CDFast: an Algorithm Combining Different Bounding Volume Strategies for Real Time Collision Detection

Andrea SANNA† Mattia MILANI‡

**Dipartimento di Automatica e Informatica,
Politecnico di Torino, corso Duca degli Abruzzi 24,
I-10129 Torino (Italy)**

andrea.sanna@polito.it† mattia@wolfman.it‡

ABSTRACT

Collision detection has a great importance in a large spectrum of disciplines, such as virtual reality, computer graphics, simulation of physical systems, robotics, solid modelling, and so on. In particular, some applications need to determine in real time if a collision occurs in order to guarantee an interactive behavior of the system.
This paper presents an algorithm that combines different bounding strategies in order both to speed up and to reduce interference tests. A recursive space subdivision is performed by uniform grids to bound static objects within the environment; on the other hand, spheretrees, at different levels of detail, are used to encapsulate moving objects.
The proposed methodology is compared with an approach that uses discrete orientation polytopes (k-dops). K-dops proved to be more efficient than other bounding volume strategies, such as oriented and axis-aligned bounding boxes. A theoretical evaluation of the algorithm complexity as well as experimental results are provided.

Keywords: collision detection, interference tests, real time interaction, computer animation.

1. INTRODUCTION

Many interactive animation systems, such as video games, virtual reality applications, real time simulations involve a large number of static and dynamic entities (objects). Within an environment where some entities can move, one or more collisions, between a static and a moving object or between two moving objects, can occur.
The task of detecting collisions is a challenge when a real time response is needed. The term "real time" assumes different meanings but, in this context, it denotes a graphics application able to produce new frames at least 10 times per second. This means there are 100 ms or few for the whole rendering process of a single frame included the time necessary for collision tests.
Despite the fact that the collision detection problem has been deeply investigated and several works are known in literature ([18][22]), open problems still exist. Many collision detection algorithms require a high computational cost and are not suitable for real time applications; moreover, some techniques are tailored for specific applications and/or kind of objects and their application field is limited.
Many algorithms ("hybrid") divide the collision detection problem in two steps [19]. In the first step a set of approximate tests are performed in order to identify pairs of interferring/colliding objects, while during the second step more accurate tests report all pairs of intersecting geometric primitives (e.g. triangles). These two steps are often called *broad phase* and *narrow phase* [10]. In particular, the first step aims to reduce the number of interference tests, and one of the most popular approach to tackle this issue is building bounding volume hierarchies surrounding objects. Simple shape volumes, such as spheres, boxes, and cylinders are often chosen to encapsulate objects; in this way, intersection tests between bounding entities can be performed very quickly. The efficiency of the algorithm also depends on two other issues: the tightness of the encapsulating hierarchy ("tighter" approximations allow to avoid more unnecessary tests) and the cost of performing rotations and translations on the bounding volume hierarchy (e.g. spheres minimize costs due to rotation/translation operations).
The proposed method addresses the issue related to the broad phase by combining different kind of bounding entities for static (elements belonging to the environment) and dynamic (elements moving within the environment) objects. A recursive uniform grid is employed to divide the environment in cells; static objects affecting each cell are then determined. Moving objects are bounded by spheretrees and every object can be associated to a set of spheretrees computed at different levels of detail. This approach has three main advantages: the tightness of bounding entities is dynamically arranged according to the environment topology, overlapping tests between bounding entities are extremely fast as they involve very simple shapes, and the costs of rotations/translations for moving objects are minimized as the bounding volumes involved are spheres.
In order to evaluate the proposed algorithm, a comparison with [20] is provided. In [20] discrete orientation polytopes (k-dops) are used to bound objects and this approach proved to be more efficient than other strategies, such as axis-aligned bounding boxes (AABBs) [28] and oriented bounding boxes (OBBs) [13]. Moreover, a theoretical evaluation of the algorithm complexity is provided.
In Section 2 the main approaches to collision detection are reviewed and k-dops are analyzed in detail. Section 3 presents the basic idea behind the proposed methodology and describes the goals of this work. Section 4 shows in detail the algorithm and Section 5 reports the theoretical and experimental results.

2. BACKGROUND

This section reviews the main approaches to collision detection. The collision detection problem can be coarsely summarized as follows: given a set of objects (motion laws may be also known for dynamic objects) determine if any pair will collide during a considered time span.

Techniques to collision detection

Collision detection algorithms can be classified either according to the geometric object model used [22] or by a systematic subdivision of the solving strategies [18]; this section follows the latter organization. Four techniques are known in the literature:

1. space-temporal intersection;
2. swept volume interference;
3. multiple interference detection;
4. trajectory parameterization.

Space-temporal intersection algorithms are based on the extrusion operation [2]. Given an object *O* and a location function *Λ* (*Λ* takes a time *t* and returns a transformation *Λ(t)* that denotes how to move the object *O* in its position at the time *t*) the point set occupied by *O* is:

$$\{x \mid (\exists y) y \in O \quad and \quad x = \Lambda(t)(y)\}$$

usually written as *Λ(t)(O)*. The above formula allows to formally define the extrusion operation:

$$Ex(\Lambda, O) = \{(x,t) \mid x \in \Lambda(t)(O)\}$$

Two objects *A* and *B* collide if and only if their extrusions intersect, that is:

$$Ex(\Lambda_A, A) \cap Ex(\Lambda_B, B) \neq \phi$$

As the extrusion operator is distributive with respect union, intersection, and set difference operations, it is well suited for Constructive Solid Geometry (CSG) model representations. The previous formula indicates a criteria to determine if two objects collide, but it does not specify how extrusions have to be constructed. Unfortunately, the extrusion construction involves high computational times and this often leads to consider linear approximations of trajectories.

Swept volume interference approaches compute the volume containing all the points occupied by objects during the time span (swept volume). This can be formalized in the following way:

$$Sw(\Lambda, O) = \{x \mid (\exists y, t) \quad x = \Lambda(t)(y)\}$$

The *Sw* operator is equivalent to an extrusion followed by a projection operation back into the original space. Therefore a test between two swept volumes is not sufficient to determine if the corresponding two objects will collide. The relative motion between the two objects has to be also taken into account to obtain a necessary and sufficient condition. The generation of swept volumes is also computationally expensive and a set of restrictions on shapes and trajectories of objects are often introduced [9][15].

Multiple interference detection techniques select a set of time values for which the collision test has to be performed. The sampling rate is crucial for the algorithm performance, as a too coarse sampling may miss a collision, while a too fine one could be computationally expensive. The simplest set of values is chosen as an arithmetic progression; given *[t_s,t_f]* the interval of time to be monitored, the time steps can be chosen as:

$$t_i = t_s + \frac{i(t_f - t_s)}{n}$$

where the value of *n* strongly affects the system performance. Another possible choice is to adopt an adaptive sampling by an estimation of the lower bound of the distance between a pair of objects based on an upper bound of their relative velocities [3][7].

Trajectory parameterization techniques exactly determine the collision instant by expressing object trajectories as functions of the time parameter. The complexity of the computation does depend on the trajectories; an arbitrary trajectory can lead to polynomials of order five and above that cannot be analytically solved. Several solutions have been proposed to tackle this issue, among these it is worthwhile to cite [17] and [27].

Interference detection

The first three approaches described in the previous Section need to test the intersection between pairs of 3D/4D volumes. The efficiency of this test (often called static interference detection) strongly affects the performance of the whole collision detection algorithm.

In [23] it has been proved that the intersection tests between two convex polyhedra objects can be performed at most in linear time. A pre-processing operation can reduce the complexity to *O(logn logm)*, where *m* and *n* are the number of vertices of the two polyhedra under analysis

The problem to cope with non-convex polyhedra is more complex. Typically, a non-convex polyhedron is decomposed into convex parts by a pre-processing step to be performed once. If the static interference detection works exclusively with convex polyhedra, the decomposition phase will produce a set of convex polyhedra [1]. Otherwise, if only the faces of the polyhedra have to be convex, a surface decomposition algorithm is sufficient [5]. Straightforward approaches able to directly cope with non-convex polyhedra have been also proposed. A quadratic number of intersection points has to be at most computed to determine the intersection between two "general" polyhedra [4], even if some improvements to reduce the computational cost have been proposed.

The performance of the whole collision test algorithm depends on two factors: the efficiency of the static interference detection and the number of times the static interference detection has to be applied. In order to reduce the occurrence of the intersection test, several strategies of space bounding have been presented. In particular, hierarchical bounding volume strategies proved to be very efficient. A hierarchical approach has two main advantages: it is often possible to detect a non-interference case at the first level of the hierarchy and it is possible to shrink the portion of space where a collision can occur.

Some methods perform either a recursive or uniform space subdivision (Octrees [14] BSP-trees [24], regular grids [11], and so on). Checks for interferences have to be performed only for pairs of objects that occupy the same portion of space. Other methodologies build a hierarchical structure surrounding objects. Typically, simple shape bounding volumes such as spheres and boxes are recursively used to encapsulate objects.

In this way, tight bounding object "approximations" are obtained and interference tests can be quickly performed. Spheretrees are used for instance in [8], [16], and [25], while axis-aligned bounding boxes (AABBs) and oriented bounding boxes (OBBs) have been proposed in [28] and [13], respectively. Bounding box groups (BBGs) are used in [26] to efficiently manage encapsulations of CSG models. Discrete orientation polytopes (k-dops) have been proposed in [20] in order to overlap the poor tightness of spheretrees and AABBs encapsulations and the high cost of interference tests due to OBBs. K-dops will be described more in detail in the following Section as this methodology will be used to compare the performance (in term of computational cost and precision in determining collisions) of the proposed algorithm. Given two models and the hierarchies used to approximate them, the total cost *T* for intersection can been quantified as in [13]:

$$T = N_v C_v + N_p C_p$$

where N_v is the number of pairs of bounding volumes tested for interference, C_v is the cost to test a pair of bounding volumes, N_p is the number of primitives tested for intersection, and C_p the cost for testing a pair of primitives. The previous formula does not consider the cost due to update the bounding volume hierarchy in the event of a rotation operation. If the bounding elements are spheres, this cost can be negligible, otherwise it can be considered as in [20]:

$$T = N_v C_v + N_p C_p + N_u C_u$$

where N_u and C_u are the number of nodes of the bounding hierarchy and the cost necessary to update a node, respectively. The previous two formulas refer to schemas that use bounding volume hierarchies, rather than being generic collision cost functions. Generic metrics for the evaluation of collision detection techniques are also proposed in [21].

Temporal coherence is also used concurrently to spatial information to reduce the pairs of objects that need to be considered for collision. I-COLLIDE [6] uses spatial and temporal coherence in addition to a sweep and prune technique to reduce the number of interference tests.

Discrete orientation polytopes (k-dops)

Discrete orientation polytopes (dops) are convex polytopes whose facets are determined by half-spaces whose outward normals come from a small fixed set of orientations. The number of orientations is denoted by *k* hence k-dops. In Figure 1 three different kinds of 2D bounding volumes are shown: AABB, OBB, and k-dop. In Figure 1 $k=8$ orientations are used to generate the 2D k-dop. AABBs in 3D are the same as six-dops with facets aligned with coordinate axes.

In [20] only k-dops whose discrete orientation normals come as pairs of collinear, but oppositely oriented, are considered vectors; in this way, the cost of testing the interference between two k-dops is reduced; in particular overlap tests have to be performed on *k/2*. The number of *k* does affect the performance of the algorithm; higher values of *k* (tighter approximations) produce lower N_v, N_p, and N_u values but, on the other hand, increase the intersection costs C_v ($C_v=O(k)$) and C_u ($C_u=O(k^2)$). A performance comparison setting $k=6, 14, 18, 26$ is provided in [20].

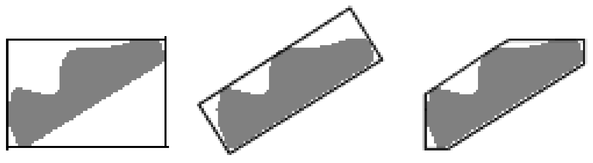

Figure 1: three bounding approximations of the same object: AABB (left) OBB (center) k-dop (right).

3. GOAL AND BASIC IDEA

A hybrid approach divides the collision detection task in two steps [19]. The first step (broad phase) reduces the number of interference tests by bounding volumes, while the second step (narrow phase) performs more accurate tests on primitives. This paper focuses on the broad phase using different kinds of bounding volumes for the environment and the moving objects, in order to design an effective collision detection algorithm for real-time/interactive systems.

The basic idea is to consider both recursive space subdivision and bounding volume hierarchies together. A recursive space subdivision can narrow the portion of space where interference tests have to be performed and it is used to identify areas where objects are present. The efficiency of this strategy depends on the granularity of the decomposition. A coarse granularity requires low computational times, on the other hand, it does not allow high precision degrees. Finer decompositions guarantee higher precision also where it is not necessary. An adaptive decomposition can overcome these problems and a high precision degree is obtained only where required, that is, where a large number of objects is present. It is worthwhile to stress that the environment decomposition has to be performed just once as it only considers static object placements.

The recursive space subdivision allows to narrow the space to be considered for interference tests, but it does not determine which static objects is affecting every single portion/cell of space. This problem can be tackled by encapsulating the objects within bounding volume hierarchies. AABBs can be effectively used to bound static objects as they require neither rotation nor translation operations. When the number of AABBs affecting, completely or partially, a cell is greater than a preset threshold, the cell can be further subdivided.

Dynamic objects can involve translation and/or rotation operations and spheretrees are well suited to bound them. Moreover, intersection tests among AABBs and spheres can be performed quickly as well as tests to check the presence of a sphere within a cell.

Three possible events can occur:

1. a cell is empty: no interference can occur;
2. a cell is completely occupied: two or more bounding entities have been detected within the same portion of space and an interference certainly occurs;
3. a cell is partially occupied: two or more bounding entities have been detected within the same portion of space and an interference occurs, if bounding volumes intersect.

The broad phase does not allow to exactly determine if a collision occurs but it can significatively reduce the number of primitives to be considered during a successive narrow step.

4. THE PROPOSED ALGORITHM

The algorithm assumes that all the bounding entities have already been computed in a pre-processing phase. This can be obtained during the modelling task; the modeler could compute both AABBs and spheretrees at different levels of detail/precision. The algorithm is shown in Figure 2. The environment description is loaded as well as the threshold used during the recursive subdivision of the environment in cells, and the program asks the user for a level of precision for the bounding entities of moving objects. The default level is *0*, that means just one sphere is used to encapsulate the entire object; otherwise, a set of spheretrees is loaded from file. Paths for moving objects are also loaded and this ends the input of data.

At this point, the recursive space subdivision of the environment can start. Borders of the environment are computed and a subdivision in uniform cells begins. Every static object is bounded by one or more AABBs; from the point of view of the data structure, this can be seen as a list of lists where the main list addresses the object and the secondary list represents the set of AABBs encapsulating the object itself. For each cell the number of AABBs, partially or fully enclosed, is determined; if the value is greater than the threshold, the cell is recursively subdivided.

This process ends when each cell contains a less or equal number of AABBs than the preset threshold. All these operations have to be performed once and they do not affect the interference detection phase. Afterwards, moving objects are encapsulated either by a single sphere (precision level equal to *0*) or by a spheretree at the precision level specified. Bounding spheres are tested to be completely or partially enclosed in cells for each step of simulation. If the cell under analysis is empty, no interference occurs, otherwise interference has to be tested. When all steps of simulation have been performed, the algorithm ends and statistics are provided. Intersection tests between primitives belonging to objects are not performed; therefore, an interference is detected, when two bounding entities overlap.

5. RESULTS AND PERFORMANCE COMPARISONS

Complexity evaluation of the algorithm

This section aims to provide a theoretical evaluation of the algorithm complexity. Let us consider the flow chart shown in Figure 2; we can identify the following phases:

1. regular and recursive environment subdivision;
2. check of moving object positions for each time step;
3. interference tests.

Let us denote with:

- N_s: number of static objects of the environment (each object is encapsulated by means of an AABB);
- N_d: number of dynamic objects moving within the environment (each object is encapsulated by means of a spheretree);
- C: number of cells obtained in the environment subdivision phase;
- T: threshold used to set a maximum number of AABBs affecting a single cell;
- S: number of spheres used to build the spheretree encapsulating a dynamic object.

The complexity of the first phase depends on the object distribution. The entire environment is initially considered as a unique cell and is regularly subdivided into a grid of C cells. For each cell the number of AABBs partially or completely

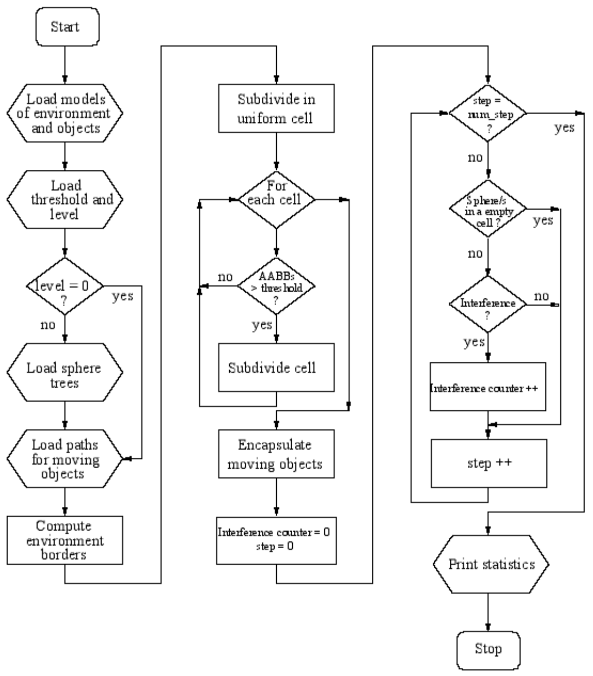

Figure 2: flow chart of the algorithm.

enclosed is determined. This operation is linear with N_s ($O(N_s)$). Let us assume a uniform object distribution that does not require further subdivisions; in this case, no more than T objects/AABBs can be detected for each cell of the set C.

The algorithm has to determine the cells partially or completely affected by moving objects (phase 2). This phase depends linearly on: number of cells, number of static objects, number of dynamic objects, number of spheres used to build spheretrees. If a cell cannot contain more than T objects, the complexity of phase 2 is: $O(C\ T\ N_d\ S)$. But C, T, and S are constant values, therefore the complexity can be denoted as $KO(N_d)$.

The final phase tests, for each dynamic object, interferences among the S spheres of spheretrees with, at most, T AABBs. The number of cells involved in this operation does depend on dynamic object sizes. This means that the complexity of phase 3 is: $O(N_d\ T\ S\ C')$ (where the term $C'= sC,\ s \in [0,1]$ denotes a percentage of the whole set C). Again, the complexity of this phase linearly depends on N_d.

Phase 1 can be performed once as a pre-processing step. The real cost of the algorithm is due to phases 2 and 3. It is important to stress, phases 2 and 3 do not depend on N_s, that is, the complexity of the algorithm does not depend on environment topology/complexity. This analysis will be confirmed by experimental results presented in the following Section, where different (for complexity and topology) environments are considered.

Experimental results

The proposed algorithm (called CDFast) has been compared with the solution proposed in [20] (called QuickCD). For sake of fairness it must be outlined that QuickCD detects intersections between all primitives (triangles) belonging to objects, while CDFast only detects interferences between bounding entities. Three different scenes have been considered to test algorithm performances. In order to guarantee the compatibility with QuickCD, all environments are enclosed in the unitary cube:

1. a room modelling a “real” environment (see Figure 3);
2. fifty spheres of radius 0.0345 randomly placed;
3. a set of objects different in shape and size randomly placed; the smallest object is a sphere of radius 0,00656 while the biggest one is a hyperboloid bounded into a box of dimensions: 0.27x0.31x0.23 (see Figure 4).

A torus and the solid shown in Figure 5 have been used as test moving objects. The torus has a radius of 0.11 and a ring 0.05 thick. On the other hand, the second object can be bounded into a box of dimensions: 0.4x0.23x0.23.

All tests have been performed on an AMD Duron 650 MHz PC running Linux. Table 1 and Table 2 provide statistics about computational times (in ms) and number of collisions detected for the torus and the complex object of Figure 5 respectively. Both objects follow a random path of 10.000 steps within environments, in this way, unexpected trajectory changes can be well simulated. K-dops with *k=18* have been used in QuickCD, as the tests presented in [20] show that *k=18* provides the best trade-off between precision and computational time. Two different encapsulations have been considered for moving objects in CDFast: a simple unique sphere and a spheretree at the first level of detail. The spheretree encapsulating the torus is composed by nine spheres while the second moving object is bounded by a five spheres spheretree (see Figure 5). Statistics show that computational times of CDFast do not depend on the environment complexity (about the same times are needed to complete the path within the three environments that strongly differ for topology and number of static objects); this is a very important attribute as a constant frame rate can be achieved independently of scene. The approximation of moving objects by means of just one sphere provides lower computational times with respect to a spheretree encapsulation, but it also reports a larger number of “false”' interferences. Spheretree approximations provide a good trade-off between computational time and precision; moreover, if polygons (triangles) have to be tested for intersection in a successive narrow phase, only the primitives encapsulated within spheres fully or partially contained in a not empty cell have to be considered and not all primitives belonging to the moving object. Finally, it is possible to observe how QuickCD provides the best performance when static and dynamic objects can be well bounded by k-dops (test scene 2 and moving object torus).

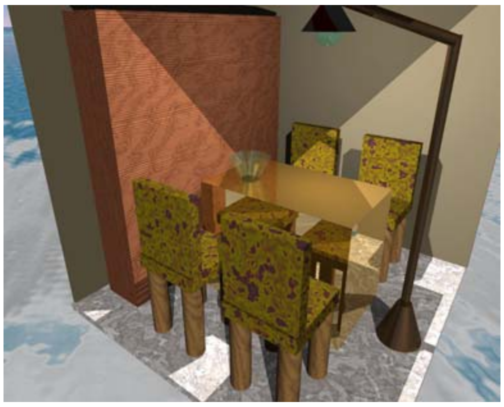

Figure 3: test scene 1.

6. CONCLUSIONS

This paper proposes a collision detection algorithm for interactive/real time applications. A hybrid approach is used to reduce the number of interference tests to be performed between bounding entities. The environment is recursively subdivided by uniform grids and static objects are encapsulated within AABBs. Moving objects are bounded by means of spheretrees, at different level of details, to minimize the cost of translation and rotation operations. A theoretical evaluation and experimental results show that computational times of the proposed techniques do not depend on scene complexity, in this way the frame rate is only related to the sampling frequency.

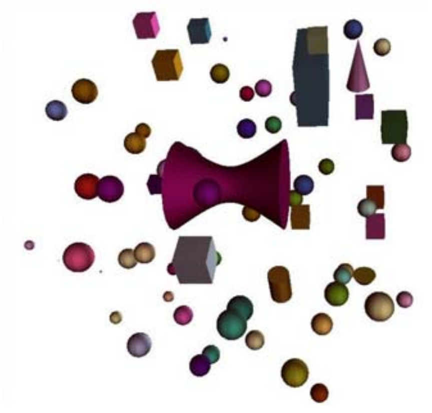

Figure 4: test scene 3.

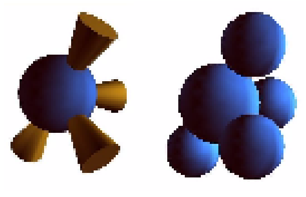

Figure 5: The second moving object and its encapsulation by a spheretree.

Scene	Quick CD Comp. Time	CDFast – one sphere Comp time	CDFast – spheretree Comp. time
1	11390	3240	3430
2	7420	3090	3250
3	11460	3100	3260
Scene	QuickCD Coll. number	CDFast – one sphere Coll. numebr	CDFAst – spheretree Coll. number
1	198830	140355	12658
2	606330	146554	26966
3	166130	67268	3250

Table 1: statistics obtained with the torus.

Scene	Quick CD Comp. Time	CDFast – one sphere Comp time	CDFast – spheretree Comp. time
1	210540	3920	5680
2	76720	3480	4930
3	170700	3179	4300
Scene	QuickCD Coll. number	CDFast – one sphere Coll. numebr	CDFAst – spheretree Coll. number
1	1177540	802562	169214
2	3166900	737921	478169
3	790410	545721	57952

Table 2: Statistics obtained with the complex object of Figure 5.

7. REFERENCES

[1]M. Bern. **Triangulations**. In Handbook of discrete and computational geometry. JE. Goodman and J. O'Rourke, Eds. CRC-Press, 1997.

[2]S.A. Cameron. Collision detection by four-dimensional intersection testing. **IEEE Transaction on Robotics and Automation**, 6(3):291-302, 1990.

[3]S.A. Cameron. A study of the clash detection problem in robotics. In **Proceedings of the IEEE Int. Conf. on Robotics and Automation**, vol. 1, pp. 488-493, 1985.

[4]J. Canny. **The Complexity of Robot Motion Planning**. MIT Press, Cambridge, MA, 1987.

[5]B. Chazelle and L. Palios. Decomposing the boundary of a nonconvex polytope. In Proceedings of the **3rd Scandinavian Workshop on Algorithm Theory**, pp. 364-375, 1992.

[6]J.D. Cohen, M.C. Lin, D. Manocha, and M.K. Ponamgi. I-collide: An interactive and exact collision detection system for large-scale environments. In **Proceedings of the ACM Int. 3D Graphics Conference**, vol. 1, pp. 189-196, 1995.

[7]R.K. Culley and K.G. Kempf. A collision detection algorithm based on velocity and distance bounds. In **Proceedings of the. IEEE Int. Conf. on Robotics and Automation**, vol. 2, pp.1064-1069, 1986.

[8]J. Dingliana and C. O'Sullivan. Graceful Degradation of Collision Handling in Physically Based Animation. **Computer Graphics Forum**. (Proceedings of Eurographics'2000), 19(3):239-247, 2000.

[9]A. Foisy and V. Hayward. A safe swept volume method for collision detection. In **The Sixth International Symposium of Robotics Research**, pp. 61-68, 1994.

[10]F. Ganovelli, J. Dingliana., and C. O'Sullivan. BucketTree: improving collision detection between deformable objects. In **Proceedings of SCCG**, pp. 156-163, 2000.

[11]A. Garcìa--Alonso, N. Serrano, and J. Flaquer. Solving the collision detection problem. **IEEE Computer Graphics and Applications**, 14(3):36-43, 1994.

[12]E.G. Gilbert and C.P. Foo. Computing the distance between general convex objects in three-dimensional space. **IEEE Transaction on Robotics and Automation**; 6(1):53-61, 1990.

[13]S. Gottschalk, M.C. Lin, and D. Manocha. Obb-tree: A hierarchical structure for rapid interference detection. In **Proceedings. of ACM Siggraph**, vol. 30, pp. 171-180, 1996.

[14]K. Hamada and Y. Hori. Octree-based approach to real-time collision-free path planning for robot manipulator. In **ACM96-MIE**, pp. 705-710, 1996.

[15]M. Herman. Fast, three-dimensional, collision-free motion planning. In Proceedings of the **IEEE Int. Conf. on Robotics and Automation**, vol. 2, pp. 1056-1063, 1986.

[16]P.M. Hubbard. Collision detection for interactive graphics applications. **IEEE Transaction on Visualization and Computer Graphics**, 1(3):218-230, 1995.

[17]P. Jimènez and C. Torras. Collision detection: a geometric approach. In **Modelling and Planning for Sensor Based Intelligent Robot Systems**, World Scientific Pub. Co., pp. 68-85, 1995.

[18]P. Jimènez., F. Thomas, and C. Torras. 3D collision detection: a survey. **Computers and Graphics**, 25(2):269-285, 2001.

[19]Y. Kitamura, H. Takemura, N. Ahuja, and F. Kishino. Efficient Collision Detection Among Objects in Arbitrary Motion Using Multiple Shape Representations. In **Proceedings of 12th IAPR Int. Conf. on Pattern Recognition,** vol. 1, pp. 390-396, 1994.

[20]J. Klosowski, M. Held, J. Mitchell, H. Sowizral, 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.

[21]E. Levey, C. Peters, and C. O'Sullivan. New metrics for evaluation of collision detection techniques. In **Proceedings of WSCG**, vol. 1, pp. 140-146, 2000.

[22]M.C. Lin and S. Gottschalk. Collision detection between geometric models: a survey. In **IMA Conference on Mathematics of Surfaces**, vol. 1, pp. 602-608, 1998.

[23]M.C. Lin and J.F. Canny. A fast algorithm for incremental distance calculation. In Proceedings of the **IEEE Int. Conf. on Robotics and Automation**, vol. 2, pp. 1008-1014, 1991.

[24]B.F. Naylor, J.A. Amatodes, and W.C. Thibault. Merging bsp trees yields polyhedral set operations. In **Proceedings of ACM Siggraph**, vol. 24, pp. 115-124, 1990.

[25]C. O'Sullivan and J. Dingliana. Real-Time Collision Detection and Response Using Sphere-Trees. In Proceedings of SCCG, pp. 83-92, 1999.

[26]A. Sanna and P. Montuschi. Spatial Bounding of Complex CSG Objects. **IEE Proceedings Part E**, 142(6):431-439, 1995.

[27]E. Schömer and C. Thiel. Efficient collision detection for moving polyhedra. In Proceedings of the **Eleventh Annual Symp. On Computational Geometry**, pp. 51-60, 1995.

[28]G. Van der Bergen. Efficient collision detection of complex deformable models using aabb trees. **Journal of Graphics Tools**, 2(4):1-13, 1997.