CAD/Graphics'2001
August 22-24, Kunming
International Academic Publishers

A PARALLEL COLLISION DETECTION ALGORITHM BASED ON HYBRID BOUNDING VOLUME HIERARCHY *

Wan Huagen, Fan Zhaowei, Gao Shuming and Peng Qunsheng
State Key Lab of CAD&CG, Zhejiang University, 310027, Hangzhou, PRC
hgwan@cad.zju.edu.cn | wanhuagen@hotmail.com

Abstract One of the key problems with current interactive virtual environments is real-time collision detection. In this paper, a parallel algorithm for real-time, exact collision detection is presented. The approach adopts a hierarchical adaptive space subdivision scheme to construct a hybrid bounding representation of arbitrary non-convex polyhedra for attaining its speed, and speeds up collision detection further by traversing the built hybrid bounding volume hierarchy in parallel. The algorithm falls into MIMD synchronous parallel algorithms and is implemented with multi-threads, which enable it to run on both single processor computers and multi-processor computers. It is suitable for dynamic and complex scenes.

Keywords: virtual environment, collision detection, hierarchical bounding representation, parallel algorithm

1. Introduction

The objective of collision detection is to automatically report one or more geometric contacts which are about to occur or have actually taken place between objects. This problem has been investigated far and wide in computer animation, computational geometry, robotics, virtual prototyping, and even, haptic rendering of graphical models. In these applications, collision detection is used to constrain an object's motion so as to avoid the penetration of the object into other objects. Though collision detection has been studied for a relatively long time, it re-inflames the enthusiasm of many researchers recently with the fast development of interactive virtual environments due to the following two major reasons:

Firstly, though the problem of collision detection can be solved as a function of time if the positions and orientations of objects are known in advance, things seem to be somewhat different in the case of an interactive virtual environment. In fact, in an interactive virtual environment, the user may change the position and orientation of an object at any time, which makes collision detection much more difficult than the traditional one. Secondly, collision detection must be fulfilled in real-time so as to satisfy the user's interactivity with the environment.

This paper presents a parallel algorithm for real-time, exact collision detection for arbitrary non-convex polyhedra. The algorithm adopts a hierarchical adaptive space subdivision scheme to construct hybrid bounding representation of objects to gain its speed, and accelerates the process of collision detection further by traversing the built hybrid bounding volume hierarchies in parallel.

The rest of the paper is organized in the following manner: some of the literatures concerning collision detection are reviewed in Section 2. Section 3 illustrates the main idea of the algorithm. In Section 4, the details of the collision detection algorithm based on hierarchical hybrid bounding volume are given. We present the parallel algorithm in Section 5. Section 6 discusses about the implementations and experimental results. Finally, we make some conclusions and discuss some of the future works in Section 7.

2. Related Literatures

The research on collision detection can be dated back to the late 1970s [Lin98].

In computational geometry, the problem of finding the intersection of two polyhedra was first investigated [Mull78]. Later researchers realized that the collision detection problem itself would be of much significance and could be performed with higher efficiency [Dobk85, Chaz87, Reic88].

In robotics, the goal of collision detection can be primarily described as finding collision-free paths for a robot between its surrounding obstacles [Lato91]. While M. Erdmann et al. proposed a particular method with which the problem was solved in configuration space [Erdm87], some other efficient algorithms suitable for distance computation were also appropriate for this kind of problems [Cann86, Gilb88, Seid90, Lin91, Quin94, Came96, Mirt98, John98]. An approximate approach

* Supported by National Natural Science Foundation of China and Excellent Young Teacher Foundation of Education Ministry of China.

was also proposed on the ground that the accurate collision result was not the main issue [Clif92].

Collision detection does play an active role in virtual prototyping, where it is utilized for the purpose of refining designs without productions of physical prototypes in the early design stage. Inspired by BSP trees, k-d trees and balanced bipartitions used in the area of VLSI layout algorithms, G. Zachmann presented a fast collision detection algorithm for interactive virtual prototyping [Zach97]. The algorithm adopted the so-called "BoxTree" to attain its speed. However, the recursive traversal of the BoxTree prevented the algorithm from achieving better results.

In the field of haptic rendering, D. Ruspini developed a collision detection algorithm which worked in a similar manner to quick-sort and employed bounding sphere representation similar to that first proposed by Quinlan [Rusp99, Quin94].

Various kinds of representations of objects and bounding volume hierarchies are proposed and used recently to gain efficiency of collision detection. The representatives are: "S-bounds" introduced by S. Cameron as a means of speeding up intersection evaluation for CSG models [Came91], sphere trees proposed by Hubbard and Quinlan independently [Hubb93, Quin94, Hubb96], axis-aligned bounding boxes (with cubes as a special case) for utilizing spatial and temporal coherence [Cohe95], the tight-fitting oriented bounding box trees (OBBTrees) [Gott96], discrete orientation polytopes (k-DOPs) which are superior approximations to bounded geometry [Klos96], octrees [Yu96], Box trees [Bare96, Zach97], and spherical shell tree which is a higher order bounding volume for matching curvature of the underlying 3D geometry like Bézier patches and NURBS [Kris98].

Up to now, however, there are little research on the parallel paradigm of collision detection algorithms. The only one work we can find relating with both collision detection and parallel processing is [Wang96], in which an object-oriented approach of collision detection in a distributed virtual environment was proposed.

3. Ideas Behind Our Approach

Given two arbitrary non-convex polyhedra A and B, collisions between A and B occur if and only if one or more of the following conditions are satisfied.

(a) there exists at least one edge in polyhedron A which intersects with a face in polyhedron B, or vice versa;

(b) Polyhedron A totally contains polyhedron B, or vice versa.

A "fundamental" collision detection algorithm which detects the collisions by primarily checking the edge-face intersections between the two objects is derived from the above-mentioned sufficient and necessary condition. And the algorithm is adopted in our approach for determining whether the two objects exactly collide or not. Limited to the space, this simple "fundamental" algorithm is not described in detail.

The "fundamental" algorithm can be accelerated certainly by involving some approaches described in Section 2, for instance, bounding box trees or bounding sphere trees. However, while bounding spheres are characterized by easiness of intersection testing and independence from rotating operations, they fit relatively loosely to the original objects. For bounding boxes, things seem to go to another extreme. The bounding box can surround an object much more tightly, but it is more time-consuming to detect if two boxes collide than that of two spheres, especially when the boxes are inclined relative to axes of the coordinate system. The idea behind our algorithm is that, would we incorporate the merits of both bounding boxes and bounding spheres, hence to fully utilizing the spatial coherence between objects? The answer lies in the hybrid bounding volume hierarchy which will be described in more detail in the next section. This makes sense to the collision detection problem on the one hand.

On the other hand, parallel processing is being paid much attention to as applications become more and more intricate. Generally speaking, there exist several ways for the development of parallel algorithms. First, the intrinsic parallelism in an existing non-parallel algorithm can be inspected and exploited to the development of parallel algorithms; Second, An existing parallel algorithm can be modified to be suitable for the case of similar issues. Third, to design a brand-new parallel algorithm from scratch according to the question itself, and representative approaches are balanced-tree method, doubling technique, divide-and-conquer strategy, partitioning principle, pipelining technique, accelerated cascading algorithm, and symmetry-breaking technique [JáJá92].

Divide-and-conquer strategy is utilized in designing our parallel algorithm of collision detection. That is to say, the task of collision detection is divided into some subtasks which are similar to the original task till the subtasks can be solved in a relatively easy way. Due to the specialty of the collision detection task, the collision detection process is in nature an or-tree search process. This search process can either be fulfilled via Depth-First-Search (DFS) recursively or via Width-First-Search (WFS) level by level. A WFS-based parallel traversal is used in our algorithm to achieve the exact collision detection result as fast as possible, which discards the portions of objects which do not collide

with other objects by the pre-check done on the bounding volume level.

Before presenting the specific parallel algorithm, we first describe the procedure of constructing the hybrid bounding volume hierarchy as well as the non-parallel process of traversing the bounding volume hierarchy in next section.

4. Construction and Traversal of the Hybrid Bounding Volume Hierarchy

4.1 Constructing the Hybrid Bounding Volume Hierarchy

As stated above, an appropriate hierarchical bounding volume representation of an object is intimately related with the efficiency of the collision detection algorithm. In our approach, a hybrid bounding volume hierarchy is proposed, which attempts to fully utilize the spatial coherence between objects by incorporating the merits of both bounding box and bounding sphere. This means that whenever the collision detection algorithm steps down one level in the hierarchy, and it discards one of the sub-nodes, we want it to be able to discard as many polygons as possible. This leads to a space subdivision scheme which tries to balance the tree in terms of polygon counts. Fortunately, the balanced binary tree happens to be a requisition for an efficient parallel algorithm, too.

Given a set of polygons of an object, the construction of the hybrid bounding volume hierarchy is a four-pass process as follows:

First, we construct a small bounding sphere for each polygon of the object with a high efficiency bounding sphere generation algorithm [Ritt90]. This sphere is named “underlying sphere” and serves at least two purposes: (a) to help fast subdivision of the bounding volumes, and (b) to be used as a pre-check aid in the “fundamental” algorithm before the edge-face intersection tests are performed.

Second, The root node of the hierarchy is built. The root node mainly consists of the following attributes: all underlying spheres (hence all polygons), an axis aligned bounding box of all of the polygons, and a bounding sphere of all of the polygons.

Third, the root node is subdivided into two sub-nodes which have the same attributes as those of the root node. This is accomplished by: (a) The bounding box of the root node is divided into two sub-boxes (say, left and right sub-box) by an axis aligned dividing plane; (b) Those underlying spheres (hence polygons) fully covered by the left or right sub-box are considered to be contained in the corresponding left or right sub-node. Those underlying spheres (hence polygons) which cross both sub-boxes are included in both left and right sub-nodes. (c) The construction of bounding spheres of the sub-nodes takes almost the same steps as that of the root node, but only those polygons contained in the corresponding sub-node are considered.

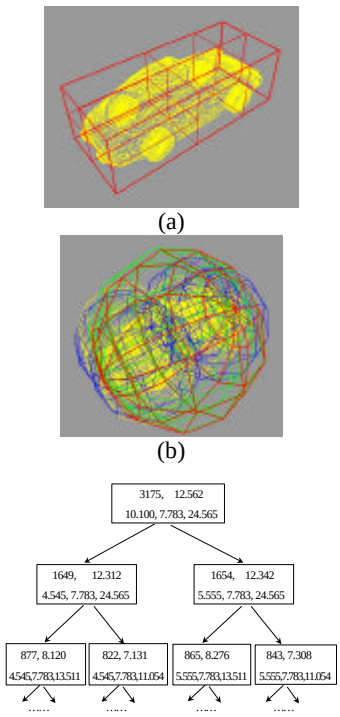
To make an approximately balanced binary tree which is characterized by the number of underlying spheres contained in each sub-node, and to minimize the number of those underlying spheres which cross both sub-boxes, we adopt a penalty function first proposed in [Zach97] to determine the dividing plane:

$$f(p) = | n_l - n_r | + l n_c$$

where p is the axis aligned dividing plane, n_l and n_r are the number of underlying spheres fully covered by the left and right sub-boxes respectively, n_c is the number of those underlying spheres crossing both left and right sub-boxes and l is a constant.

Note that underlying spheres rather than the polygons are used in the determination of the dividing plane. This is because the determination of whether a sphere is fully covered or crossed by an axis aligned box is more efficient than that of a polygon.

Finally, The same subdivision procedures described in the above step are recursively applied to both newly generated sub-nodes to acquire the final hierarchical hybrid bounding volume representation of the object.



(a) (b) (c)

Fig.1 Hybrid bounding volume hierarchy of a car

Appropriate exits are crucial for a recursive procedure. The following three conditions are provided in our approach to terminate the recursive construction of the hybrid bounding volume hierarchy:

- The recursion depth achieves the maximally pre-defined depth of the tree. In our implementation, the pre-defined depth of the tree is associated with the polygon number of

the object. The more polygons an object consists of, the deeper the resulting binary tree will be.

- The number of underlying spheres (i.e. the number of polygons) contained by the sub-node being processed reaches the pre-defined minimal value (for example, 4, in our implementation).
- The sub-node being processed has its left or right son containing almost as many underlying spheres as the father. This condition is set due to the reason that unbalanced binary tree will be generated if we allow such node to be subdivided.

Fig.1(a) and (b) illustrate respectively the hierarchy of bounding boxes and bounding spheres of a car (totally contains 3175 polygons) after 2 times of recursive subdivisions. Fig.1(c) is the resulting approximately balanced binary tree, where the two numbers in the first row of each node represent respectively the number of underlying spheres (hence polygons) contained in the node and the radius of the bounding sphere of the node, and the three numbers in the second row of each node represent the width, height and depth of the bounding box of the node.

4.2 Traversal of the Hybrid Bounding Volume Hierarchy

Once the hybrid bounding volume hierarchies of two objects A and B are constructed, collision detection between the two objects can be performed by first traversing the corresponding approximately balanced binary trees of A and B. To be concise, we introduce the concept of *task tree*. If we define the collision detection between two nodes of the hierarchies of A and B as a task, then collision detection between objects A and B by traversing the corresponding approximately balanced binary trees of A and B simultaneously can be regarded as the traversal of a task tree.

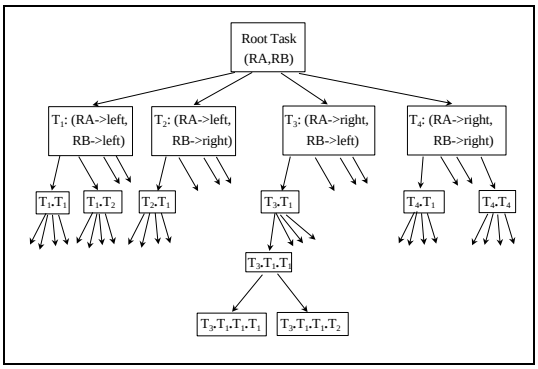


Fig.2 Illustration of a task tree

Fig.2 illustrates a typical task tree, in which RA and RB are respectively the root nodes of the built hierarchies of objects A and B, and T_1~T_4 are four subtasks of a task.

The collision detection process is in nature an or-tree search process of the task tree. On the one hand, if collision is found at some subtask, then collision occurs between the two objects A and B. On the other hand, if no collision can be found after all the subtasks are processed, then no collision occurs between object A and object B. Thus the collision detection result (TRUE or FALSE) of each task can be achieved by the logical OR of the collision detection results of its all subtasks. That is to say, the collision detection problem itself can be considered as the search process of a task tree for its logical OR result. Taking advantage of task tree, the WFS-based traversal of the hybrid bounding volume hierarchy can be given in pseudo codes in Fig. 3, in which a queue, LIVESET, is used to store the task nodes which need to be processed.

The difference between the above algorithm and the traditional one lies in following three aspects:

First, before performing the exact collision detection between the two objects, the pre-checks of sphere-sphere, sphere-box and box-box intersections are used to discard the polygons that don't collide with others obviously.

Second, a criterion mainly based on the volume ratio of the bounding sphere and the bounding box of the same node is used to determine whether the bounding box level pre-check is necessary or not. Currently, the threshold (what the function RatioThreshold( ) returns) for the volume ratio is associated with the level of current node in its bounding volume hierarchy, and is determined in a statistical sense after all the volume ratios of its sibling nodes are taken into account.

Third, the algorithm of the intersection tests between bounding volumes is devised based on following three issues:

- The sphere-sphere intersection can be done in a very high efficiency manner. First, an axis aligned bounding cube is built for each bounding sphere. The simple comparison between coordinates of these axis aligned bounding cubes can be used as a pre-check before the real intersection test of the spheres is performed.

  Second, the inequation $\|C_1C_2\|^2 \le (R_1+R_2)^2$ is used to determine whether two spheres intersect, where C_1 , C_2 are centers of the two spheres, and R_1, R_2 are radius of the two spheres.

Note that the square of the distance rather than the distance itself is used to aviod the time-consuming calculation of square roots.

- The overlap test algorithm for oriented bounding boxes [Gott96] is adopted for box-box intersection test.
- The intersection test between the bounding sphere and the bounding box is performed in the local coordinate system (LCS) of the bounding box.

```
Input: root node of the task tree (RA, RB)
Output: whether A and B collide // TRUE: collision, FALSE: no collision
WFS-Traversal(RA, RB)
{
  LIVESET ← root task node (RA, RB) //push the root node in queue
  while (LIVESET != NULL)
    (a1, b1) ← LIVESET; // pop a task node up from queue
    SplitNode = TRUE;
    if ( not BoundingSpheresOverlap(a1, b1) ) {
      SplitNode = FALSE ; // no collision, hence no subtasks
    else // bounding spheres do intersect
      Flag1=(Sphere2BoxVolumeRatio(a1) >= RatioThreshold(a1) );
      Flag2=(Sphere2BoxVolumeRatio(b1) >= RatioThreshold(b1));
      if (Flag1) // bounding box of a1 should be respected
        if (Flag2) // bounding box of b1 should be respected
          if ( not BoundingBoxsOverlap(a1 , b1) ) {
            SplitNode = FALSE ; // no collision, hence no subtasks
          fi // bounding boxes not overlap
        else // Flag2 == FALSE, bounding box of b1 ignored
          if ( not BoundingBoxBoundingSphereOverlap(a1 , b1) )
            SplitNode = FALSE ; // no collision, hence no subtasks
          fi // bounding box and bounding sphere not overlap
        fi // flag2
      else // Flag1 == FALSE, bounding box of a1 ignored
        if (Flag2) { // bounding box of b1 should be respected
          if ( not BoundingBoxBoundingSphereOverlap(b1, a1))
            SplitNode = FALSE ; // no collision, hence no subtasks
          fi // bounding box and bounding sphere not overlap
        fi // Flag2
      fi // Flag1
    fi // not BoundingSpheresOverlap(a1, b1)
    if (SplitNode == FALSE) // no collision, hence no subtasks
        free the memory occupied by this node;
    else // create subtasks or do collision detection
         // by adopting fundamental algorithm
      if ( IsLeafNode (a1) ) // a1 is a leaf
        if( IsLeafNode (b1 ) ) // b1 is also a leaf
          CollisionDetectionFundamentalAlgorithm(a1, b1);
          if (a1 and b1 collide) return TRUE; // collision occurs
        else // b1 is not a leaf, two subtasks need to be created
          LIVESET←( a1, b1->left);
          LIVESET←( a1, b1->right);
        fi // IsLeafNode (b1)
      else // a1 is not a leaf
        if ( IsLeafNode (b1 ) ) {
          // b1 is a leaf, two subtasks need to be created
          LIVESET← (a1->left, b1);
          LIVESET← (a1->right, b1);
        else // b1 is not a leaf, four subtasks need to be created
          LIVESET← (a1->left, b1->left);
          LIVESET← (a1->left, b1->right);
          LIVESET← (a1->right, b1->left);
          LIVESET← (a1->right, b1->right);
        fi // IsLeafNode (b1 )
      fi // IsLeafNode (a1)
    fi // SplitNode
  end-while // LIVESET
  return FALSE; // A and B not collide
} // WFS-Traversal
```

Fig. 3 WFS-based traversal of the task tree

5. Parallelization of the WFS-based Traversal

The resulting approximately balanced binary trees ensure that the complexity of each subtask equates in the rough. Moreover, the subtasks at the same level of the task tree, i.e. sibling nodes, are independent from one another. These two issues make the algorithm seem to be particularly suitable for parallelization. Our scheme for parallel traversal of the task tree is described in Fig.4 on the basis of the WFS-Traversal and the above analysis, in which LIVESET is also used as a queue to store the task nodes needing to be processed.

```
Input: root node of the task tree (RA, RB)
Output: whether A and B collide // TRUE: collision, FALSE: no collision
Parallel-Traversal(RA, RB)
{
  //push the root node in queue LIVESET
  LIVESET ← root task node (RA, RB)
  while (LIVESET != NULL)
    (a1, b1) ← LIVESET; // pop a task node up from queue
    for (all task nodes in the LIVESET , parallel processing)
      if (the current task finds there is no intersection
        between the bounding volumes)
        cancel current task;
      else
        if (current task contains a leaf node) {
          invoke CollisionDetectionFundamentalAlgorithm()
                 at current task;
          if (the result is TRUE)
            cancel all the other tasks being parallelly processed;
            return TRUE; // collision found
          fi
        else // current task contains intermediate node
          push subtasks into LIVESET in a similar manner
          with the WFS_Traversal algorithm.
        fi // current task contains …
      fi //the current task finds no intersection between bounding volumes
      the main thread waits for the completion of
      all other tasks parallelly processed.
    end-for // all task nodes in the LIVESET , parallel processing
  end-while // LIVESET
  return FALSE; // A and B not collide
} // Parallel-Traversal
```

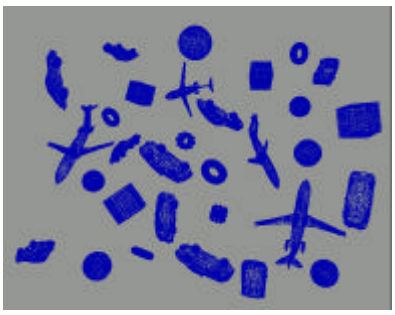
Fig. 4 Parallel Collision Detection Algorithm

6. Implementations and Experimental Results

The above algorithm has been implemented on an SGI ONYX2 with 4 CPUs and 2 graphics pipelines in C language. The multithreaded programming technique under the POSIX threads standard, POSIX.1c, is used for the parallel processing purpose. Inter-thread communication is achieved by sharing of the same address space. Different threads of a multithreaded program can run on different CPUs simultaneously with no special input from the user and no effort on the part of the programmer, which realizes parallelism [Lewi96]. In fact, multi-thread is an efficient way for application

developers to exploit the parallelism of the hardware. In general, whether a multithreaded program can run or not is independent of the number of processors. Therefore, the multithreaded implementation of our algorithm enables the algorithm to run on both single processor machines and multi-processor machines.

Two testing cases were plotted for timing test. In the first case, 31 objects (totally contain over 80,000 polygons) move at random inside a "crowded space" (i.e. most likely collisions between objects occur) (Fig.5 (a)). At start-time, each object was given randomly an initial position, and initial translational and rotational velocities. Then each object acted in its own way. Collision detection was performed between each pair of two objects. If collisions occurred at a certain time step, the collided objects bounced off each other by simply exchanging translational and rotational velocities. If the translational and rotational velocities of an object did not change for thirty time steps (i.e. the object did not collide with any other objects during this period), they will be substituted by new randomly generated translational and rotational velocities to prevent the object from moving too far from the center of the "crowded space". In the second scenario, a virtual hand is moving toward the door of a car along a specified path (Fig.5(b)). The virtual hand contains more than 30,000 triangles, while the car is composed of more than 60,000 triangles.



(a) Case 1: crowded space

(b) Case 2: virtual hand and car

Fig. 5 Scenarios for timing test

The timing test results (measured in seconds over 1000 iterations) using both parallel and non-parallel algorithms with different hierarchical bounding volumes are listed in Table 1.

Table 1 Timing test results with different algorithms

Bounding Volume \ Testing Case		Case 1	Case 2
Foundamental algorithm with bounding box	Non-parallel	10218	>50,000
	parallel	-----	-----
Hierarchical bounding box	Non-parallel	4431	724
	parallel	1724	303
hierarchical bounding sphere	Non-parallel	3465	4671
	parallel	1391	1804
hierarchical hybrid bounding volume	Non-parallel	205	234
	parallel	81	97

7. Conclusions and Future Work

Collision detection is recognized as both a fundamental problem and a bottleneck in many fields, especially in interactive virtual environment area. This paper presents a parallel algorithm for real-time, exact collision detection for arbitrary non-convex polyhedra. Compared to previous algorithms, our algorithm is characterized by:

- The algorithm is parallel. Moreover, the idea of parallelization runs through its design and implementation stages. The divide-and-conquer strategy we adopt makes it possible for the building of an approximately balanced binary tree. This guarantees that the complexity of each subtask equates in the rough.
- Hierarchical hybrid bounding sphere and bounding box representation is proposed and used to make full use of spatial coherence.
- The algorithm falls into MIMD synchronous parallel algorithms and is implemented with multi-threads, which enable it to run with much higher efficiency on both single processor computers and multi-processor computers.
- It can be used for dynamic and complex scenes as well as interactive virtual environment.

On the other hand, the proposed algorithm still has room for further speedup. Future work will concentrate on:

- The algorithm will be better parallelized by the parallelization of matrix operations, intersection tests between bounding boxes and the edge-face intersection tests in the fundamental algorithm;
- A more sophisticated criterion on the using of bounding spheres and/or bounding boxes as

means of pre-check, which takes the characteristics of a particular scenario into account will be developed;

- PVM techniques will be explored to enable the algorithm to be used in distributed virtual environments.

References

Bare96 Barequet G., Chazelle B., Guibas L., Mitchell J., and Tal A., Boxtree: A hierarchical representation of surfaces in 3d, In *Proc. of Eurographics'9*6, 1996.

Came91 Cameron S., Approximation hierarchies and s-bounds, In *Proc. of Symposium on Solid Modeling Foundations and CAD/CAM Application*s, Austin, TX, 1991, pp. 129–137.

Came96 Cameron S., A comparison of two fast algorithms for computing the distance between convex polyhedra, *IEEE Transactions on Robotics and Automatio*n, December 1996, Vol.13, No.6, pp.915–920.

Cann86 Canny J.F., Collision detection for moving polyhedra, *IEEE Transactions PAMI*, 8, 1986, pp.200-209.

Chaz87 Chazelle B., and Dobkin D.P., Intersections of convex objects in two and three dimensions, *Journal of the ACM*, 34, 1987, pp.1-27.

Chun96 Chung K. and Wang W., Quick collision detection of polytopes in virtual environments, *Proc. Of the ACM Symposium on Virtual Reality Software and Technology*, 1996, pp.125-132.

Clif92 Clifford A. Shaffer, A real-time robot arm collision avoidance system, *IEEE Transactions on Robotics and Automation*, 1992 , Vol.8, No.2.

Cohe95 Cohen J., Lin M., Manocha D., and Ponamgi M., I-collide: An interactive and exact collision detection system for large-scale environments, In *Proc. Of ACM Interactive 3D Graphics Conferenc*e, 1995, pp. 189–196.

Dobk85 Dobkin D.P., and Kirkpatrick D.G., A linear algorithm for determining the separation of convex polyhedra, *J. Algorithms*, 1985, No.6, pp.381~392.

Erdm87 Erdmann M., and Lozano-Perez T., On multiple moving objects. *Algorithmica*, 1987, No.2, pp. 477-521.

Gilb88 Gilbert E. G., Johnson D. W., and Keerthi S. S., A fast procedure for computing the distance between objects in three-dimensional space. *IEEE J. Robotics and Automatio*n, 1988, vol. RA-4, pp.193–203.

Gott96 Gottschalk S., Lin M., and Manocha D., Obb-tree: A hierarchical structure for rapid interference detection, In *Proc. of ACM Siggraph'9*6, 1996, pp. 171–180.

Hubb93 Hubbard P. M., Interactive collision detection, In *Proc. of IEEE Symposium on Research Frontiers in Virtual Realit*y, October 1993.

Hubb96 Hubbard P.M., Approximating polyhedra with spheres for time-critical collision detection, *ACM Transactions on Graphics*, Vol.15, No.3, 1996, pp.179-210.

JáJá92 JáJá J., "An introduction to parallel algorithms", Addison-wesley publishing company, 1992.

Jonh98 Johnson D., and Cohen E., A framework for efficient minimum distance computation, In *IEEE Conference on Robotics and Automatio*n, 1998, pp.3678–3683.

Klos96 Klosowski J., Held M., Mitchell J.S.B., Sowizral H., and Zikan K., Efficient collision detection using bounding volume hierarchies of k-dops, In *Siggraph'96 Visual Proceeding*s, 1996, page 151.

Kris98 Krishnan S., Gopi M., Lin M., Manocha D., and Pattekar A., Rapid and accurate contact determination between spline models using shelltrees, In *Proc. of Eurographics'9*8, 1998.

Lato91 Latombe J.C., Robot motion planning, Kluwer Academic Publishers, 1991.

Lewi96 Lewis B., and Berg D. J., "PthreadsPrimer--A guide to multithreaded programming", SunSoft Press, A Prentice Hall Title, 1996.

Lian84 Liang Y.D., and Barsky B.A., A new concept and method for line clipping, *ACM Transactions on Graphics*, 1984, No.3, pp.1-22.

Lin91 Lin M.C., and Canny J.F., Efficient algorithms for incremental distance computation, In *IEEE Conference on Robotics and Automatio*n, 1991, pp. 1008–1014.

Lin98 Lin M., and Gottschalk S., Collision detection between geometric models: A survey, In *Proc. of IMA Conference on Mathematics of Surface*s, 1998.

Mirt98 Mirtich B., V-Clip: fast and robust polyhedral collision detection, *ACM Transactions on Computer Graphics*, Vol.17, No.3, 1998, pp.177-208.

Mull78 Muller D.E., and Preparata F.P., Finding the intersection of two convex polyhedra, Theoret. Comput. Sci., 1978, No.7, pp. 217~236 .

Quin94 Quinlan S., Efficient distance computation between non-convex objects, In *International Conference on Robotics and Automation*, April 1994, pp.3324-3329.

Reic88 Reichling, M., On the detection of a common intersection of *k* convex polyhedra, In *Computational Geometry and its Applications*, 1988, vol.*333* of *Lecture Notes in Computer Science*, Springer-Verlag, pp. 180-186.

Ritt90 Ritter J., An efficient bounding sphere, in Graphics Gems (edited by Andrew Glassner), Academic Press, Inc., 1990, pp.301-303.

Rusp99 Ruspini D., Haptic rendering of graphical models, In *Course Notes of Siggraph'99*.

Seid90 Seidel R., Linear programming and convex hulls made easy, In *Proc. 6th Annal ACM Conference on Computational Geometr*y, Berkeley, California, 1990, pp. 211–215.

Wang98 Wang Z.Q., Zhao Q.P., and Wang C.W., Object-oriented collision detection method and its applications in distributed virtual environments, *(in Chinese), Chinese Journal of Computers*, 1998,Vol.21, No.11, pp.990~994.

Wils99 Wilson A., Larsen E., Manocha D. And Lin M.C., Partitioning and handling massive models for interactive collision detection, In Proc. Of Eurographics'99.

Yu96 Yu Y., Wu M., and Zhou J., An octree algorithm for dynamic interference detection using space partitioning, In *Proc. Of The 1996 ASME Design Engineering Technical Conference and Computers in Engineering Conference*, Irvine, CA, Paper Number #96-DETC/DAC-1046.

Zach97 Zachmann G., Real-time and exact collision detection for interactive virtual prototyping, In *Proc. of 1997 ASME Design Engineering Technical Conferences*, September, 1997, Sacramento, California, Paper Number #DETC97/CIE4306.5