Alcom-FT Technical Report Series
ALCOMFT-TR-03-131

Time-Critical Collision Detection Using an Average-Case Approach

Jan Klein
Heinz Nixdorf Institute and
Institute of Computer Science
University of Paderborn, Germany
janklein@uni-paderborn.de

Gabriel Zachmann
Dept. of Computer Graphics and Virtual Reality
University of Bonn, Germany
zach@cs.uni-bonn.de

ABSTRACT

We present a novel, generic framework and algorithm for hierarchical collision detection, which allows an application to balance speed and quality of the collision detection.

We pursue an average-case approach that yields a numerical measure of the quality. This can either be specified by the simulation or interaction, or it can help to assess the result of the collision detection in a time-critical system.

Conceptually, we consider sets of polygons during traversal and estimate probabilities that there is an intersection among these sets. This can be done efficiently by storing characteristics about the average distribution of the set of polygons with each node in a bounding volume hierarchy (BVH). Consequently, we neither need any polygon intersection tests nor access to any polygons during the collision detection process.

Our approach can be applied to virtually any BVH. Therefore, we call a BVH that has been augmented in this way an ***a**verage-**d**istri**b**ution tree* or *ADB-tree*.

We have implemented our new approach with two basic BVHs and present performance measurements and comparisons with a very fast previous algorithm, namely the DOP-tree. The results show a speedup of about a factor 3 to 6 with only approximately 4% error.

Categories and Subject Descriptors

I.3.5 [**Computer Graphics**]: Computational Geometry and Object-Modeling—*Geometric algorithms, languages and systems; object hierarchy; physically-based modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation; virtual reality*; G.2.1 [**Discrete Mathematics**]: Combinatorics—*Combinatorial algorithms*

Keywords

Interference Detection, Virtual Prototyping, Hierarchical Partitioning, Bounding Volume Trees, Hierarchical Data Structures, Probabilistic Analysis, Average-Case Algorithms

1. INTRODUCTION

Fast collision detection of polygonal objects is needed in many highly interactive applications such as virtual prototyping, 3D games, physically-based simulation, and robotics. Most of these applications simulate some kind of more or less realistic object behavior.

It has often been noted previously, that the *perceived quality* of a virtual environment and, in fact, most interactive 3D applications, crucially depends on the real-time response to collisions [27]. At the same time, humans cannot distinguish between physically correct and *physically plausible* behavior of objects (at least up to some degree) [8]. [1] Since collision detection is still the major bottleneck of many of these simulations and interactions, it is obvious that this is where we can achieve the best speedup.

Therefore, we introduce the novel framework of collision detection using an average-case approach, thus extending the set of techniques for plausible simulation. To our knowledge, this is the first time that the *quality* of collision detection can be decreased in a controlled way (while increasing the speed), such that a numeric *measure* of the quality of the results is obtained (which can then be related to the perceived quality). The methods developed in this paper can be applied to virtually any hierarchical collision detection algorithm.

Conceptually, the main idea of the new algorithm is to consider *sets of polygons* at inner nodes of the BV hierarchy, and then, during traversal, check pairs of sets of polygons. However, we neither check pairs of polygons derived from such a pair of polygon sets, nor store any polygons with the nodes. Instead, based on a small number of parameters describing the *distribution* within the polygon sets, we will derive an estimation of the probability that there *exists* a pair of intersecting polygons. This has two advantages:

1. The application can control the runtime of the algorithm by specifying the desired "quality" of the collision detection (to be defined later).

2. The probabilities can guide the algorithm to those parts of the BV hierarchies that allow for faster convergence of the estimate.

[1] Analogously to rendering, a number of human factors determine whether or not the "incorrectness" of a simulation will be noticed, such as the mental load of the viewing person, cluttering of the scene, occlusions, velocity of the objects and the viewpoint, point of attention, etc.

The next section will review some of the work related to ours. Section 3 describes the framework and derivations in detail, while Section 4 provides various results and benchmarks of our new approach. Finally, Section 5 draws some conclusions and describes possible avenues for further work.

2. RELATED WORK

Bounding volume (BV) hierarchies have proven to be a very efficient data structure for rigid collision detection, and, to some extent, even for deformable objects.

One of the design choices with BV trees is the type of BV. In the past, a wealth of BV types has been explored, such as spheres [13, 25], OBBs [12], DOPs [18, 30], Boxtrees [2, 31], AABBs [19, 28], and convex hulls [10].

Alternatives to BV hierarchies are approaches that utilize the graphics hardware [5, 6, 21, 23, 26]. Most of these methods utilize the stencil buffer or the feedback mechanism of OpenGL. However, all of them compete with the rendering module for the graphics resources (unless one spends another board just for the collision detection).

Another alternative are space-subdivision approaches, for instance by an octree [15] or a voxel grid [22]. In general, non-hierarchical data structures seem to be more promising for collision detection of deformable objects [1, 11, 14], although some geometric data structures suggest a natural BV hierarchy [20]. Deformable collision detection is not the focus of our work presented here.

BV hierarchies lend themselves well to time-critical collision detection, i.e., the scheduler interrupts the traversal when the time budget is exhausted. This has been observed by several researchers [9, 13]. Hubbard presented the idea of interruptible collision detection using sphere trees [13]. Dingliana and O'Sullivan [9] are concerned with modelling contacts based on interrupted sphere tree traversals. The method described there can be applied in our framework too. However, they do not provide any theoretical foundations concerning the error incurred by an incomplete traversal. In addition, their methods do not support application-driven "levels-of-detail" of collision detection, where the application can specify an allowable error rate beforehand.

A different approach to reducing query times is to try to learn and model the query probability distribution either before the hierarchy construction [3] or at runtime [4] (i.e., the construction is done on-demand). While being quite effective, their data structures and traversal algorithms are still pretty much the classical ones (besides the fact that they apply the approach only to the problem of detecting an intersection between a line segment and the environment/object). However, our framework can be combined with theirs very well and easily.

Probabilistic methods have been applied to other problems of computer graphics, such as out-of-core walkthroughs of virtual environments [16] and the randomized z-buffer [29]. To our knowledge, however, there is neither literature about probabilistic collision detection nor about algorithms using a probabilistic analysis or an average-case approach to control the quality and speed of collision detection.

3. AN AVERAGE-CASE APPROACH

Given two BV hierarchies for two objects, virtually all collision detection approaches traverse the hierarchies simultaneously by an algorithm similar to that shown in Figure 1. It allows to quickly zoom into areas of close proximity. The algorithm (usually) stops if an intersection is found or if the traversal has visited all relevant sub-trees.

```
traverse(A, B)
  if A and B do not overlap then return;
  if A and B are leaves then
    return intersection of primitives enclosed
           by A and B;
  else
    for all children A[i] and B[j] do
      traverse(A[i], B[j]);
```

Figure 1: Traditional hierarchical collision detection algorithm.

As mentioned in the previous section, it is, of course, possible to just cut off this traversal any time the application or scheduler deems suitable. The problem with this approach is that it gives absolutely no hint as to the confidence in the result.

In contrast, our novel approach enables an application to trade accuracy for speed in a controlled fashion, so that it always has a "measure of confidence" into the result reported by the collision detection algorithm.

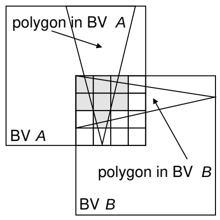

Figure 2: We partition the intersection volume by a grid. Then, we determine the probability that there are *collision cells* where polygons of different objects *can* intersect (highlighted in grey).

3.1 Overview of our Approach

The idea of our algorithm is to guide and to abort the traversal by the *probability* that a pair of BVs contains intersecting polygons. The design of our algorithm was influenced by the idea to develop an algorithm that works well and efficient for most practical cases — in other words, that works well in the average case. Therefore, we estimate the probability of a collision within a pair of BVs by some characteristics about the average distribution of the polygons, but we do not use the exact positions of the polygons during the collision detection.

Conceptually, the intersection volume of A and B, $A \cap B$, is partitioned into a regular grid (see Figure 2). If a cell contains *enough* polygons of one BV, we call it a *possible collision cell* and if a cell is a possible collision cell with respect to A and also with respect to B, we call it a *collision cell* (a more precise definition is given in Section 3.2). Given the total number of cells in $A \cap B$, the number of possible collision cells from A and B, resp., lying in $A \cap B$, we can compute the probability that there are at least x collision cells in $A \cap B$. This probability can be used to estimate the probability that the polygons from A and B intersect. For the computations, we assume that the probability of being a possible collision cell is evenly distributed among all cells of the partitioning because we are looking for an algorithm that works well in the average case where the polygons are uniformly distributed in the BVs. Of course, this assumption is more realistic for smaller BVs (compared to the object size) than for larger ones. Later in Section 3.4.3 we pay special attention to uneven polygon distributions.

An outline of our traversal algorithm is shown in Figure 3. Function computeProb estimates the probability of an intersection between the polygon sets of two BVs. By descending first into those sub-trees that have highest probability, we can quickly increase the confidence in the result and determine the end of the traversal. Basically, we are now dealing with priorities of pairs of nodes, which we maintain in a priority queue. It contains only pairs whose corresponding polygons can intersect. The queue is sorted by the probability of an intersection. Instead of a recursive traversal, our algorithm just extracts the front node pair of the queue and inserts a number of child pairs.

The quality and speed of the collision detection strongly depends on the accuracy of the probability computation. Several factors contribute to that, such as the kind of partitioning and the size of the polygons relative to the size of the cells. This is discussed more detailed in Section 4.2.2.

There are two other important parameters in our traversal algorithm, p_{min} and k_{min}, that affect the quality and the speed of the collision detection. Both can be specified by the application every time it performs a collision detection. A *pair of collision nodes* is found if the probability of an intersection between their associated polygons is larger than p_{min}. A collision is reported if at least k_{min} such pairs have been found. The smaller p_{min} or k_{min}, the shorter is the runtime and, in most cases, the more errors are made.

```
traverse(A, B)
    priorityQueue q;  k:=0;
    q.insert(A, B, 1);
    while q is not empty do
        A, B := q.pop;
        for all children A[i] and B[j] do
          p := computeProb(A[i], B[j]);
          if  p ≥ p_min
             k:=k+1;
             if k ≥ k_min return "collision";
          if p > 0 q.insert(A[i], B[j], p);
    return "no collision";
```

Figure 3: Our algorithm traverses two BV hierarchies by maintaining a priority queue of BV pairs sorted by the probability of an intersection.

The remainder of this section explains this framework more precisely in a top-down manner.

3.2 Terms and Definitions

For the sake of accuracy and conciseness, we introduce the following terms and definitions. We treat the terms *bounding volume* (BV) and *node* of a hierarchy synonymous. A and B will always denote BVs of two different hierarchies.

Definition 1 All polygons of the object contained in BV A or intersecting A are denoted as $P(A)$.

Let c be a cell of the partitioning of $A \cap B$. The total area of all polygons in $P(A)$ clipped against cell c is denoted as $\mathrm{Area}_c(A)$.

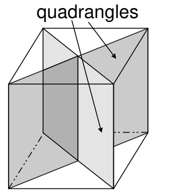

Figure 4: A cubic collision cell c with side length a. $\mathrm{Area}_c(A)$ and $\mathrm{Area}_c(B)$ must be at least $\mathrm{MaxArea}(c) = a^2\sqrt{2}$, which is exactly the area of the two quadrangles.

$\mathrm{MaxArea}(c)$ denotes the area of the largest polygon that can be contained completely in cell c.

Definition 2 (possible collision cell) Given a BV A and a cell c. c is a *possible collision cell*, if $\mathrm{Area}_c(A) \geq \mathrm{MaxArea}(c)$.

Definition 3 (collision cell) Given two intersecting BVs A and B as well as a partitioning of $A \cap B$. Then, A and B have a (common) *collision cell* iff $\exists c : \mathrm{Area}_c(A) \geq \mathrm{MaxArea}(c) \wedge \mathrm{Area}_c(B) \geq \mathrm{MaxArea}(c)$ (with suitably chosen $\mathrm{MaxArea}(c)$).

Definitions 2 and 3 are actually the first steps towards computing the probability of an intersection among the polygons of a pair of BVs. In particular, definition 3 is motivated by the following observation. Consider a cubic cell c with side length a, containing exactly one polygon from A and B, resp. Assuming $\mathrm{Area}_c(A) = \mathrm{Area}_c(B) = \mathrm{MaxArea}(c)$, then we must have exactly the configuration shown in Figure 4, i.e., an intersection, if we choose $\mathrm{MaxArea}(c) = a^2\sqrt{2}$. Obviously, a set of polygons is not planar (usually), so even if $\mathrm{Area}_c(A) > \mathrm{MaxArea}(c)$ there might still not be an intersection. But since almost all practical objects have bounded curvature in most vertices, the approximation by a planar polygon fits better and better as the polygon set covers smaller and smaller a surface of the object.

Definition 4 ($LB(c_{A\cap B})$) Given an arbitrary collision cell c from the partitioning of $A\cap B$. A lower bound for the probability that a collision occurs in c is denoted as $LB(c_{A\cap B})$.

Let us conclude this subsection by the following important definition.

Definition 5 ($Pr[c(A \cap B) \geq x\ ]$) The probability that at least x collision cells exist in $A \cap B$ is denoted as $Pr[c(A \cap B) \geq x]$.

Overall, given the probability $Pr[c(A \cap B) \geq 1]$, a lower bound for the probability that the polygons from A and B intersect is given by

$$Pr[P(A) \cap P(B) \neq \varnothing] \geq Pr[c(A \cap B) \geq 1] \cdot LB(c_{A\cap B}). \quad (1)$$

A better lower bound is given in Section 3.5.2, where $x > 1$ is used for $Pr[c(A \cap B) \geq x]$. Section 3.5.1 will derive $Pr[c(A \cap B) \geq x]$, while Section 3.5.3 will derive $LB(c_{A\cap B})$.

3.3 ADB-Trees

As mentioned before, our approach is applicable to virtually all BV hierarchies by augmenting them with a simple description of the distribution of the set of polygons. The resulting hierarchies are called *ADB trees*. In the following, we explicitly mention the type of BV only if necessary.

Our function computeProb(A, B) needs to estimate the probability $Pr[c(A \cap B) \geq x]$ that is defined in the previous section. However, partitioning $A \cap B$ during runtime is too expensive.

Therefore, we partition each BV during the construction of the hierarchy into a fixed number of cuboidal cells, (the partitioning is discussed more detailed in Section 4.2.2) and then we count the number of *possible collision cell* according to Definition 2 and store it with the node. Note that, thanks to our average-case approach making the assumption that each cell of the partitioning has the same probability to be a possible collision cell, we are not interested in exactly which cells are possible collision cells, but only in their number. As a consequence, this additional parameter per node incurs only a very small increase in the memory footprint of the BV hierarchy, even when utilizing very "light-weight" nodes such as spheres [13] or restricted boxes [31]. It is, of course, computed during preprocessing after the construction of the BV hierarchy.

Note that we do not need to store any polygons or pointers to polygons in leaf nodes. A possible intersection is determined solely based on the probabilities described so far.

In addition to the ADB-trees, we will need a number of lookup tables in order to compute $Pr[c(A \cap B) \geq x]$ efficiently (see Section 3.5). Fortunately, they do not depend on the objects nor on the type of BV, so we need to precompute the lookup tables only once.

3.4 Probability Parameters

As will be explained in Section 3.5, $Pr[c(A \cap B) \geq x]$ can be computed from the following 3 parameters only:

$$\begin{aligned} s &= \#\text{ cells contained in } A \cap B, \\ s_A &= \#\text{ possible collision cells from } A \text{ in } A \cap B, \\ s_B &= \#\text{ possible collision cells from } B \text{ in } A \cap B. \end{aligned}$$

In this section, we explain how to determine them during the collision detection process. Figure 5 gives an overview of the algorithm computeProb(A, B).

```
computeProb(A, B)
    compute s, s_A, s_B;
    look up for Pr[c(A ∩ B) ≥ x]
            using (s, s_A, s_B);
    estimate Pr[P(A) ∩ P(B) ≠ ∅] by
        Pr[c(A ∩ B) ≥ x] and LB(c_{A∩B});
```

Figure 5: computeProb(A, B) estimates the probability $Pr[P(A) \cap P(B) \neq \varnothing]$ by only 3 parameters that can be efficiently computed on-the-fly.

3.4.1 Equally sized BVs

For a moment, let us assume that BVs A and B are of the same size and that the polygons are evenly distributed in both of them. Later in this section, we will lift both assumptions. Let $c(A)$ denote the number of cells lying in A. Then, the number s of cells in $A \cap B$ can easily be approximated by

$$s = \frac{\mathrm{Vol}(A \cap B)}{\mathrm{Vol}(A)} \cdot c(A) \tag{2}$$

Analogously, the parameters s_A and s_B are computed depending on the number of possible collision cells of A and B that have been determined during preprocessing.

Obviously, the cells of the preprocessing partitioning of A and B are not congruent with the cells of the partitioning of $A \cap B$. But congruence is not needed, because our probability computations are only based on the *number* of possible collision cells and the *number* of cells lying in $A \cap B$; they are not based on geometrical properties.

3.4.2 Differently sized BVs

Now, consider the case that A and B are of different size. Without loss of generality, the cells in A are smaller than the cells in B. Then, we first compute s and s_A as described above. If we would also compute s_B this way, we would get too small a probability of collision because the number of possible collision cells would be assumed too small. In practice, the quality of the collision detection would not be affected but the performance, because the traversal would stop later than necessary.

As a remedy, we have to compute s_B depending on a partitioning with a cell size equal to the cell size of the BV A. Therefore, we look for child nodes of B whose sizes are (almost) equal to the size of A and compute s_B depending on these nodes. As a consequence, we have to traverse to the child nodes of B and we stop the traversal at a node B_i if $\mathrm{Vol}(B_i) \leq \mathrm{Vol}(A)$. Let $pc(B_i)$ denote the number of possible collision cells lying in B_i. Then, we compute s_{B_i} by

$$s_{B_i} = \frac{\mathrm{Vol}(B_i \cap A)}{\mathrm{Vol}(B_i)} \cdot pc(B_i) \tag{3}$$

and s_B depending on the n nodes B_i where the traversal was aborted

$$s_B = \sum_{i=1}^{n} s_{B_i}. \tag{4}$$

In the case the traversal has reached a leaf node B_i, it could happen that $\mathrm{Vol}(B_i)$ is still larger than $\mathrm{Vol}(A)$. Then, we compute s_{B_i} by Equation 3 and derive $s_{B'_i}$ that denotes the number of possible collision cells lying in $A \cap B_i$ depending on a partitioning, where the cell size is equal to that of A.

It is self-evident to compute $s_{B'_i}$ by increasing s_{B_i} depending on the ratio between the sizes of A and B_i. But if we would only increase s_{B_i} by $\tau_{AB_i} := \mathrm{Vol}(B_i)/\mathrm{Vol}(A)$ it could happen that this would be an overestimate. Figure 6 illustrates the problem. A single cell from the partitioning of the BV B_i (left) encloses the same volume as the 8 cells in the middle and as the 27 cells in the right. The cells in the middle and on the right have the same volume as the cells of a BV A. So, the cell sizes differ by a factor of 8 and 27, but as one can see, the number of possible collision cells differs only by $8 \cdot 1/2 = 4$ and $27 \cdot 1/3 = 9$.

Of course, this is only correct, if the polygons are aligned as shown in the figure. But as we are dealing with leaf nodes, the assumption that the polygons can be approximated by a single plane (as shown in our figure) is realistic for real-world models. We will discuss this later in detail in Section 3.5.3. As a consequence, we compute $s_{B'_i}$ by Equation 5 and use this instead of s_{B_i} for evaluating Equation 4.

$$s_{B'_i} = s_{B_i} \cdot \tau_{AB_i} \cdot \frac{1}{\sqrt[3]{\tau_{AB_i}}} \tag{5}$$

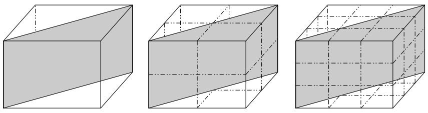

Figure 6: While the cell sizes differ by $\tau_{AB_i} = \mathrm{Vol}(B_i)/\mathrm{Vol}(A)$, the number of possible collision cells differs by $\tau_{AB_i} \cdot 1/\sqrt[3]{\tau_{AB_i}}$.

3.4.3 Uneven polygon distributions

Until now, we have assumed that the polygons are evenly distributed in the BVs A and B. In practice, this is obviously not always correct so that the parameters s, s_A, and s_B could lead to probabilities that are too low or too high.

As a remedy, we propose a two-level partitioning of each BV in the hierarchy: first, we partition the BV by a grid or pyramidal sectors (see Figure 7) into a (small) number of regions. Then, for each region, we compute the number of possible collision cells as described above. That way, we can capture the polygon distribution by a set of values, instead of only one. During collision detection, a better estimate of the number of possible collision cells lying in the intersection volume can be determined by considering only those regions that lie in the intersection. Let $R_1, \ldots, R_n$ denote the n regions of a BV A and $pc(R_j)$ the number of possible collision cells corresponding to the region R_j. Then, s_A can be computed as follows:

$$s_A = \sum_{j=1}^{n} pc(R_j) \cdot \frac{\mathrm{Vol}(R_j \cap B)}{\mathrm{Vol}(R_j)}.$$

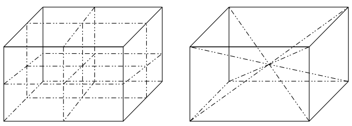

Figure 7: A BV can be subdivided into regions in order to capture uneven polygon distributions.

3.5 Probability Computations

In this section, we explain the computation of the probability $Pr[c(A \cap B) \geq x]$ and its usage.

3.5.1 Probability of collision cells

Recall that computeProb(A, B) estimates the probability that polygons contained in two BVs A and B intersect (see Figure 5). Given a partitioning of $A \cap B$ and the numbers s, s_A, s_B, the question is: what is the probability that at least x of the s cells are possible collision cells of the s_A cells and are *also* possible collision cells of the s_B cells? This is the probability that at least x collision cells exit.

Note that the $s_A + s_B$ possible collision cells are randomly but not independently distributed among the s cells: obviously, it can never happen that two or more of the s_A or s_B, resp., possible collision cells are distributed on the same cell, i.e., s_A possible collision cells are distributed on exactly the same number of cells of the partitioning. This problem can be stated more abstractly and generalized by the following definition.

Definition 6 ($Pr[\#\ \textit{filled bins} \geq x]$) Given u bins, v blue balls, and w red balls. The balls are randomly thrown into the u bins, whereby a bin never gets two or more red or two or more blue balls. The probability that at least x of the u bins get a red and a blue ball is denoted as $Pr[\#\text{ filled bins } \geq x]$.

If $u = s, v = s_A$ and $w = s_B$, this definition is related to our original problem by the following observation, because we assume that each cell of the partitioning has the same probability of being a possible collision cell.

Observation 1
$Pr[c(A \cap B) \geq x] \approx Pr[\#\text{ filled bins } \geq x]$.

Now, let us determine $Pr[\#\text{ filled bins } \geq x]$. The probability, that exactly t of the u bins get a red and a blue ball, is

$$\frac{\binom{w}{t}\binom{u-w}{v-t}}{\binom{u}{v}}$$

Explanation: Let us assume, the w red balls have already been thrown into the u bins. Now, the question is: what is the probability that t of the v blue balls are thrown into bins containing a red ball? $\binom{u}{v}$ is the number of possibilities to distribute the v blue balls to the u bins. The number of possibilities that t of the v blue balls are distributed to bins already filled with a red ball is $\binom{w}{t}$. And the remaining $v-t$ blue balls have to be distributed simultaneously to the $u-w$ empty bins.

Thus, the probability that at least x of the u bins get a red and a blue ball, is

$$Pr[\#\text{ filled bins } \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{w}{t}\binom{u-w}{v-t}}{\binom{u}{v}} \tag{6}$$

In the case $x = 1$, this equation can be simplified yielding

$$\begin{aligned} Pr[\#\text{ filled bins } \geq 1] &= 1 - \frac{\binom{u-w}{v}}{\binom{u}{v}} \\ &\overset{v \geq 0}{=} 1 - \prod_{i=1}^{v}\left(1 - \frac{w}{u-v+i}\right). \end{aligned} \tag{7}$$

It is obvious that in Equation 7, $u - v$ has to be computed only once. Therefore, $4v + 1$ operations are necessary to compute $Pr[\#\text{ filled bins } \geq 1]$ and in the worst case (where $v = 8^3 \wedge w = 8^3$) $4 \times 8^3 + 1 = 2049$ operations are necessary to calculate $Pr[\#\text{ filled bins } \geq 1]$ for a single pair of nodes. Of course, this would be too expensive during runtime. Therefore, we compute a complete lookup table for $Pr[\#\text{ filled bins } \geq x]$, which has to be done only once for our algorithm. Note that $Pr[\#\text{ filled bins } \geq x]$ is completely independent of the type of BV or model.

3.5.2 Probability of collision

Until now, for computing a lower bound for $Pr[P(A) \cap P(B) \neq \varnothing]$ (see Equation 1) we have only used the probability that at least *one* collision cell exists in $A \cap B$. Although the algorithm achieves very good quality using only that probability, we can improve the lower bound by using the probability that *several* collision cells are in the intersection, i.e., by using $Pr[c(A \cap B) \geq x]$, $x > 1$. Obviously, $Pr[c(A \cap B) \geq x]$ decreases as x increases. But the more collision cells (with high probability) in the intersection volume are, the higher the probability is that a collision really takes place in the pair of BVs.

Let a partitioning of $A \cap B$ be given. Then, a lower bound for the probability $Pr[P(A) \cap P(B) \neq \varnothing]$ can be computed by

$$Pr[P(A) \cap P(B) \neq \varnothing] \geq \qquad (8)$$
$$\max_{x \leq \min\{s_A, s_B\}} \left\{ Pr[c(A \cap B) \geq x] \cdot \left(1 - (1 - LB(c_{A\cap B}))^x\right) \right\}$$

because $\left(1 - (1 - LB(c_{A\cap B}))^x\right)$ denotes a lower bound for the probability that in at least one of the x collision cells a collision takes place. Note that, if we use the approximation shown in Observation 1, this is not a lower bound any longer, but *only* a good estimation of it.

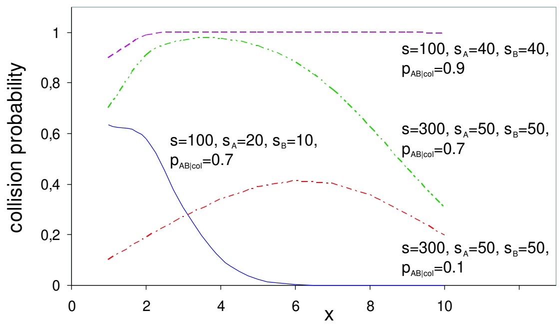

Figure 8: Usually, $\left(1 - (1 - LB(c_{A\cap B}))^x\right) \cdot Pr[c(A \cap B) \geq x]$ is maximal for $x \leq 10$.

In practice, it is sufficient to evaluate Equation 8 for small x, because for realistic values of s, s_A, s_B, and $LB(c_{A\cap B})$ it assumes the maximum at a small x (see Figure 8). Consequently, we bound x by a small number (e.g., 10) in Equation 8. Summarizing this section, in order to get a better lower bound for the collision probability, $Pr[P(A) \cap P(B) \neq \varnothing]$ can be computed by Equation 8 instead of Equation 1.

3.5.3 Probability of intersection in a cell

It remains to derive $LB(c_{A\cap B})$, which denotes a lower bound for the probability of an intersection in an arbitrary collision cell from the partitioning of $A \cap B$ (see Defintion 4).

For most real-world models, we can assume that the curvature of the surfaces is bounded (possibly except in a finite number of curves on the surface). Now consider a collision cell $c \subset A \cap B$, i.e., it contains two sets of polygons, $P(A)$ and $P(B)$. Because we are looking for a lower bound, we can only assume that $\text{Area}_c(A)$ and $\text{Area}_c(B)$ are equal to $\text{MaxArea}(c)$.

Suppose the cell c is large compared to the size of the objects. Then, the possible curvature of the surface parts in $P(A)$ and $P(B)$ can lead to convex hulls of $P(A)$ and $P(B)$ that are small with respect to c. Therefore, the probability of an intersection is small.

On the other hand, as the traversal of the BV hierarchies reaches lower levels, c becomes small compared to the size of the objects. Then, because of the bounded curvature, $P(A)$ and $P(B)$ can be approximated better and better by two plane polygons. In the extreme, we reach exactly the situation shown in Figure 4. Therefore, we estimate the lower bound $LB(c_{A\cap B})$ by

$$LB(c_{A\cap B}) \approx \frac{d_A + d_B}{d_{max_A} + d_{max_B}}$$

where d_A, d_B are the depth of node A, B in their respective BV hierarchies, and d_{max_A}, d_{max_B} are the maximum depths. In other words, the larger the depth of the nodes of A and B, the smaller the BVs, and the larger is the probability that the polygons in a collision cell intersect.

Note that if we approximate $LB(c_{A\cap B})$ as described above, the lower bound given in Equation 8 is not a lower bound any longer, but *only* a good estimation of it.

3.6 Intersection Volume

Since Equation 2 has to be computed once per node pair during the hierarchy traversal, we need a fast way to compute $\text{Vol}(A \cap B)$. However, an exact computation is prohibitively expensive for most BVs (except spheres), even for cubes, because they are not aligned with each other. So we need to resort to approximations. A simple and efficient one is to enclose the BVs by spheres and compute their intersection volume. Unfortunately, this is also a very inaccurate one. In the case of DOPs (note that cubes are special 6-DOPs), we could also enclose one of A or B by an aligned DOP efficiently [30]. Another method that works only for boxes (and similarly for DOPs) is to choose three slabs out of the 2×3 slabs defined by two boxes A and B [7]. Figure 9 illustrates this idea. The volume of the resulting parallelepiped can be computed by the scalar triple product.

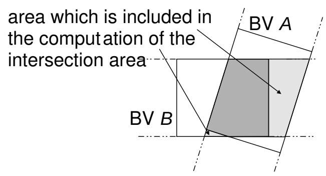

Figure 9: The intersection volume (highlighted area) could be approximated by taking the three best slabs [7]. For the sake of illustration, the figure is in 2D.

Since the above mentioned methods can all produce fairly gross overestimations, and since the latter two methods are also quite expensive (for our purpose), we propose a different method. The idea is shown in Figure 10.

Given two bounding boxes of (nearly) the same size at a certain distance d that are not necessarily aligned with each other. Then, an upper bound of their intersection volume is given by two BVs of the same size with the same distance d, that are aligned as one of the 3 cases shown in the Figure 10. So we only need to tentatively compute the intersection volume V_i for each of them. Then, $\max\{V_1, V_2, V_3\}$ is an upper bound of the intersection volume. In the following, let a, b, and c denote the side lengths of the BVs, where $a \geq b \geq c$. Then

$$\begin{aligned} V_1 &= a \cdot b \cdot c \cdot (1 - \frac{d}{\sqrt{a^2}})^1 = (a - d) \cdot b \cdot c \\ V_2 &= a \cdot b \cdot c \cdot (1 - \frac{d}{\sqrt{a^2 + b^2}})^2 \\ V_3 &= a \cdot b \cdot c \cdot (1 - \frac{d}{\sqrt{a^2 + b^2 + c^2}})^3 \end{aligned}$$

To prove this claim, one has to perform two steps. First of all, assume that the boxes are axis-aligned. Without loss of generality, let one box be centered at the origin and the other centered at $P = (x, y, z)$. Then, the intersection volume is $V = (a - x)(b - x)(c - x)$, which has to be maximized under the constraint that $x^2 + y^2 + z^2 = d^2$. Then, one has to show that $V \leq \max\{V_1, V_2, V_3\}$ always holds. In the second step, one has to prove that for a rotated BV the intersection volume is smaller or equal than when aligning that BV while keeping the same distance. If the difference of the size of the two bounding boxes is above a certain threshold, we use the intersection volume of the two bounding spheres as an estimate. In our experience this seems to work well.

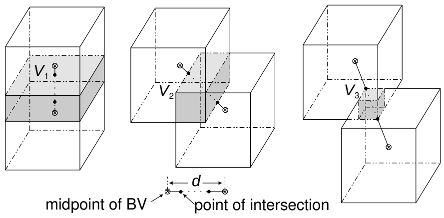

Figure 10: We estimate the intersection volume for two not necessarily aligned BVs by the maximum of three corresponding aligned cases, $\max\{V_1, V_2, V_3\}$, which is an upper bound. Note that, also for non-cubic BVs, the line through the midpoints of the two BVs has to intersect the vertices and/or edges of the BV of the intersection volume as shown.

4. RESULTS

Because our approach is applicable to most hierarchical BV hierarchies, we have decided to implement two basic data structures, namely an octree and an AABB tree, that are used in many VR applications and that can easily be turned into ADB-trees. The construction heuristic of the AABB tree is the same as that used for the restricted boxtree [31]. Figure 12 shows the timing results of a comparison between our ADB-trees based on the octree and the AABB tree. Obviously, the AABB tree performs better by a factor > 3 and we have obtained similar results with all other models. Therefore, all the following benchmarks were performed using our ADB-tree based on AABBs.

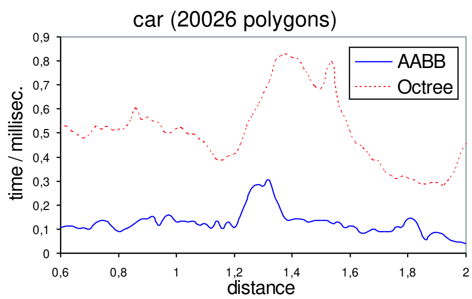

Figure 12: Comparison of our ADB-trees based on an octree and on an AABB tree. Here, the results for the car body with 20,000 polygons are shown, but we have obtained similar ones for all other objects of our suite.

We implemented our new algorithm in C++. As of yet, the implementation is not fully optimized. In the following, all results have been obtained on a 2.4 GHz Pentium-IV with 1 GB main memory.

4.1 Benchmark Scenario

For timing the performance and measuring the quality of our algorithm, we have used a set of CAD objects, each of them with varying complexities (see Figure 11 in the Color Plates Appendix).

Benchmarking is performed by the procedure proposed in Zachmann [31], which computes average collision detection times for a range of distances between two identical objects.

Figure 13 shows a snapshot of this benchmarking procedure, where one pair of BVs has been highlighted, which contain at least one collision cell with high probability.

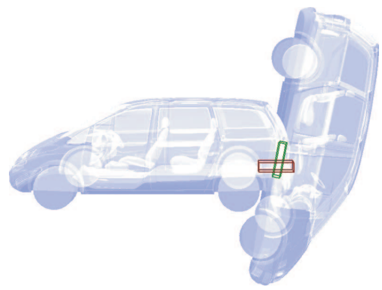

Figure 13: A snapshot of the benchmarking procedure (objects are rendered transparent). The two boxes show one pair of BVs that contain at least one collision cell with high probability.

BV hierarchy	Bytes
ADB tree based on AABBs	36
sphere tree	16
AABB tree	28
OBB tree	64
24-DOP tree	100

Table 1: This table compares the amount of memory per node for our ADB-tree with some traditional ones.

4.2 Preprocessing

4.2.1 *Memory requirements*

Table 1 summarizes the number of bytes per node for different BV hierarchies. Note that we do not need to store any polygons or pointers to polygons in leaf nodes. Therefore, we need exactly the same number of bytes for each node in our hierarchy. We use 24 bytes for storing two vectors that define the BV, 4 bytes for a pointer to child nodes, 4 bytes for storing possible collision cells and 4 bytes for storing the volume of the BV.

4.2.2 *Partitioning of BVs*

As mentioned in Section 3.1, the quality of the collision detection depends, to some extent, on the number of cells a BV is partitioned into. Our experiments have shown, that 8^3 cells per node are sufficient: if the nodes are partitioned into more cells, the collision quality (in the sense of the error rate) does not seem to improve and if less cells are chosen, the quality gets worse. Of course, the finer the partitioning, the more possible collision cells are stored at each node and the larger are s_A and s_B — but also the larger is s. So, the probability $Pr[\#\text{ filled bins } \geq x]$ does not necessarily increase if the number of cells of the partitioning increases.

An other reason for restricting the number of cells by 8^3 is that the finer the subdivision into cells, the larger are the lookup tables (and the higher is the memory consumption) because the values s, s_A and s_B directly depend on the partitioning.

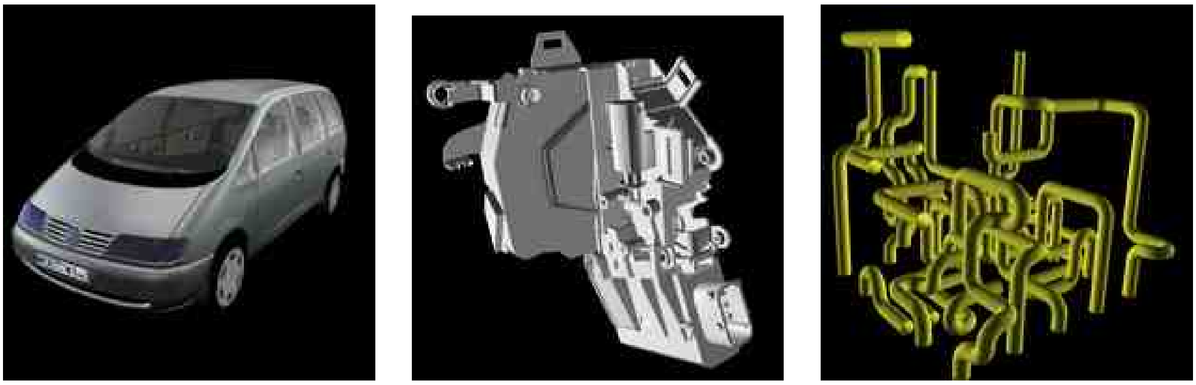

Figure 11: Some objects of our test suite: body of a car, lock of a car door, and a set of pipes. (Data courtesy of VW and BMW)

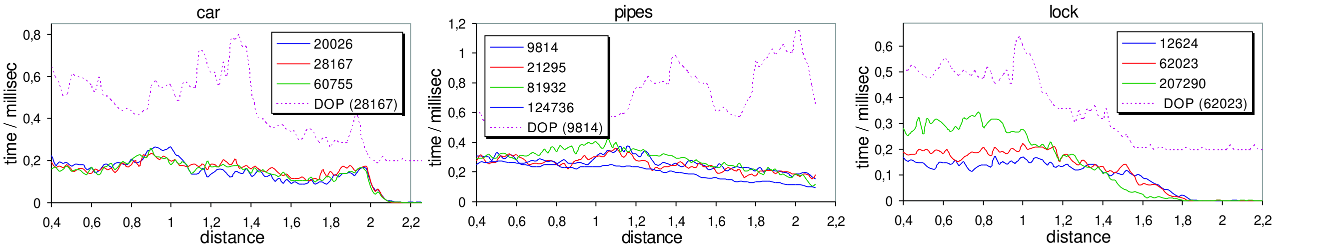

Figure 14: Timings for different models and different polygon counts ($k_{min} = 10$ and $p_{min} = 0.99$). Also, a runtime comparison to a DOP tree is shown (see Section 4.3.3).

4.2.3 Lookup Tables

As mentioned before, we need a lookup table for the probabilities $Pr[\#\text{ filled bins } \geq x]$ with $u, v, w \in \{1, \ldots, 512\}$ and $x \leq 10$ (see Section 3.5.2). By exploiting the fact that $Pr[\#\text{ filled bins } \geq x]$ for $u = u', v = v', w = w'$ is equal to $Pr[\#\text{ filled bins } \geq x]$ for $u = u', v = w', w = v'$, we can reduce the memory usage of that lookup table by a factor 2. Still, for each x, the lookup table would contain $512 \cdot \sum_{i=1}^{512} i$ entries amounting to about 256 MB. Fortunately, we can reduce the number of entries significantly by exploiting the monotonicity of $Pr[\#\text{ filled bins } \geq x]$ in the variables v and w. If the probability for $u = u', v = v', w = w'$ is close to 1 (e.g., $\geq 1 - 0.9 \cdot 10^{-5}$), then we do not compute the probabilities for $u = u', v = v', w \geq w' + 1$; instead, we continue at $u = u', v = v' + 1, w = v' + 1$. On average, this reduces the number of probabilities for one lookup table by a factor of 17. Figure 15 gives an overview of the memory consumption for storing the lookup tables.

Note that for computing $Pr[\#\text{ filled bins } \geq x]$ the binomial coefficients can become very large. Therefore, in order to compute the probabilities as accurate as possible, one should use arbitrary precision for numeric calculations. Using Maple, the time for computing all ten tables took about 93 hours. As mentioned earlier, these lookup tables are absolutely independent of the models, so that they have to be computed only once and can be stored in a file.

4.3 Performance and Quality

4.3.1 Time and Quality versus Complexity

Each plot in Figure 14 shows the runtime for a model of varying complexity (the legend gives the number of polygons per object). In most cases, the runtime is fairly independent of the complexity. There are exceptions, for instance, the pipes with 9,814 polygons take even slightly longer than the pipes modelled with 124,736 polygons. We conjecture that this is caused by larger polygons in the coarser resolution of the model which allows the construction heuristic less possibilities to construct optimal BV hierarchies.

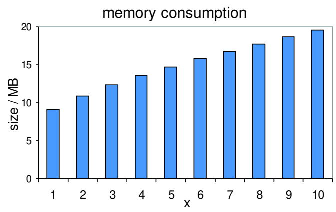

Figure 15: Memory consumption of our ten lookup tables.

Figure 16 shows the error rates corresponding to the timings in Figure 14. Here, the error is defined as the percentage of wrong detections. For measuring them, we have compared our results with an exact approach. Only collision tests are considered where at least the outer BVs, which enclose the whole objects, intersect. Apparently, the error rates are always relatively low and mostly independent of the complexities: on average, only 1.89% (sharan), 1.54% (door lock), and 2.10% (pipes) wrong collisions are reported if the objects have a distance between 0.4 and 2.1, and about 3.19% (sharan), 1.71% (door lock), and 3.15% (pipes) wrong collisions are reported for distances between 1 and 2.

4.3.2 Time versus Quality

In this section, we examine how the runtime depends on the quality of the collision detection.

As mentioned in Section 3.1, the runtime and the qual-

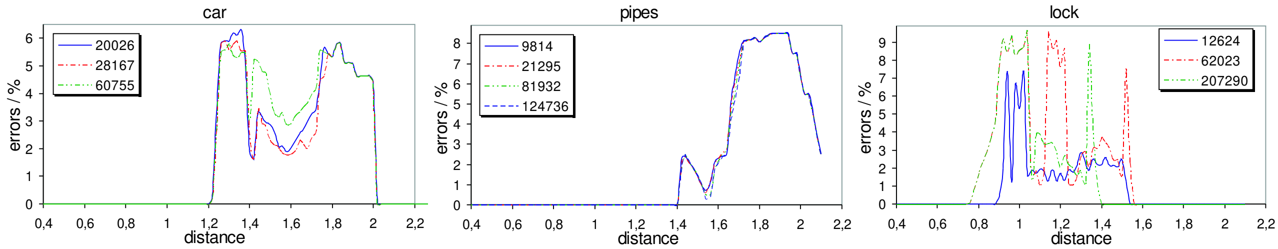

Figure 16: Error rates corresponding to the timings in Figure 14.

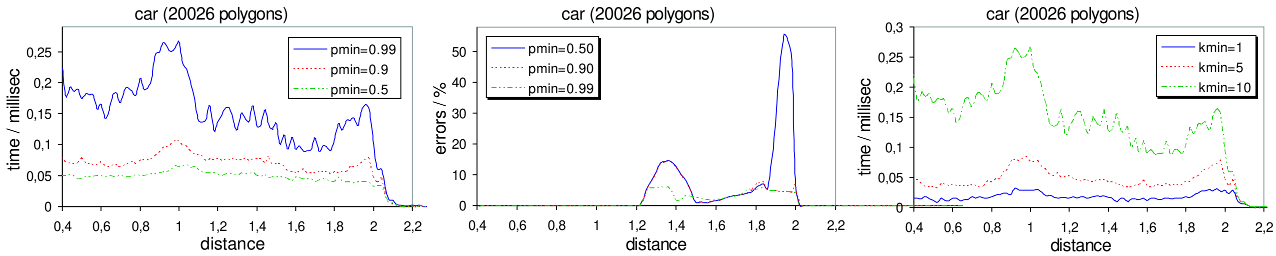

Figure 17: Runtime and error comparisons for the car body with 20,000 polygons. Left, center: timings and error rates for different p_{min} ($k_{min} = 10$); right: Timings for different k_{min} ($p_{min} = 0.99$).

ity can be influenced by the values of p_{min} and k_{min} (see also Figure 3): the smaller p_{min} or k_{min}, the shorter is the runtime and, usually, the more errors are made.

Figure 17 on the left shows the correlation between the runtime and p_{min} (car, 20026 polygons). The corresponding error rates are shown in the middle. Obviously, as p_{min} increases the error rate decreases. There are a few exceptions, where more errors are made when using a larger p_{min}. We conjecture that this is caused by pairs of BVs where corresponding polygons (that do intersect) have a low probability of intersection. For $p_{min} = 0.9$ and $p_{min} = 0.99$ the errors differ only by less than 2% points while for $p_{min} = 0.9$ and $p_{min} = 0.5$ the errors differ by about 5% points on average (object distances between 1.2 and 2).

In Figure 17 (right) the timings for different k_{min} (the number of pairs of collision nodes that have to be found before the traversal stops) are compared (car, 20026 polygons). Due to space limitations, the corresponding errors can only be found in the extended version of this paper [17]. Only about 0.2% points less errors are made if k_{min} increases from 5 to 10, while 2% points less errors occur if k_{min} is changed from 1 to 5 (object distances between 1.2 and 2). Comparing the timings for $k_{min} = 5$ and $k_{min} = 10$, it is questionable whether an increase in accuracy by 0.2% points justifies a decrease in speed by a factor ≈ 2.3.

4.3.3 *Performance comparison*

A runtime comparison between our approach and a DOP tree implementation, where no probabilities are utilized, can be found in Figure 14. We have implemented the DOP tree with the same care as for our new approach. The runtime for the DOP tree is only shown for a single resolution at which the highest performance was achieved using the DOP tree approach. As you can notice, our algorithm is always remarkably faster, e.g., in the case of the car body our new algorithm is ≈ 3 times faster on average ($k_{min} = 10, p_{min} = 0.99$) and > 6 times faster if the error rate may increase by only 0.2% points ($k_{min} = 5, p_{min} = 0.99$, see also Figure 17).

5. CONCLUSION AND FUTURE WORK

In this paper we have presented a general method to turn a conventional hierarchical collision detection algorithm into one that uses probability estimations to decrease the quality of collision detection in a controlled way. To our knowledge, this is the first approach to this problem.

Our algorithm can be utilized to increase the perceived quality of simulations and interactions by increasing the performance without noticeably decreasing the correctness. More precisely, our novel framework can be useful in several ways:

- It allows applications that do not need a precise collision detection to take advantage of that opportunity by specifying a desired *quality* threshold, thus decreasing the collision detection time significantly.
- It allows a scheduler to *interrupt* the collision detection while still allowing the collision detection to make a better effort than in the traditional traversal schemes *and* return a kind of measure of *confidence* in the result.

Note that in both cases, the application can still obtain meaningful contact information in order to handle the collision.

Our approach is made possible by augmenting traditional BV hierarchies with just a few additional parameters per node, which are utilized during traversal to efficiently compute the probability of a collision occurring among the polygons of a pair of BVs. These probabilities are then used as priorities to direct the traversal into those parts of the BV trees with higher probability.

We have implemented our new ADB-trees (average-distribution trees) based on AABBs and octrees and present performance measurements and comparisons with a fast traditional algorithm, namely the DOP-tree. The results show a speedup of about a factor 3 to 6 with only about 4% error on average. Furthermore, error rates and performance are almost independent of the number of polygons.

Currently, we are investigating the exploitation of curvature distributions in order to improve error rates and performance.

We are also working on the application of our approach to other BV hierarchies, in particular DOP trees and Restricted Boxtrees. In addition, we would like to apply it to non-hierarchical data structures for deformable collision detection.

An interesting extension of our new algorithm would be the modelling of contacts, and an estimation of separation distance or penetration depth.

Since we can now better control the error of the collision detection, we think it should now be possible to establish a precise correlation between that error and the perceived correctness [24].

There are many areas in computer graphics that utilize BV hierarchies, such as ray tracing, occlusion culling, or shadow rendering. So a natural question is if an average-case approach can be applied there too.

6. REFERENCES

[1] P. K. Agarwal, J. Basch, L. J. Guibas, J. Hershberger, and L. Zhang. Deformable free space tiling for kinetic collision detection. In *Algorithmic and Computational Robotics: New Directions (Proc. 5th Workshop Algorithmic Found. Robotics)*, pages 83–96, 2001.

[2] P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H. J. Haverkort. Box-trees and R-trees with near-optimal query time. In *Proc. Seventeenth Annual Symposium on Computational Geometry (SCG 2001)*, pages 124–133, 2001.

[3] S. Ar, B. Chazelle, and A. Tal. Self-customized BSP trees for collision detection. *Computational Geometry: Theory and Applications*, 15(1–3):91–102, 2000.

[4] S. Ar, G. Montag, and A. Tal. Deferred, self-organizing BSP trees. In *Eurographics*, pages 269–278, 2002.

[5] G. Baciu and W. Sai-Keung Wong. Hardware-assisted self-collision for deformable surfaces. In *Proc. VRST 2002*, pages 129–136, Hong Kong, China, 2002.

[6] G. Baciu and W. Sai-Keung Wong. Image-based techniques in a hybrid collision detector. *IEEE Transactions on Visualization and Computer Graphics*, 9(2):254–271, 2003.

[7] W. Barth and E. Huber. Computations with tight bounding volumes for general parametric surfaces. In *Proc. 15th European Workshop on Computational Geometry (CG 1999)*, pages 123–126, 1999.

[8] R. Barzel, J. Hughes, and D. N. Wood. Plausible motion simulation for computer graphics animation. In *Proc. Eurographics Workshop Computer Animation and Simulation*, pages 183–197, 1996.

[9] J. Dingliana and C. O'Sullivan. Graceful degradation of collision handling in physically based animation. *Proc. Eurographics 2000*, 19(3):239–247, 2000.

[10] S. A. Ehmann and M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Proc. Eurographics 2001*, 20(3):500–510, 2001.

[11] S. Fisher and M. Lin. Fast penetration depth estimation for elastic bodies using deformed distance fields. In *Proc. International Conf. on Intelligent Robots and Systems (IROS)*, 2001.

[12] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *SIGGRAPH 1996 Conference Proc.*, pages 171–180, 1996.

[13] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996.

[14] S. Huh, D. N. Metaxas, and N. I. Badler. Collision resolutions in cloth simulation. In *IEEE Computer Animation Conf.*, pages 122–127, Seoul, Korea, 2001.

[15] Y. Kitamura, A. Smith, H. Takemura, and F. Kishino. A real-time algorithm for accurate collision detection for deformable polyhedral objects. *Presence*, 7(1):36–52, 1998.

[16] J. Klein, J. Krokowski, M. Fischer, M. Wand, R. Wanka, and F. Meyer auf der Heide. The randomized sample tree: A data structure for interactive walkthroughs in externally stored virtual environments. In *Proc. VRST 2002*, pages 137–146, Hong Kong, China, 2002.

[17] J. Klein and G. Zachmann. Controlling the error of time-critical collision detection using ADB-trees. Tech. Rep. tr-ri-03-242, Institute of Computer Science, University of Paderborn, 2003. `http://www.upb.de/cs/janklein`.

[18] J. T. Klosowski, M. Held, J. S. B. Mitchell, H. Sowrizal, and K. Zikan. Efficient collision detection using bounding volume hierarchies of k-DOPs. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):21–36, 1998.

[19] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics*, pages 325–333, 2001. short presentation.

[20] R. W. H. Lau, O. Chan, M. Luk, and F. W. B. Li. A collision detection method for deformable objects. In *Proc. VRST 2002*, pages 113–120, 2002.

[21] J.-C. Lombardo, M.-P. Cani, and F. Neyret. Real-time collision detection for virtual surgery. In *Proc. of Computer Animation*, pages 82–90, Geneva, Switzerland, 1999.

[22] W. A. McNeely, K. D. Puterbaugh, and J. J. Troy. Six degrees-of-freedom haptic rendering using voxel sampling. In *SIGGRAPH 1999 Conference Proc.*, pages 401–408, 1999.

[23] K. Myszkowski, O. G. Okunev, and T. L. Kunii. Fast collision detection between complex solids using rasterizing graphics hardware. *The Visual Computer*, 11(9):497–512, 1995.

[24] C. O'Sullivan, R. Radach, and S. Collins. A model of collision perception for real-time animation. In *Proc. 1999 Conference on Computer Animation and Simulation - Eurographics Workshop (EGCAS 1999)*, pages 67–76, 1999.

[25] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Proc. Eurographics 1995*, 14 (2):105–116, 1995.

[26] M. Shinya and M.-C. Forgue. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, 2(4):132–134, 1991.

[27] S. Uno and M. Slater. The sensitivity of presence to collision response. In *Proc. VRAIS*, page 95, Albuquerque, New Mexico, 1997.

[28] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, 2(4):1–14, 1997.

[29] M. Wand, M. Fischer, I. Peter, F. Meyer auf der Heide, and W. Straßer. The randomized z-buffer algorithm: Interactive rendering of highly complex scenes. In *SIGGRAPH 2001 Conference Proc.*, pages 361–370, 2001.

[30] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. VRAIS 1998*, pages 90–97, Atlanta, Georgia, 1998.

[31] G. Zachmann. Minimal hierarchical collision detection. In *Proc. VRST 2002*, pages 121–128, Hong Kong, China, 2002.

[1] We wish to thank Birgitta Derenthal, and Jens Krokowski for valuable discussion and their support. This work is partially supported by DFG grant ME 872/8-1, the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT), and the DFG program "Aktionsplan Informatik" by grant ZA292/1-1.