 7.12 | 0.79 | 6.11 |
| BVH-SAH | 3530.8 | 1765.4 | 1765.4 | 1.76 | 30.76 | 1.76 | 5.78 | 0.41 | 6.19 | 2.85 | 0.83 | 2.71 |
| H-tree | 4119.5 | 1765.4 | 1765.4 | 1.87 | 21.89 | 1.87 | 5.71 | 0.28 | 5.99 | 2.88 | 1.21 | 2.80 |
| AH-tree | 4064.7 | 1765.4 | 1765.4 | 2.15 | 42.24 | 2.15 | 2.51 | 0.45 | 2.96 | 6.55 | 0.76 | 5.68 |
| Dynamic Scenes | | | | | | | | | | | | |
| | Scene - Robots - 71,580 primitives, 400×300 pixels | | | | | | | | | | | |
| kd-tree | 506.3 | 186.5 | 356.3 | 7.55 | 58.38 | 9.43 | 0.75 | 0.24 | 0.99 | 1.00 | 1.00 | 1.00 |
| UG | 358.9 | 358.9 | 229.4 | 1074.9 | 13.99 | 13.99 | 0.73 | 5.30 | 6.03 | 1.03 | 0.05 | 0.16 |
| BVH-Med | 143.2 | 71.6 | 71.6 | 9.56 | 115.97 | 9.56 | 0.11 | 0.54 | 0.65 | 6.82 | 0.44 | 1.52 |
| BVH-SAH | 143.2 | 71.6 | 71.6 | 5.15 | 81.68 | 5.15 | 0.23 | 0.41 | 0.64 | 3.26 | 0.59 | 1.55 |
| H-tree | 166.9 | 71.6 | 71.6 | 9.22 | 86.36 | 9.22 | 0.22 | 0.36 | 0.58 | 3.41 | 0.67 | 1.71 |
| AH-tree | 165.2 | 71.6 | 71.6 | 43.48 | 262.17 | 43.48 | 0.13 | 1.06 | 1.19 | 5.77 | 0.23 | 0.83 |
| | Scene - Museum - 75,687 primitives, 400×300 pixels | | | | | | | | | | | |
| kd-tree | 313.6 | 156.8 | 1782.3 | 10.17 | 30.68 | 4.48 | 2.57 | 0.16 | 2.74 | 1.00 | 1.00 | 1.00 |
| UG | 378.0 | 378.0 | 271.6 | 14.69 | 43.56 | 43.56 | 0.79 | 0.20 | 0.99 | 3.25 | 0.80 | 2.77 |
| BVH-Med | 151.4 | 75.7 | 75.7 | 8.94 | 49.96 | 8.94 | 0.10 | 0.25 | 0.35 | 25.70 | 0.64 | 7.83 |
| BVH-SAH | 151.4 | 75.7 | 75.7 | 9.81 | 57.84 | 9.81 | 0.22 | 0.28 | 0.50 | 11.68 | 0.58 | 5.48 |
| H-tree | 163.4 | 75.7 | 75.7 | 9.88 | 55.44 | 9.88 | 0.22 | 0.21 | 0.44 | 11.68 | 0.76 | 6.23 |
| AH-tree | 163.9 | 75.7 | 75.7 | 14.71 | 60.88 | 14.71 | 0.13 | 0.27 | 0.40 | 19.77 | 0.59 | 6.85 |
| | Scene - Kitchen - 110,540 primitives, 400×300 pixels | | | | | | | | | | | |
| kd-tree | 335.6 | 167.8 | 353.7 | 6.29 | 56.59 | 8.60 | 1.03 | 0.22 | 1.25 | 1.00 | 1.00 | 1.00 |
| UG | 555.1 | 555.1 | 233.7 | 278.16 | 38.57 | 38.57 | 0.79 | 1.36 | 2.15 | 1.30 | 0.16 | 0.58 |
| BVH-Med | 221.1 | 110.5 | 110.5 | 7.22 | 106.91 | 7.22 | 0.15 | 0.50 | 0.64 | 6.87 | 0.44 | 1.95 |
| BVH-SAH | 221.1 | 110.5 | 110.5 | 5.21 | 87.50 | 5.21 | 0.35 | 0.42 | 0.76 | 2.94 | 0.52 | 1.64 |
| H-tree | 250.2 | 110.5 | 110.5 | 7.58 | 71.42 | 7.58 | 0.34 | 0.35 | 0.69 | 3.03 | 0.62 | 1.81 |
| AH-tree | 251.5 | 110.5 | 110.5 | 29.82 | 205.32 | 29.82 | 0.19 | 0.86 | 1.04 | 5.42 | 0.26 | 1.20 |

Table 9.1: Results for five static scenes and three dynamic scenes. For dynamic scenes we report average values per frame. $N_{NS}$ is the number of nodes in the data structure, $N_E$ is the number of non-hierarchical nodes in the data structure, $N_{ER}$ is the number of references to objects. Furthermore, $N_{IT}$ is the number of ray-object intersection per ray, $N_{TS}$ is the number of traversed nodes per ray, $N_{ETS}$ is the number of traversed non-hierarchical nodes per ray. Timings are given as: $T_C$ is the time for the data structure construction, $T_R$ is the rendering time for $500 \times 500$ pixels image for static scenes and $400 \times 300$ for dynamic scenes, $T_T = T_C + T_R$ is the total time needed to render the image including data construction. Speedups are: $s(T_C)$ is the speedup for construction time with respect to kd-trees, $s(T_R)$ is the speedup of the rendering, $s(T_T)$ is the speedup for construction + rendering time.

# 10 Conclusion and Future Work

In this paper we have focused on the fast construction of efficient spatial hierarchical data structures for ray tracing with the applications in rendering via ray tracing. We have addressed the problem in several ways by: decreasing constants behind the big O notation while keeping $O(N \log n)$ complexity, extending bounding volume hierarchies in a top down fashion by surface area heuristics, relaxing bounding volume primitives from boxes to two-plane slabs, and by designing a hybrid tree (H-tree) that efficiently combines the properties of spatial subdivisions and bounding volume hierarchies. Furthermore, we have proposed an approximate construction of AH-trees via discretization of spatial sorting that decreases the algorithmic complexity from $O(N \log N)$ to expected $O(N \log \log N)$ assuming objects of limited size and distributions. AH-trees handle objects of arbitrary sizes, in the worst case resulting in $O(N \log N)$ time complexity.

We tested our both novel data structures on six static scenes and three animated scenes. The ray tracing based on H-trees handles well uniform and also highly skewed distributions of objects in general scenes. The performance of ray tracing with H-trees is sometimes slightly faster (by up to $\approx$ 33%) sometimes slower (by up to $\approx$ 33%) than that one for kd-trees (as the state of the art technique), on average by 3% faster. However, the speedup for the construction of H-trees yields from 2.4 to 11.7, that allows application in interactive scenarios.

The AH-trees thanks to discretization can be constructed up to twice faster than H-trees. However, the decreased precision of splitting plane selections decreases the performance of ray tracing for highly skewed distributions of objects in the scene. On the other hand, AH-trees are perfectly suited for individual animated meshes, where the performance for ray tracing is comparable with kd-trees, but the construction is 4 to 20 times faster. In practice we can construct the efficient data structures for meshes having 100,000 to

200,000 objects in less than 300 milliseconds (hardware of year 2005) on a single CPU core (it is $\approx 3$ frames per second).

The proposed data structures can be used in general visibility preprocessing relying on spatial hierarchies for example in occlusion culling where queries are organized along lines as in ray tracing. We plan to study more deeply the relation between precision of splitting plane determination and the performance of resulting data structures, since making this unusual trade-off is also an algorithmic contribution of this paper. It is almost obvious that an efficient SSE implementation of a ray-packet traversal for kd-trees [44, 35] can be implemented for H-trees and AH-trees. Furthermore, we assume the hierarchical version of ray casting [28] for primary and shadow rays can also be implemented on the proposed data structures.

# Bibliography


[1] B. Adams, R. Keiser, M. Pauly, L. Guibas, M. Gross, and P. Dutre. Efficient Raytracing of Deformable Point-Sampled Surfaces. In *Proceedings of the 2005 Eurographics Conference*, pages 677–684.

[2] J. Arvo and D. Kirk. *A Survey of Ray Tracing Acceleration Techniques*, pages 201–262. Academic Press, 1989.

[3] J. L. Bentley. Multidimensional Binary Search Trees Used for Associative Searching. *Communications of the ACM*, 18(9):509–517, 1975.

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

[5] Y. Chrysanthou and M. Slater. Computing Dynamic Changes to BSP Trees. In A. Kilgour and L. Kjelldahl, editors, *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings)*, volume 11, pages 321–332, September 1992.

[6] M. de Berg, M. J. Katz, A. F. van der Stappen, and J. Vleugels. Realistic input models for geometric algorithms. In *Symposium on Computational Geometry*, pages 294–303, 1997.

[7] V. Gaede and O. Günther. Multidimensional Access Methods. *ACM Computing Surveys*, 30(2):170–231, 1998.

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

[9] O. Günther and H. Noltemeier. Spatial database indices for large extended objects. In *ICDE*, pages 520–526. IEEE Computer Society, 1991.

[10] A. Guttman. R-trees: A Dynamic Index Structure for Spatial Searching. In *SIGMOD '84: Proceedings of the 1984 ACM SIGMOD international conference on Management of data*, pages 47–57, New York, NY, USA, 1984. ACM Press.

[11] V. Havran. *Heuristic Ray Shooting Algorithms*. PhD thesis, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[12] V. Havran, J. Prikryl, and W. Purgathofer. Statistical comparison of ray-shooting efficiency schemes. Technical Report TR-186-2-00-14, May 2000.

[13] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. *Proceedings of Graphics Interface '89*, pages 164–172, June 1989.

[14] T.L. Kay and J.T. Kajiya. Ray Tracing Complex Scenes. *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 20(4):269–278, 1986.

[15] Donald E. Knuth. *The Art of Computer Programming, Volume 3 Sorting and Searching*. Addison-Wesley, 1998.

[16] T. Larsson and T. Akenine-Möller. A Dynamic Bounding Volume Hierarchy for Generalized Collision Detection. In *Proceedings of the 2nd Workshop on Virtual Reality Interactions and Physical Simulations*, pages 91–100, Pisa, Italy, November 2005.

[17] J. Lext and T. Akenine-Möller. Towards Rapid Reconstruction for Animated Ray Tracing. In *Eurographics 2001 – Short Presentations*, pages 311–318, 2001.

[18] J. Lext, U. Assarsson, and T. Möller. BART: A Benchmark for Animated Ray Tracing. Technical report, Department of Computer Engineering, Chalmers University of Technology, Göteborg, Sweden, May 2000. Available at `http://www.ce.chalmers.se/BART/`.

[19] J. Lext, U. Assarsson, and T. Möller. A Benchmark for Animated Ray Tracing. *IEEE Comput. Graph. Appl.*, 21(2):22–31, 2001.

[20] J. D. MacDonald and K. S. Booth. Heuristics for Ray Tracing using Space Subdivision. In *Proceedings of Graphics Interface*, pages 152–63, 1989.

[21] J. Mahovsky. *Ray Tracing with Reduced-Precision Bounding Volume Hierarchies*. PhD thesis, University of Calgary, 2005.

[22] J. P. Molina Masso and P. Gonzalez Lopez. Automatic Hybrid Hierarchy Creation: a Cost-model Based Approach. *Computer Graphics Forum*, 22(1):5–13, 2003.

[23] B. Ooi, K.J. McDonell, and R. Sacks-Davis. Spatial KD-tree: An Indexing Mechanism for Spatial Databases. In *Proceedings of the IEEE COMPSAC Conference*, 1987.

[24] B. C. Ooi, R. Sacks-Davis, and J. Han. Indexing in Spatial Databases, 1993. Unpublished Manuscript, available at: `http://www.iscs.nus.edu.sg/~ooibc/`.

[25] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan. Interactive Ray Tracing. In *Proceedings of Interactive 3D Graphics*, pages 119–126, 1999.

[26] J.H. Reif and S.R. Tate. Fast Spatial Decomposition and Closest Pair Computation for Limited Precision Input. *Algorithmica: An International Journal in Computer Science*, 28(3):271–287, 2000.

[27] E. Reinhard, B. Smits, and C. Hansen. Dynamic Acceleration Structures for Interactive Ray Tracing. In *Proceedings of the Eurographics Workshop on Rendering*, pages 299–306, Brno, Czech Republic, June 2000.

[28] A. Reshetov, A. Soupikov, and J. Hurley. Multi-Level Ray Tracing Algorithm. *ACM Transaction of Graphics*, 24(3):1176–1185, 2005. (Proceedings of ACM SIGGRAPH).

[29] S. M. Rubin and T. Whitted. A Three-Dimensional Representation for Fast Rendering of Complex Scenes. *Computer Graphics*, 14(3):110–116, July 1980.

[30] H. Samet. *Foundations of Multidimensional and Metric Data Structures.* Morgan Kaufmann, 2006. to appear.

[31] N. Santoro and J. B. Sidney. Interpolation-Binary Search. *Inf. Proc. Lett.*, 20:179–181, 1985.

[32] J. Shagam and J. Pfeiffer. Dynamic Irregular Octrees. Technical Report NMSU-CS-2003-004, 2003.

[33] P. Shirley and R. K. Morley. *Realistic Ray Tracing.* A K Peters, second edition, 2003.

[34] G. Simiakakis. *Accelerating Ray Tracing with Directional Subdivision and Parallel Processing*. PhD thesis, University of East Anglia, 1995.

[35] P. Slusallek, P. Shirley, I. Wald, G. Stoll, and B. Mark, editors. *SIGGRAPH 2005 Course Notes 38 on Interactive Ray Tracing*, 2005.

[36] O. Sudarsky and C. Gotsman. Dynamic Scene Occlusion Culling. In Hans Hagen, editor, *IEEE Transactions on Visualization and Computer Graphics*, volume 5 (1), pages 13–29. IEEE Computer Society, 1999.

[37] L. Szécsi, B. Benedek, and L. Szirmay-Kalos. Accelerating Animation Through Verification of Shooting Walks. In *Proceedings of SCCG*, pages 231–238. ACM Press, 2003.

[38] L. Szirmay-Kalos, V. Havran, B. Balázs, and L. Szécsi. On the Efficiency of Ray-shooting Acceleration Schemes. In Alan Chalmers, editor, *Proceedings of SCCG*, pages 89–98. ACM Siggraph, 2002.

[39] S.R. Tate and K. Xu. General-Purpose Spatial Decomposition Algorithms: Experimental Results. In *Proceedings of ALENEX 00*, pages 179–216, 2000.

[40] M. Teschner, B. Heidelberger, D. Manocha, N. Govindaraju, G. Zachmann, S. Kimmerle, J. Mezger, and A. Fuhrmann. Collision Handling in Dynamic Simulation Environments. In *Eurographics Tutorials*, pages 79–185, 2005.

[41] I. Wald. Course on Interactive Ray Tracing, Handling Dynamic Scenes. In *ACM SIGGRAPH 2005 Course Notes 38*, pages 1–25. ACM Press, 2005.

[42] I. Wald, C. Benthin, and P. Slusallek. Distributed Interactive Ray Tracing of Dynamic Scenes. In *Proceedings of the IEEE Symposium on Parallel and Large-Data Visualization and Graphics (PVG)*, 2003.

[43] I. Wald and V. Havran. On building fast kd-trees for ray tracing, and on doing that in o(n log n). SCI Institute Technical Report UUSCI-2006-009, University of Utah, 2006.

[44] I. Wald, P. Slusallek, C. Benthin, and M. Wagner. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001. (Proceedings of Eurographics).

[45] K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Cho, C. M. Park, and I. Y. Song. Octree-R: An adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):343–349, 1995.

Below you find a list of the most recent technical reports of the Max-Planck-Institut für Informatik. They are available by anonymous ftp from `ftp.mpi-sb.mpg.de` under the directory `pub/papers/reports`. Most of the reports are also accessible via WWW using the URL `http://www.mpi-sb.mpg.de`. If you have any questions concerning ftp or WWW access, please contact `reports@mpi-sb.mpg.de`. Paper copies (which are not necessarily free of charge) can be ordered either by regular mail or by e-mail at the address below.

Max-Planck-Institut für Informatik
Library
attn. Anja Becker
Stuhlsatzenhausweg 85
66123 Saarbrücken
GERMANY
e-mail: `library@mpi-sb.mpg.de`

---

| | | |
|---|---|---|
| MPI-I-2006-5-004 | F. Suchanek, G. Ifrim, G. Weikum | Combining Linguistic and Statistical Analysis to Extract Relations from Web Documents |
| MPI-I-2006-5-003 | V. Scholz, M. Magnor | Garment Texture Editing in Monocular Video Sequences based on Color-Coded Printing Patterns |
| MPI-I-2006-5-002 | H. Bast, D. Majumdar, R. Schenkel, M. Theobald, G. Weikum | IO-Top-k: Index-access Optimized Top-k Query Processing |
| MPI-I-2006-5-001 | M. Bender, S. Michel, G. Weikum, P. Triantafilou | Overlap-Aware Global df Estimation in Distributed Information Retrieval Systems |
| MPI-I-2006-4-007 | O. Schall, A. Belyaev, H. Seidel | Feature-preserving Non-local Denoising of Static and Time-varying Range Data |
| MPI-I-2006-4-004 | V. Havran, R. Herzog, H. Seidel | On Fast Construction of Spatial Hierarchies for Ray Tracing |
| MPI-I-2006-4-003 | E. de Aguiar, R. Zayer, C. Theobalt, M. Magnor, H. Seidel | A Framework for Natural Animation of Digitized Models |
| MPI-I-2006-4-002 | G. Ziegler, A. Tevs, C. Theobalt, H. Seidel | GPU Point List Generation through Histogram Pyramids |
| MPI-I-2006-4-001 | R. Mantiuk | ? |
| MPI-I-2006-2-001 | T. Wies, V. Kuncak, K. Zee, A. Podelski, M. Rinard | On Verifying Complex Properties using Symbolic Shape Analysis |
| MPI-I-2006-1-007 | I. Weber | ? |
| MPI-I-2006-1-006 | M. Kerber | Division-Free Computation of Subresultants Using Bezout Matrices |
| MPI-I-2006-1-005 | I. Albrecht | ? |
| MPI-I-2006-1-004 | E. de Aguiar | ? |
| MPI-I-2006-1-001 | M. Dimitrios | ? |
| MPI-I-2005-5-002 | S. Siersdorfer, G. Weikum | Automated Retraining Methods for Document Classification and their Parameter Tuning |
| MPI-I-2005-4-006 | C. Fuchs, M. Goesele, T. Chen, H. Seidel | An Emperical Model for Heterogeneous Translucent Objects |
| MPI-I-2005-4-005 | G. Krawczyk, M. Goesele, H. Seidel | Photometric Calibration of High Dynamic Range Cameras |
| MPI-I-2005-4-004 | C. Theobalt, N. Ahmed, E. De Aguiar, G. Ziegler, H. Lensch, M.A.,. Magnor, H. Seidel | Joint Motion and Reflectance Capture for Creating Relightable 3D Videos |
| MPI-I-2005-4-003 | T. Langer, A.G. Belyaev, H. Seidel | Analysis and Design of Discrete Normals and Curvatures |
| MPI-I-2005-4-002 | O. Schall, A. Belyaev, H. Seidel | Sparse Meshing of Uncertain and Noisy Surface Scattered Data |
| MPI-I-2005-4-001 | M. Fuchs, V. Blanz, H. Lensch, H. Seidel | Reflectance from Images: A Model-Based Approach for Human Faces |
| MPI-I-2005-2-004 | Y. Kazakov | A Framework of Refutational Theorem Proving for Saturation-Based Decision Procedures |

| | | |
|---|---|---|
| MPI-I-2005-2-003 | H.d. Nivelle | Using Resolution as a Decision Procedure |
| MPI-I-2005-2-002 | P. Maier, W. Charatonik, L. Georgieva | Bounded Model Checking of Pointer Programs |
| MPI-I-2005-2-001 | J. Hoffmann, C. Gomes, B. Selman | Bottleneck Behavior in CNF Formulas |
| MPI-I-2005-1-008 | C. Gotsman, K. Kaligosi, K. Mehlhorn, D. Michail, E. Pyrga | Cycle Bases of Graphs and Sampled Manifolds |
| MPI-I-2005-1-008 | D. Michail | ? |
| MPI-I-2005-1-007 | I. Katriel, M. Kutz | A Faster Algorithm for Computing a Longest Common Increasing Subsequence |
| MPI-I-2005-1-003 | S. Baswana, K. Telikepalli | Improved Algorithms for All-Pairs Approximate Shortest Paths in Weighted Graphs |
| MPI-I-2005-1-002 | I. Katriel, M. Kutz, M. Skutella | Reachability Substitutes for Planar Digraphs |
| MPI-I-2005-1-001 | D. Michail | Rank-Maximal through Maximum Weight Matchings |
| MPI-I-2004-NWG3-001 | M. Magnor | Axisymmetric Reconstruction and 3D Visualization of Bipolar Planetary Nebulae |
| MPI-I-2004-NWG1-001 | B. Blanchet | Automatic Proof of Strong Secrecy for Security Protocols |
| MPI-I-2004-5-001 | S. Siersdorfer, S. Sizov, G. Weikum | Goal-oriented Methods and Meta Methods for Document Classification and their Parameter Tuning |
| MPI-I-2004-4-006 | K. Dmitriev, V. Havran, H. Seidel | Faster Ray Tracing with SIMD Shaft Culling |
| MPI-I-2004-4-005 | I.P. Ivrissimtzis, W.-. Jeong, S. Lee, Y.a. Lee, H.-. Seidel | Neural Meshes: Surface Reconstruction with a Learning Algorithm |
| MPI-I-2004-4-004 | R. Zayer, C. Rssl, H. Seidel | r-Adaptive Parameterization of Surfaces |
| MPI-I-2004-4-003 | Y. Ohtake, A. Belyaev, H. Seidel | 3D Scattered Data Interpolation and Approximation with Multilevel Compactly Supported RBFs |
| MPI-I-2004-4-002 | Y. Ohtake, A. Belyaev, H. Seidel | Quadric-Based Mesh Reconstruction from Scattered Data |
| MPI-I-2004-4-001 | J. Haber, C. Schmitt, M. Koster, H. Seidel | Modeling Hair using a Wisp Hair Model |
| MPI-I-2004-2-007 | S. Wagner | Summaries for While Programs with Recursion |
| MPI-I-2004-2-002 | P. Maier | Intuitionistic LTL and a New Characterization of Safety and Liveness |
| MPI-I-2004-2-001 | H. de Nivelle, Y. Kazakov | Resolution Decision Procedures for the Guarded Fragment with Transitive Guards |
| MPI-I-2004-1-006 | L.S. Chandran, N. Sivadasan | On the Hadwiger's Conjecture for Graph Products |
| MPI-I-2004-1-005 | S. Schmitt, L. Fousse | A comparison of polynomial evaluation schemes |
| MPI-I-2004-1-004 | N. Sivadasan, P. Sanders, M. Skutella | Online Scheduling with Bounded Migration |
| MPI-I-2004-1-003 | I. Katriel | On Algorithms for Online Topological Ordering and Sorting |
| MPI-I-2004-1-002 | P. Sanders, S. Pettie | A Simpler Linear Time 2/3 - epsilon Approximation for Maximum Weight Matching |
| MPI-I-2004-1-001 | N. Beldiceanu, I. Katriel, S. Thiel | Filtering algorithms for the Same and UsedBy constraints |
| MPI-I-2003-NWG2-002 | F. Eisenbrand | Fast integer programming in fixed dimension |
| MPI-I-2003-NWG2-001 | L.S. Chandran, C.R. Subramanian | Girth and Treewidth |
| MPI-I-2003-4-009 | N. Zakaria | FaceSketch: An Interface for Sketching and Coloring Cartoon Faces |
| MPI-I-2003-4-008 | C. Roessl, I. Ivrissimtzis, H. Seidel | Tree-based triangle mesh connectivity encoding |
| MPI-I-2003-4-007 | I. Ivrissimtzis, W. Jeong, H. Seidel | Neural Meshes: Statistical Learning Methods in Surface Reconstruction |