ADB-Trees: Controlling the Error of Time-Critical Collision Detection

Jan Klein

Heinz Nixdorf Institute and
Institute of Computer Science
University of Paderborn, Germany
Email: `janklein@upb.de`

Gabriel Zachmann

Dept. of Computer Graphics and Virtual Reality
University of Bonn, Germany
Email: `zach@cs.uni-bonn.de`

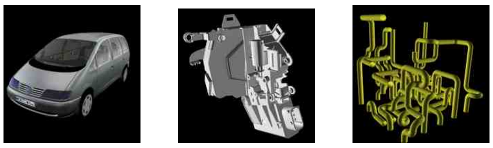

Figure 1: Some objects of our test suite: body of a car, lock of a car door, and a set of pipes. (Data courtesy of VW and BMW)

Abstract

We present a novel framework for hierarchical collision detection that can be applied to virtually all bounding volume (BV) hierarchies. It allows an application to trade quality for speed. Our algorithm yields an estimation of the quality, so that applications can specify the desired quality. In a time-critical system, applications can specify the maximum time budget instead, and quantitatively assess the quality of the results returned by the collision detection afterwards.

Our framework stores various characteristics about the average distribution of the set of polygons with each node in a BV hierarchy, taking only minimal additional memory footprint and construction time. We call such augmented BV hierarchies ***a**verage-**d**istri**b**ution trees* or *ADB-trees*.

We have implemented our new approach by augmenting AABB trees 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.

1 Introduction

Fast collision detection of polygonal objects is needed in many highly interactive applications such as virtual prototyping and 3D games. 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 [18]. At the same time, humans cannot distinguish between physically correct and *physically plausible* behavior of objects (at least up to some degree) [4]. Since collision detection is still the major bottleneck of many of these simulations, 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.

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.

2 Related Work

Bounding volume hierarchies have proven to be a very efficient data structure for rigid collision detection, and, to some extent, 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 [8, 16], OBBs [7], DOPs [12, 21], Boxtrees [22, 1], AABBs [19, 13], and convex hulls [6].

Alternatives to BV hierarchies are approaches that utilize the graphics hardware [17,15,14]. However, all of them compete with the rendering module for the graphics resources (unless one spends another board just for the collision detection).

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 [5, 8]. Hubbard presented the idea of interruptible collision detection using sphere trees [8]. Dingliana and O'Sullivan [5] 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 [2] or at runtime [3] (i.e., the construction is done on-demand). 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 [9] and the randomized z-buffer [20]. 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 Controlling the Error

Virtually all hierarchical collision detection approaches traverse the hierarchies simultaneously by an algorithm, which allows to quickly zoom into areas of close proximity. 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 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 algorithm.

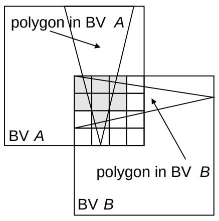

Figure 2: We partition the intersection volume by a grid. Then, we determine the probability that there are *collision cells* where polygons of different objects *can* intersect (highlighted in grey).

3.1 Overview of our Approach

Our idea is to guide and to abort the traversal by the *probability* that a pair of BVs contains intersecting polygons.

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 number of possible collision cells from A and B, resp., 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.

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 subtrees 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.

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.

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.

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.

```
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 $p \geq p_{min}$
                k:=k+1;
                if $k \geq k_{min}$ 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.

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 $\text{Area}_c(A)$.

$\text{MaxArea}(c)$ 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 $\text{Area}_c(A) \geq \text{MaxArea}(c)$.

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 $\exists c : \text{Area}_c(A) \geq \text{MaxArea}(c) \wedge \text{Area}_c(B) \geq \text{MaxArea}(c)$ (with suitably chosen $\text{MaxArea}(c)$).

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 $\text{Area}_c(A) = \text{Area}_c(B) = \text{MaxArea}(c)$, then we must have exactly the configuration shown in Figure 4, i.e., an intersection, if we choose $\text{MaxArea}(c) = a^2\sqrt{2}$. Obviously, a set of polygons is not planar (usually), so even if $\text{Area}_c(A) > \text{MaxArea}(c)$ 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 $LB(c_{A\cap B})$.

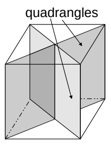

Figure 4: A cubic collision cell c with side length a. $\text{Area}_c(A)$ and $\text{Area}_c(B)$ must be at least $\text{MaxArea}(c) = a^2\sqrt{2}$, which is exactly the area of the two quadrangles.

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 3.6.2, where $x > 1$ is used for $Pr[c(A \cap B) \geq x]$. Section 3.6.1 will derive $Pr[c(A \cap B) \geq x]$, while Section 3.6.3 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, 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. It is computed during preprocessing after the construction of the BV hierarchy. We explain this computation more precisely in Section 3.4.

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 3.6). 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 Counting Possible Collision Cells

We propose two algorithms for computing the number of possible collision cells. It is convenient to do this after the BV hierarchy has already been built.

The first one is very simple and computes an exact value of the number. It partitions each node into a grid, clips the polygons associated with the node, inserts the fragments into the grid, and counts the number of possible collision cells. Assuming a complete binary BV hierarchy, the runtime of this algorithm can be estimated as

$$T_1 = c_1 n \log m + c_1' m$$

where n is the number of polygons and m is the number of nodes in the hierarchy. c_1 is the cost of clipping and inserting one polygon, while c_1' is the (average) cost of counting the number of possible collision cells of one node.

```
posColCells($A, A'$)
    $pc := 0$;
    if Vol$(A') \leq$ Vol$(c_A)$ then
      if Area$(A') \geq$ MaxArea$(c_A)$ then
        return 1;
    else
      if $A'$ is a leaf then
        if Area$(A') \geq$ MaxArea$(A')$ then
          return $1 \cdot \frac{\text{Vol}(A')}{\text{Vol}(c_A)} \cdot \frac{1}{\sqrt[3]{\frac{\text{Vol}(A')}{\text{Vol}(c_A)}}}$;
      else
        for all children $A'[i]$ do
          $pc := pc+$ posColCells$(A, A'[i])$;
    return $pc$;
```

Figure 5: The number of possible collision cells in BV A can be approximated efficiently by propagating polygon areas up through the tree. c_A denotes an arbitrary cell of A, $pc(A) :=$ posColCells(A, A).

The second algorithm approximates the number of possible collision cells, $pc(A)$, for a node A by the algorithm shown in Figure 5. Its main idea is to use the sub-tree of A for the computation of $pc(A)$. The algorithm looks for child nodes A' of A with Vol$(A') \leq$ Vol(c_A), which is the size of one cell of A. If such a child node contains *enough* polygons (in some sense), then we increase $pc(A)$ by 1. Therefore, we do not need to partition A into a grid and test each cell. Of course, the recursive search for such cells could end at a leaf node A'. Then, if this node contains *enough* polygons, we approximate the number of possible collision cells by $pc(A') := 1 \cdot \frac{\text{Vol}(A')}{\text{Vol}(c_A)} \cdot \frac{1}{\sqrt[3]{\frac{\text{Vol}(A')}{\text{Vol}(c_A)}}}$. Due to space limitations, this is only discussed in the extended version of this paper [10].

Let c_2 denote the cost for checking one node if it is a possible collision cell. Then, the runtime for computing $pc(A)$ for all nodes can be estimated by

$$\begin{aligned} T_2 &= \sum_{i=0}^{d} c_2 2^i \left(\frac{2^{d+1}}{2^i} - 2\right) \\ &= c_2 2^{d+1}(d-1) + 2 \leq c_2 m \log m \end{aligned}$$

because for a node with depth i maximal $2^{d-i+1} - 2$ child nodes have to be checked.

Obviously, T_2 is better, because $c_1' \gg \log m$, and because c_1 is a very expensive operation compared to c_2. Indeed, our experiments in Section 4 show that our second algorithm is substantially faster so that it can be performed at startup time.

3.5 Probability Parameters

As will be explained in Section 3.6, $Pr[c(A \cap B) \geq x]$ can be computed from 3 parameters only:

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

They can be determined very fast during the collision detection process [11]. Figure 6 gives an overview of the algorithm computeProb(A, B).

```
computeProb($A, B$)
    compute $s, s_A, s_B$;
    look up for $Pr[c(A \cap B) \geq x]$
             using $(s, s_A, s_B)$;
    estimate $Pr[P(A) \cap P(B) \neq \varnothing]$ by
       $Pr[c(A \cap B) \geq x]$ and $LB(c_{A \cap B})$;
```

Figure 6: 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.

3.6 Probability Computations

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

3.6.1 Probability of collision cells

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. This problem can be stated more abstractly and generalized by the following definition.

Definition 6 (Pr[# $filled\ bins \geq x$]) 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 $\binom{w}{t}\binom{u-w}{v-t}/\binom{u}{v}$.

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

$$Pr[\text{\# filled bins } \geq x] = 1 - \sum_{t=0}^{x-1} \frac{\binom{w}{t}\binom{u-w}{v-t}}{\binom{u}{v}} \quad (2)$$

3.6.2 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 $Pr[c(A \cap B) \geq x], x > 1$.

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

$$Pr[P(A) \cap P(B) \neq \varnothing] \geq \max_{x \leq \min\{s_A, s_B\}} \Big\{ Pr[c(A \cap B) \geq x] \cdot \big(1 - (1 - LB(c_{A\cap B}))^x\big) \Big\} \quad (3)$$

because $\big(1 - (1 - LB(c_{A\cap B}))^x\big)$ 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.

In practice, it is sufficient to evaluate Equation 3 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. Consequently, we bound x by a small number (e.g., 10) in Equation 3. To give an overview of the behaviour of $Pr[P(A) \cap P(B) \neq \varnothing]$, Figure 7 visualizes Equation 3 for different s_A and s_B (x is bounded as described above). 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 3 instead of Equation 1.

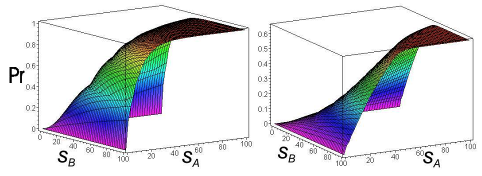

Figure 7: Probability $Pr[P(A) \cap P(B) \neq \varnothing]$ for a fixed s (=300). Left: $LB(c_{A\cap B}) = 0.5$, right: $LB(c_{A\cap B}) = 0.1$.

3.6.3 Probability of intersection in a cell

We have already shown how to estimate $LB(c_{A\cap B})$ in a previous paper [11]. Therefore, we only give the solution for estimating it. Let d_A and d_B denote the depth of node A and B in their respective BV hierarchies and d_{max_A}, d_{max_B} the maximum depths of the hierarchies. Then,

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

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 [22]. Our measurements show that the AABB tree performs better than the octree by a factor > 3. Therefore, all the following benchmarks were performed using our ADB-tree based on AABBs.

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. According to our experiments, 8^3 cells are optimal. The number of possible collision cells can be computed by our algorithms shown in Section 3.4. For our models, the exact algorithm needs about 2 minutes on average for the computation of possible collision cells for one complete BV hierarchy, while the approximative algorithm needs only less than 2 seconds, for our most complex model of 200,000 polygons. All our measurements were performed using the exact algorithm, but the approximative one reduces the quality of the collision detection only by about 0.2% points on average.

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.

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 1 in the Color Plates Appendix).

Benchmarking is performed by the procedure proposed in Zachmann [22], which computes average collision detection times for a range of distances between two identical objects.

4.1 Distribution of Possible Collision Cells

One premise of our average-case approach is the assumption that the probability of being a possible collision cell is evenly distributed among all cells of the partitioning (see Section 3.1). Here, we give some empirical results suggesting that in practical cases this assumption is actually valid.

Given an ADB-tree, we can identify corresponding cells of all nodes by a number $x \in \{1, \ldots, 512\}$. Thus, for all x we can count over all nodes how often that cell is a possible collision cell throughout the tree (this number $db(x) \leq n$).

Figure 8 shows the distribution of the possible collision cells for different models with varying complexities.

Obviously, our assumption seems to be met by almost all objects occurring in practice. An exception might be the door-lock model with 207 290 polygons, where $\max\{db(x)\}$ and $\min\{db(x)\}$ are about 30% larger and smaller than the average.

4.2 Time and Quality versus Complexity

Each plot in Figure 9 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.

Figure 10 shows the error rates corresponding to the timings in Figure 9. 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 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 quality 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 11 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 11 (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 [10]. 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

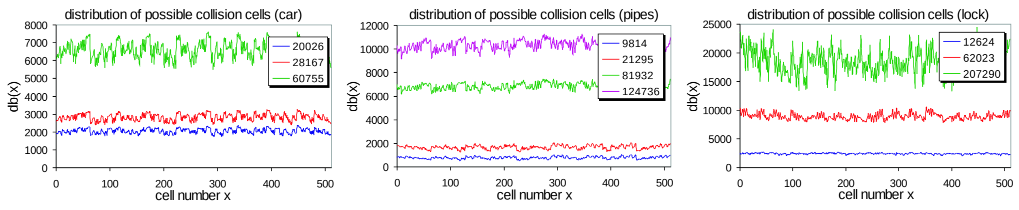

Figure 8: Histogram of the number of times, $db(x)$, cell x occurred as a possible collision cell in the ADB-tree. The number in parentheses in the legend gives the number of polygons.

in accuracy by 0.2% points justifies a decrease in speed by a factor ≈ 2.3.

4.4 Performance Comparison

A runtime comparison between our approach and a DOP tree algorithm can be found in Figure 9. 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 11).

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 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.

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. 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.

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

References

[1] 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. 1

[2] 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. 2

[3] S. Ar, G. Montag, and A. Tal. Deferred, self-organizing BSP trees. In *Eurographics*, pages 269–278, 2002. 2

[4] 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. 1

[5] J. Dingliana and C. O'Sullivan. Graceful degradation of collision handling in physically based animation. *Proc. Eurographics 2000*, 19(3):239–247, 2000. 2

[6] 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. 1

[7] 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. 1

[8] P. M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transactions on Graphics*, 15(3):179–210, 1996. 1, 2

[0] 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.

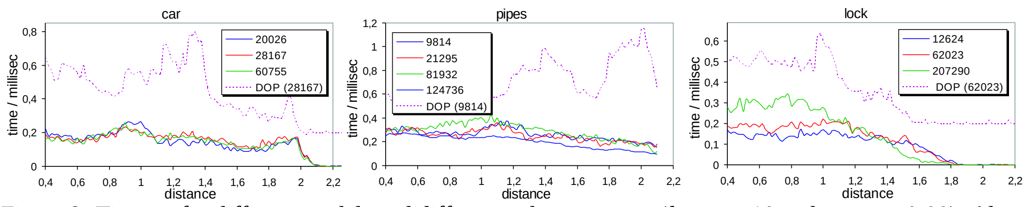

Figure 9: Timings for different models and different polygon counts ($k_{min} = 10$ and $p_{min} = 0.99$). Also, a runtime comparison to a DOP tree is shown (see Section 4.4).

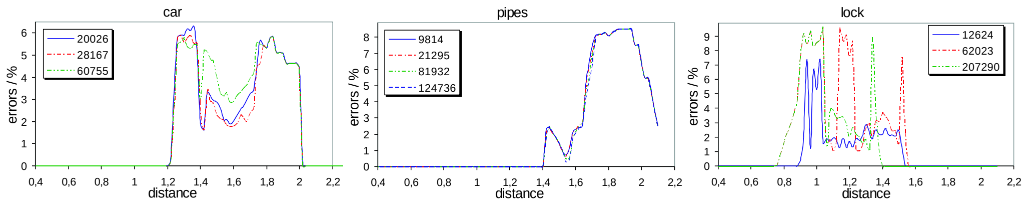

Figure 10: Error rates corresponding to the timings in Figure 9.

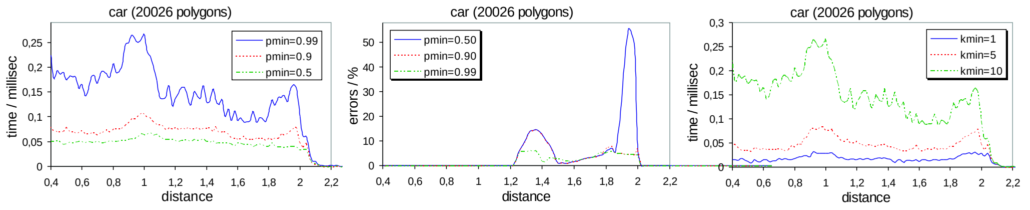

Figure 11: Runtime and error comparisons for the car body with 20,000 polygons. Left, center: timings and error rates for different p_{min} ($k_{min} = 10$); right: Timings for different k_{min} ($p_{min} = 0.99$).

[9] 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. 2

[10] 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`. 4, 6

[11] J. Klein and G. Zachmann. Time-critical collision detection using an average-case approach. In *Proc. VRST*, Osaka, Japan, 2003. 4, 5

[12] 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. 1

[13] T. Larsson and T. Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics*, pages 325–333, 2001. short presentation. 1

[14] 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. 1

[15] 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. 1

[16] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Proc. Eurographics 1995*, 14(2):105–116, 1995. 1

[17] M. Shinya and M.-C. Forgue. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, 2(4):132–134, 1991. 1

[18] S. Uno and M. Slater. The sensitivity of presence to collision response. In *Proc. VRAIS*, page 95, Albuquerque, New Mexico, 1997. 1

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

[20] 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. 2

[21] G. Zachmann. Rapid collision detection by dynamically aligned DOP-trees. In *Proc. VRAIS 1998*, pages 90–97, Atlanta, Georgia, 1998. 1

[22] G. Zachmann. Minimal hierarchical collision detection. In *Proc. VRST 2002*, pages 121–128, Hong Kong, China, 2002. 1, 6