Rapid Collision Detection by Dynamically Aligned DOP-Trees

Gabriel Zachmann
Fraunhofer Institute for Computer Graphics
Rundeturmstraße 6, 64283 Darmstadt, Germany
email: zach@igd.fhg.de

Abstract

Based on a general hierarchical data structure, we present a fast algorithm for exact collision detection of arbitrary polygonal rigid objects. Objects consisting of hundreds of thousands of polygons can be checked for collision at interactive rates.

The pre-computed hierarchy is a tree of discrete oriented polytopes (DOPs). An efficient way of re-aligning DOPs during traversal of such trees allows to use simple interval tests for determining overlap between DOPs. The data structure is very efficient in terms of memory and construction time.

Extensive experiments with synthetic and real-world CAD data have been carried out to analyze the performance and memory usage of the data structure. A comparison with OBB-trees indicates that DOP-trees as efficient in terms of collision query time, and more efficient in memory usage and construction time.

Keywords: Interference detection, virtual prototyping, hierarchical bounding volumes, physically-based modeling, shape approximation.

1 Introduction

Real-time collision detection of polygonal objects undergoing rigid motion is of critical importance in many interactive virtual environments. In particular, simulation algorithms, utilized in virtual reality systems to enhance object behavior and properties, need several collision queries per frame. It is a fundamental problem of dynamic simulation of rigid bodies and simulation of natural interaction with objects. For example, in virtual prototyping, parts should be rigid and slide along each other. A very demanding simulation is force-feedback which needs at least 1000 collision queries per second. It is very important for a VR system to be able to do all simulations at interactive frame rates. Otherwise, the feeling of immersion, the "believability" of the virtual environment, and even the usability of the VR system will be impaired severely.

For real-time simulations, the collision detection must be fast enough under all circumstances to allow several iterations per frame with objects consisting of 5,000 to 50,000 polygons at least. The algorithm must not make any assumption about the input, such as convexity, topology, or manifolded-ness, because polygonal geometry, converted from CAD data, is usually not "well-formed": it may have cracks, identical or degenerate polygons, etc. (hence they have been named "polygon soups"). Also, it cannot make any assumptions or estimations about the position of moving objects in the future, especially when they are being moved by the user. Most collision response algorithms need at least one point of intersection (a witness) in order to take reasonable steps.

Fast collision detection of moving non-convex polygonal objects seems to be a "hard" problem because of its immanent "all-pairs weakness" [13]. Several bounding volume (BV) hierarchies have been developed to overcome this problem in order to find quickly a pair of intersecting polygons, or to reject pairs of polygons which cannot intersect[1].

The performance of any collision detection based on hierarchical bounding volumes depends on two conflicting constraints: (1) the tightness of the BVs, which will influence the number of BV tests, and (2) the simplicity of the BVs, which determines the efficiency of an overlap test of BVs. Our algorithm is more biased towards optimizing (2). However, the bias can be shifted to (1) by making them arbitrarily tight.

Our approach is based on the BVs introduced by [15]. They have been named *discrete orientation polytopes* (DOPs) by [12]. For each object, we build a hierarchy of DOPs (DOP-Tree), each leave of which encloses a single polygon of the object. In order to quickly determine ether two DOPs overlap, one of them has to be "re-aligned". This can be done "on-the-fly" during traversal of the two DOP-Trees by a simple affine transformation of the "non-aligned" DOPs.

[1]We conjecture that for any algorithm there are two objects and two positions for them such that the algorithm performs no better than $\Omega(n^2)$.

The next section provides a quick overview of other hierarchical collision detection methods. Section 3 describes the general framework which most hierarchical collision detection methods are based on. In Section 4 we explain the application of hierarchical DOPs to collision detection including the construction of DOP-trees. Section 5 presents results of the effect of the number of orientations on collision detection time. Also, our DOP-Trees are compared to OBB-Trees. Finally, Section 6 draws some conclusions and describes future directions for research on hierarchical collision detection.

2 Related work

Interference and collision detection problem have been extensively studied in the literature. Computational geometry first focused on the *construction* of the intersection of two polyhedra [18, 17] and later on the *detection problem* [5, 20]. The algorithms are very efficient in the asymptotical worst-case. However, most of them are restricted to static environments and many of them have not been implemented.

Configuration space has been used to detect collisions for path-planning in robotics (see [6], for example). For collision avoidance, Shaffer et al. [4] have used an octree to approximate objects.

Good results have been achieved for convex polyhedra [16, 3, 8]. Almost all approaches for general, non-convex objects utilize some sort of BV hierarchy. A few non-hierarchical approaches have been taken as well. Garcia-Alonso et al. [7] partition the set of polygons of an object by a uniform grid.

Sphere trees were developed by [13, 14, 19]. Gottschalk et al. [11] developed a bounding box hierarchy based on oriented boxes (OBBs). Its box overlap test is much more expensive than for DOPs. Aligned Box-Trees have been presented in [21]. Barequet et al. [1] presented some theoretical results on a class of BV hierarchies called BOX-TREE, too.

DOP-trees have been used for collision detection before by [12]. However, they seem to use more general DOPs and do hill-climbing to compute the axis-aligned DOPs from the “tumbled” ones, which is probably less efficient than our method.

3 The hierarchical collision detection scheme

The goal of any hierarchical BV scheme used for collision detection is to discard quickly any pairs of polygons which cannot intersect. Each node in the tree is associated with a subset of the primitives of the object, together with a BV that encloses this subset with a smallest containing instance of some specified class of shapes. Given two objects and the roots of their associated BV trees, a simultaneous traversal of the two trees recursively checks all pairs of their children BVs for overlap. If such a pair does not overlap, then the polygons enclosed by them cannot intersect.

The following pseudo-code outlines the basic scheme of collision detection algorithms based on a BV hierarchy:

```
 Simultaneous traversal of BV trees
a = bounding volume of A's tree,
b = bounding volume of B's tree
a[i], b[i] children of a and b, resp.

traverse(a,b):
a or b is empty → return
b leaf →
  a leaf →
    check primitives enclosed by a and b
    return
  a not leaf →
    forall i:
      a[i],b overlap → traverse(a[i],b)
b not leaf →
  a leaf →
    forall i:
      a,b[i] overlap → traverse(a,b[i])
  a not leaf →
    forall i:  forall j:
      a[i],b[j] overlap→ traverse(a[i],b[j])
```

Figure 1 visualizes the set of polygons considered in the worst-case for exact intersection calculations. The picture has been obtained with our DOP-Tree data structure, but all other hierarchical BV schemes would produce a similar image.

4 Discrete orientation polytopes

Discrete orientation polytopes (DOPs) are convex polytopes whose faces can have only normals which come from a fixed small set of k orientations (k-DOPs). Probably the fastest overlap check for axis-aligned boxes is the well-known interval test. In order to be able to apply such a test to DOPs, we restrict the set of orientations further such that for each orientation of the set there is also an anti-parallel one; the two corresponding faces form what is commonly known as a *slab*. Thus, this special kind of k-DOPs can be viewed as a generalization of axis-aligned boxes.

Being the intersection of k half-spaces

$$H_i : B_i x - d_i \leq 0,$$

a k-DOP can be represented by the point $d \in \mathbb{R}^k$, where B_i are the k fixed orientations. We will call d the *plane*

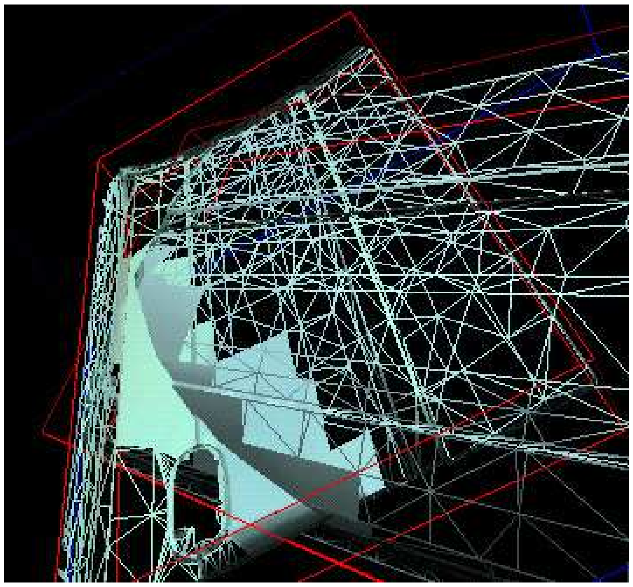

Figure 1: All polygons which are considered for intersection in the worst-case are rendered solid. The rejection of all other polygons is based on the hierarchical BV data structure.

offsets for that DOP. At each node of our hierarchical data structure we need to store only k floating point numbers (since the orientations are the same for all DOPs) plus 2 pointers (assuming binary trees).

We would like to remark that, except for the just mentioned interval overlap test of DOPs, all other parts of our algorithms apply to the case of general DOPs as well.

4.1 Aligning DOPs

Given two DOP-Trees A and B, the basic step of the simultaneous traversal is an overlap test of two nodes. In order to apply the simple and very fast interval overlap test to DOP-Trees, they must be given in the same space. However, at least one of the associated objects has been transformed by a rigid motion. By choosing A's object space (w.l.o.g.), we need to re-align only B's nodes as we encounter them during traversal. Aligning a node (a "non-aligned DOP") is done by enclosing it in another (aligned) DOP. We will show that this can be done by a simple affine transformation of the node's plane offsets.

Assume we are given a (non-aligned) DOP D of B's DOP-Tree, which is represented by d. Assume also, that the object associated with B has been transformed by a rotation M and a translation o, with respect to A's reference frame. Then D is the intersection of k half-spaces

$$h_i : b_i x - d_i + o_i \leq 0,$$

where $b_i = B_i M^{-1}$ (see Figure 3).

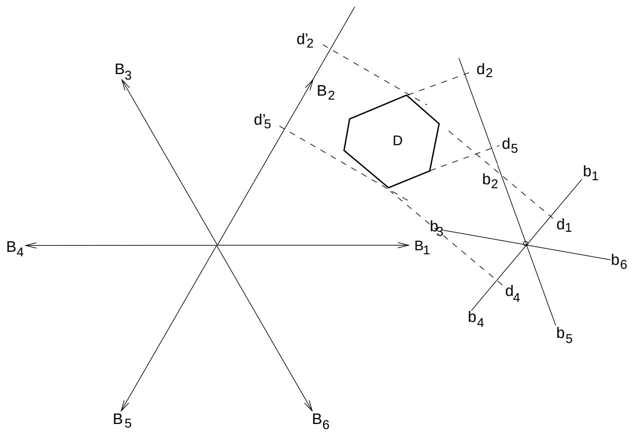

Figure 3: A rotated DOP can be enclosed by an aligned one by computing new plane offsets d'_i. Each d'_i can be computed by an affine combination of 3 $d_{j^i_l}, 1 \leq l \leq 3$ (2 in 2-space). The correspondence j^i_l depends only on the affine transformation of the associated object and the fixed orientations B_i.

Now suppose we want to compute the d'_i of the enclosing DOP D' of D. There is (at least) one extremal vertex P_i of the convex hull of D with respect to B_i. This vertex is the intersection of 3 (or more) half-spaces $h_{j^i_l}, 1 \leq l \leq 3$. It is easy to see that

$$d'_i = B_i \begin{pmatrix} b_{j^i_1} \\ b_{j^i_2} \\ b_{j^i_3} \end{pmatrix}^{-1} \begin{pmatrix} d_{j^i_1} \\ d_{j^i_2} \\ d_{j^i_3} \end{pmatrix} + B_i o$$

The correspondence established by j^i_l is the same for all (non-aligned) DOPs of the whole tree, if the following condition is met: DOPs must not possess any *completely redundant* half-spaces, i.e., all planes must be supported by at least one vertex of the convex hull of the DOP. (We do allow *almost redundant* half-spaces, i.e., planes which are supported by only a single vertex of the convex hull.) Fortunately, this condition is trivially met when constructing the DOP-Tree.

Thus, enclosing a non-aligned DOP by an aligned one takes only $3k$ multiplications and $3k$ additions.

The correspondence j^i_l is established by two steps: First, we compute the vertices of the convex hull of a generic DOP (all $d_i = 1$). Each vertex is supported by three planes (or more), the orientations of which we store with that vertex in an intermediate correspondence.

This has to be done only once for a given set of orientations. We can choose any DOP to establish this (intermediate) correspondence. However, we must make sure that all planes do support a non-degenerate face.

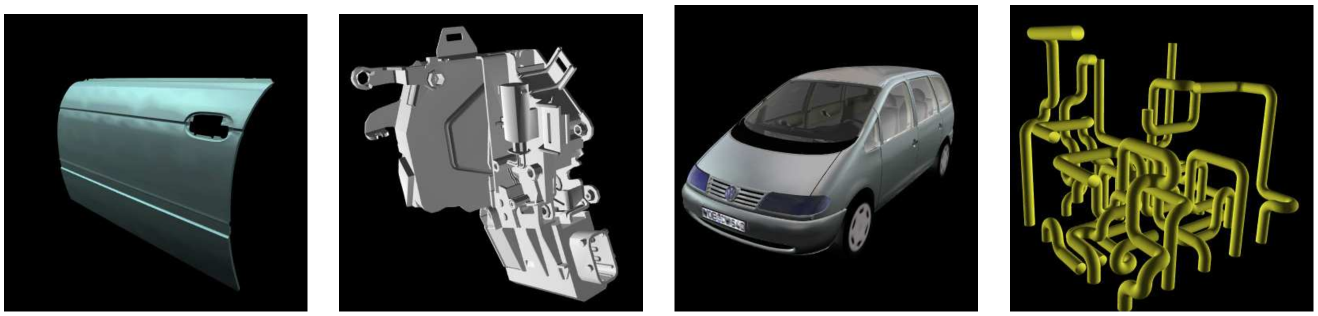

Figure 2: The suite of test objects. They are (left to right): the skin of a car door ($\approx$ 3,000 polygons; data courtesy of BMW), the lock of a car door ($\approx$ 20,000 polygons; data courtesy of BMW), skin and seats of a car($\approx$ 60,000 polygons; data courtesy of VW), a section of pipes($\approx$ 120,000 polygons).

Second, at the beginning of each DOP-tree traversal, we transform the vertices of the generic DOP by the object's rotation. Then, we use these to establish the final correspondence j^i_l.

4.2 Building DOP-Trees

Any hierarchical algorithm can be only as good as the hierarchical data structures. "Good" hierarchical data structures for ray-tracing are characterized by a low stabbing number [1]. "Good" BV trees for collision detection are characterized by a low number of primitive intersection tests.

It is not clear yet, if there is a single measure which should be optimized during the construction of a BV hierarchy in order to achieve an optimal tree for collision detection. Obviously, the following criteria should guide the construction algorithm:

- The total volume of all BVs should be small [1].
- The tree should be balanced in terms of polygon counts.
- The volume of overlap of two siblings should be small.

Our construction of DOP-Trees can be contrasted to "insertion" methods [9, 2] and "bottom-up" methods [1]. We proceed in a "top-down" fashion starting with the complete set of polygons, splitting the set at each node.

For our application it is very important, that the construction process be fast, so that it can be done at load time. Therefore, we make use of several heuristics and estimations which try to emulate the criteria listed above.

The input to the construction algorithm is a set $\mathcal{F}$ of k-DOPs (each of them encloses one of the polygons of the object) and a set $\mathcal{C}$ of points which are the barycenters of the polygons.

The algorithm starts by finding $c_i, c_j \in \mathcal{C}$ with almost maximal distance[2]. Then it determines that orientation which is "most parallel" to $\overline{c_i c_j}$. Now we sort $\mathcal{C}$ along that orientation, which induces a sorting on $\mathcal{F}$.

After these preliminaries (which are done for each step of the recursion), we can split $\mathcal{F}$ in two parts $\mathcal{F}_1$ and $\mathcal{F}_2$. We start with $\mathcal{F}_1 = f_i$, $\mathcal{F}_2 = f_j$, where f_i, f_j are associated to c_i, c_j, resp. Then, we consider all other $f \in \mathcal{F}$ in turn and assign them to $\mathcal{F}_1$ or $\mathcal{F}_2$, whichever BV increases less. If both BVs of $\mathcal{F}_1$ and $\mathcal{F}_2$ would increase by the same amount (in particular, if they would not increase at all), then f is added to the set which has fewer polygons so far.

This algorithm seems to produce very good trees. In particular, they are fairly well balanced. Therefore, the average depth is almost the same for all sets of orientations (which has been verified by our experiments). The following table shows a histogram of the depth of the leaves of the 6-DOP-Tree for two objects. The sphere has approximately 20,000 polygons, while the car door has approximately 3,300 polygons:

depth	10	11	12	13	14	15	16
sphere	0	0	1	375	8853	13233	642
car door	92	740	1639	659	162	42	26

Other heuristics have been implemented and tested, but the one described above has produced the best results.

4.3 Geometric robustness and accuracy

The construction algorithm is geometrically robust and can be applied to all unstructured models. No adjacency information is required. There are no connectivity restrictions and the faces can be degenerate (a line segment or a point), which happens frequently in CAD data.

The overlap test is very robust, since it involves only multiplications, additions, and comparisons. A small ϵ

[2]We compute only a near-optimal pair by a simple $O(n)$ heuristic.

margin guards against arithmetic round-off errors. Actually, that guard can be applied to the DOPs while the DOP-Tree is being constructed (by inflating them a little), so during traversal, no ϵ additions/subtraction need to be done. No error accumulation can occur during traversal of the tree.

5 Benchmarks and results

It is extremely difficult to evaluate and compare collision detection algorithms, because in general they are very sensitive to specific scenarios, i.e., the relative size of the two objects, the relative position to each other, the distance, etc. We propose a simple benchmark program which (hopefully) eliminates these effects. It has been kept very simple so that other researchers can easily reproduce our results and compare their algorithms to ours[3].

Our test scenario involves two identical objects which are positioned at a certain distance from each other. The distance is computed between the centers of the bounding boxes of the two objects. Then one of them performs a full revolution around the z-axis (which is pointing towards the viewer in Figure 2) in a fixed, large number (here 2000) of small steps. With each step a collision query is done, and the average collision detection time for that distance is computed. This procedure is done for different distances, which yields graphs as shown in Figures 4, 5.

We have carried out extensive experiments using this benchmark procedure with different objects, both synthetic and real-world CAD data. Synthetic objects are spheres, tori, cylinders in various resolutions from 1,000 polygons through 100,000 polygons. The CAD data are depicted in Figure 2. The complexity ranges from $\approx$ 3,000 to $\approx$ 120,000 polygons. All objects are scaled to fit in a cube of size 2^3.

All test have been done on a SGI R10000 (194 MHz).

5.1 The optimal number of orientations

Obviously, there are two contradictory effects when the number of orientations of DOP-Trees is increased: on the one hand, they can better approximate the convex hull of the set of polygons enclosed by them; on the other hand, an overlap test between them is more expensive.

Three different sets with 6, 8, and 14 orientations have been tested: the faces of 6-DOPs have the same normals as a cube, 8-DOPs have normals of an octahedron, and 14-DOPs have normals of the union of the former two.

The results are depicted in Figures 4 and 5. From the figures it becomes clear why comparing collision detection

[3]The source code of the "main loop" and the synthetic objects can be ftp'ed from `http://www.igd.fhg.de/~zach/-coldet/benchmark.html`. The CAD objects can be obtained via the author (for scientific purposes only).

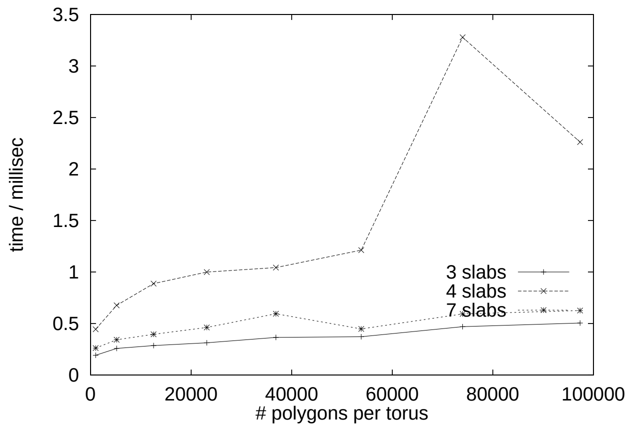

Figure 6: As complexity increases the average collision detection time increases only slightly for DOP-Trees.

		avg. time / millisec k-DOP-Tree		
	#polygons	$k = 6$	$k = 8$	$k = 14$
door	3391	0.27	1.81	0.38
lock	20898	3.02	24.6	3.10
car	60755	0.42	5.25	0.59
pipes	124736	1.69	232	23.7

Table 1: Comparison of 6-, 8-, and 14-DOPs. There is no significant advantage of 14-DOPs over 6-DOPs.

algorithms is difficult: Depending on the situation and the particular objects chosen for the tests, the collision query times can vary by an order of magnitude.

We feel that a fair measure for the overall performance is the mean collision detection time averaged over all distances and all rotations. The average time for tori is shown in Figure 6 while Table 1 summarizes the results for all CAD objects.

There doesn't seem to be a significant and general performance benefit from using 14 orientations compared to 6 orientations. It is not clear to us why 8 orientations perform so bad.

We also compared the memory requirements and construction time among DOP-Trees using 6, 8, or 14 orientations. As expected, construction time increases slightly as the number of orientations increases (see Table 2). Also, the amount of memory required increases slightly (see Table 3). This is due to the fact that the depth of trees doesn't change with different sets of orientations.

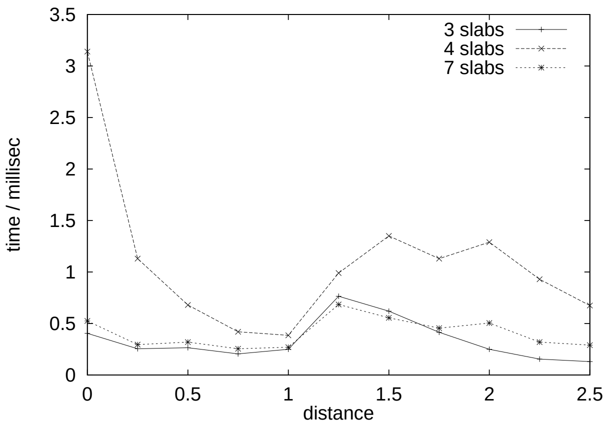

Figure 4: Two tori (53824 polygons each), one of them is rotating. At each distance, 2000 rotations and collision queries have been performed. The time shown is the average collision detection time.

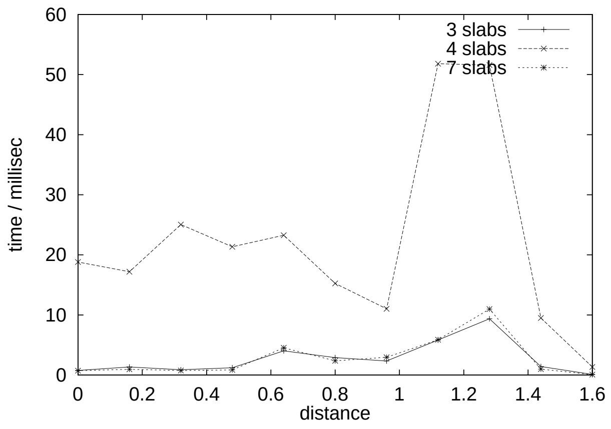

Figure 5: The locks (20898 polygons each) are different in that most of the polygons are in the interior, b/c it has many parts on the inside. This might explain the "inversion" of the shape of the curve.

		time / sec # orientations		
	#polygons	6	8	14
door	3391	0.14	0.17	0.19
lock	20898	1.3	1.4	1.8
car	60755	4.2	4.7	6.4
pipes	124736	9.4	10.3	13.8

Table 2: Construction time of the BV hierarchy for various CAD and synthetic objects. The time increases by about 10 percent when 14-DOPs are used instead of 6-DOPs.

		MB # orientations		
	#polygons	6	8	14
door	3391	0.1	0.1	0.1
lock	20898	1	1	2
car	60755	5	6	9
pipes	124736	11	13	19

Table 3: The amount of memory required by a DOP-Tree with various orientation sets. 14-DOP-Trees need about 10 percent more memory than 6-DOP-Trees.

		avg. time / millisec		
	#pgons	6-DOP-Tree	OBB-Tree	speedup
door	3391	0.27	0.34	1.2
lock	20898	3.0	2.2	0.7
car	60755	0.42	1.1	2.6
pipes	124736	1.67	1.04	0.6
tori	12544	0.28	0.33	1.2
spheres	22952	0.48	0.56	1.1
tori	73984	0.47	0.43	0.9
spheres	97032	1.02	0.73	0.7

Table 4: Summary comparing the performance of DOP-Trees and OBB-Trees.

5.2 Comparison of DOP-Trees and OBB-Trees

Since there is no significant advantage in 14-DOPs over 6-DOPs, we used 6-DOPs to compare DOP-Trees with OBB-Trees. For OBB-Trees, the Rapid implementation [10] has been used.

Figures 7 and 8 show the results of the benchmark for DOP-Trees and OBB-Trees. Table 4 summarizes the results for several other objects. All timings include the time to transform the (necessary) vertices and normals.

Table 5 shows the amount of memory required by DOP-Trees and OBB-Trees. For both, memory usage depends linearly on the number of polygons. At each node DOP-Trees store only 6 floats whereas an OBB-node stores (at least) 18 floats. The table indicates that DOP-Trees need about 4-8 times less memory. (For the memory comparison, we have changed all `double`s to `float`s in the Rapid

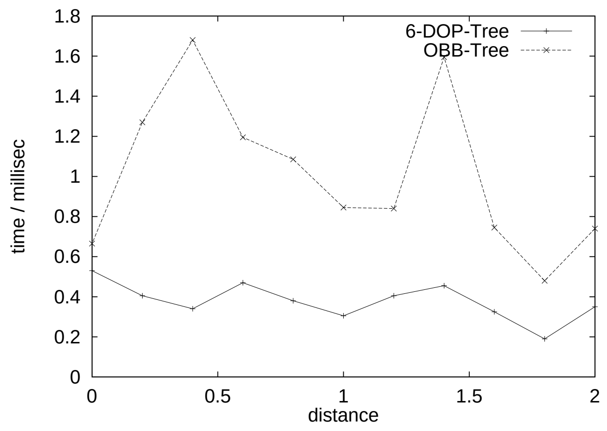

Figure 7: Comparison of DOP-Trees and OBB-Trees for the car model.

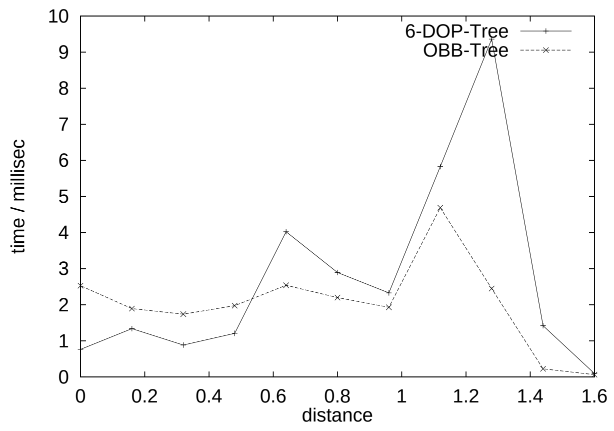

Figure 8: Comparison for the door lock.

		MB	
	#polygons	6-DOP	OBB
door	3391	0.1	1
lock	20898	1	8
car	60755	5	21
pipes	124736	11	44
tori	12544	0.1	9
spheres	22952	1	17
tori	73984	6	60
spheres	97032	9	73

Table 5: A comparison of memory requirements indicates that DOP-Trees need about 4-8 times less memory then OBB-Trees.

		time / sec	
	#polygons	6-DOP-Tree	OBB-Tree
door	3391	0.1	0.4
lock	20898	1.3	4.5
car	60755	4.2	10.5
pipes	124736	9.4	18.4
tori	12544	0.6	2.4
spheres	22952	1.3	5.4
tori	73984	4.8	16.1
spheres	97032	6.7	25.6

Table 6: Comparison of the construction time of the BV hierarchy for various CAD and synthetic objects. The construction of DOP-Trees is about 2-4 times faster.

code [10]. Of course, the memory required by the list of vertices and polygons is counted, too.)

For virtual reality applications to be utilized at floor shops and at design evaluation sessions it is important that startup time be as short as possible. Since it is not desirable to store an abundance of auxiliary data structures with a virtual environment on disk, construction time of data structures needed for collision detection should be minimal.

Table 6 summarizes the comparison of the construction time of the data structures for DOP- and OBB-Trees.

6 Conclusion and future work

We have developed and implemented a hierarchical data structure for fast and exact collision detection of polygonal models with very large polygon counts. The algorithm is general-purpose and makes no assumption about the input model. It utilizes discrete oriented polytopes (DOPs) for bounding volumes. We show that “re-aligning” rotated DOPs can be done by a simple affine transformation of their plane offset representations. Also, we have presented a new method of constructing DOP-Trees.

We have carried out extensive and thorough experiments to analyze the impact of the orientation count on performance, construction time and memory usage. Our experiments indicate that 6-DOPs (i.e., boxes) are as good as 14-DOPs.

Our algorithm using DOP-Trees is about as fast as Rapid using OBB-Trees. Furthermore, DOP-Trees are significantly more memory efficient and can be constructed much faster than OBB-Trees.

There are several issues of hierarchical collision detection which still need to be investigated. One of them is the characterization and construction of optimal BV hierarchies for collision detection. While our method of constructing DOP-Trees performs quite well, it would be interesting to develop precise criteria for the construction of optimal BV trees in the sense of minimum average collision detection time. Also, it would be interesting to increase the orientation count even further to verify our hypothesis stated above.

So far, we have applied our hierarchical data structures only to the problem of collision detection. However, we feel that it has great potential for other applications, such as ray-tracing, interference detection of curved surfaces, and point-location and range queries.

Finally, the algorithm is a good candidate for implementation in hardware, since the flow of control is quite simple, and the overlap test involves only simple linear arithmetic.

References

[1] G. Barequet, B. Chazelle, L. J. Guibas, J. S. B. Mitchell, and A. Tal. BOXTREE: A hierarchical representation for surfaces in 3D. *Computer Graphics Forum*, 15(3):387–396, Sept. 1996.

[2] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R*-tree: An efficient and robust access method for points and rectangles. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 322–331, 1990.

[3] K. Chung. Quick collision detection of polytopes in virtual environments. In M. Green, editor, *Proc. of the ACM Symposium on Virtual Reality Software and Technology*, pages 125–131, 1996.

[4] G. M. H. Clifford A. Shaffer. A real-time robot arm collision avoidance system. *IEEE Transactions on Robotics and Automation*, 8(2), April 1992.

[5] D. P. Dobkin and D. G. Kirkpatrick. A linear algorithm for determining the separation of convex polyhedra. *J. Algorithms*, 6:381–392, 1985.

[6] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[7] A. García-Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. *IEEE Computer Graphics and Applications*, pages 36–43, May 1994.

[8] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, 4(2):193–203, 1988.

[9] J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.

[10] S. Gottschalk. Rapid library. http://www.cs.unc.edu/ geom/OBB/-OBBT.html, Vers. 2.01.

[11] S. Gottschalk, M. Lin, and D. Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 171–180. ACM SIGGRAPH, Addison Wesley, Aug. 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[12] M. Held, J. T. Klosowski, and J. S. Mitchell. Real-time collision detection for motion simulation within complex environments. In *Siggraph 1996 Technical Sketches, Visual Proceedings*, page 151, New Orleans, Aug. 1996.

[13] P. M. Hubbard. Interactive collision detection. In *IEEE Symposium on Research Frontiers in VR, San José, California*, pages 24–31, October 25–26 1993.

[14] P. M. Hubbard. Collision detection for interactive graphics applications. *IEEE Transactions on Visualization and Computer Graphics*, 1(3):218–230, Sept. 1995. ISSN 1077-2626.

[15] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In D. C. Evans and R. J. Athay, editors, *Computer Graphics (SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, Aug. 1986.

[16] M. C. Lin and D. Manocha. *Efficient Contact Determination Between Geometric Models*. PhD dissertation, University of California, University of North Carolina Chapel Hill, URL: ftp://ftp.cs.unc.edu/pub/techreports/94-024.ps.Z, 1991(?).

[17] K. Mehlhorn and K. Simon. Intersecting two polyhedra one of which is convex. In L. Budach, editor, *Proc. Found. Comput. Theory*, volume 199 of *Lecture Notes Comput. Sci.*, pages 534–542. Springer-Verlag, 1985.

[18] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoret. Comput. Sci.*, 7:217–236, 1978.

[19] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, 14(2):105–116, June 1995.

[20] M. Reichling. On the detection of a common intersection of k convex polyhedra. In *Computational Geometry and its Applications*, volume 333 of *Lecture Notes Comput. Sci.*, pages 180–186. Springer-Verlag, 1988.

[21] G. Zachmann. The BoxTree: Enabling real-time and exact collision detection of arbitrary polyhedra. In *Informal Proc. First Workshop on Simulation and Interaction in Virtual Environments, SIVE 95*, University of Iowa, Iowa City, July 1995. The OX Association for Computing Machinery.