Efficient Collision Detection Among Objects in Arbitrary Motion Using Multiple Shape Representations

Yoshifumi KITAMURA, Haruo TAKEMURA*, Narendra AHUJA† and Fumio KISHINO

ATR Communication Systems Research Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun, Kyoto, 619-02, Japan
kitamura@atr-sw.atr.co.jp

Abstract

We propose an efficient method for detecting potential collisions among multiple objects with arbitrary motion (translation and rotation) in three-dimensional (3-D) space. The method is useful for on-line monitoring and path planning in a 3-D environment in which there are multiple independently-moving objects. The method consists of two main stages. In the first, coarse stage, an approximate test is performed to identify interfering objects in the entire workspace using octree representation of object shapes. In the second, fine stage, polyhedral representation of object shapes is used to more accurately identify any object parts that might cause interference and collisions. For this purpose, specific pairs of faces belonging to any of the interfering objects found in the first stage are tested, thus performing detailed computation on a reduced amount of data. Experimental results, which demonstrate the efficiency of the proposed collision detection method, are given.

1 Introduction

Interference or collision detection methods using polyhedral shape representation [1] [2] [3] are common. They yield relatively accurate results, but must test all combinations of all the faces and edges of objects in the environment for interference.. Therefore, the computational cost increases with the number and shape complexity of objects.

Using octree shape representation, interference among objects can be detected by traversing their octrees in parallel [4]. Octrees can represent the details of object shapes to different degrees using different numbers of levels, but the computational cost is proportional only to the actual number of nodes visited. An octree with even limited depth is useful for approximately identifying object parts causing interference.

This paper proposes a method for detecting interference and potential collisions among objects with arbitrary motion (translation and rotation) in 3-D space using octree and polyhedral shape representations. Collisions are checked at discrete time instants, but collisions between time instants are not missed. Every collision detection involves all objects in the environment, and determines the pairs of faces that are likely to collide. The experimental results show the efficiency of the proposed method for non-convex objects.

*Currently with the Nara Institute of Science and Technology, Japan

†was visiting professor from the University of Illinois at Urbana-Champaign, USA.

2 Shape Representations for Collision Detection

2.1 Polyhedral Shape Representation

Polyhedral shape representation is one of the most common shape representations. Interference between polyhedral objects is detected by testing all combinations of faces and edges. The average time complexity for the test (for n objects) is $O(n^2 \cdot EF)$, where E, F are the number of edges and faces in the average object. Therefore, the computational cost is high for a large number of objects. However, if the number of faces or edges that are likely to collide can be restricted, the cost may be acceptable and the method may be useful.

2.2 Octree Shape Representation

The octree represents an object shape by recursively subdividing it into octants. A tree node is labeled black (white) if it is completely contained within an object (free space); otherwise, the node is labeled gray.

Using octree representation, interference among objects can be detected by traversing trees in parallel [4]. Since the time taken is proportional to the actual number of nodes visited, the average time complexity for detecting interferences among n objects is $O(Kn)$, where the average number of nodes in a tree is K.

2.3 Model Generation

One way to acquire the required octree and polyhedral models is to derive them from range data of an object as it moves. Another method is to maintain the polyhedral representation of an object under motion, which is easy, and to convert the polyhedral model into an octree. For a convex object, this can be easily achieved by testing the positional relations (interior, exterior or intersection) between each face and

1051-4651/94 $04.00 © 1994 IEEE

octree node. For example, the method in [5] generates the octree of non-convex objects by collecting the octree nodes corresponding to each polygonal face of an object using its quadtree representation of its planar projection.

The algorithm used here generates an octree from the polyhedral representation of general (non-convex) objects by using the point inclusion test based on the Jordan curve theorem for polygons [6]. The method is suitable for parallel implementation.

3 Hybrid Collision Detection

This section describes an efficient method, which first performs an approximate test to identify interfering objects in the entire workspace and then performs a more accurate test to identify the object parts causing interference/collisions by using octree and polyhedral representations, respectively.

3.1 Outline

Figure 1 shows the control flow used in this method. Suppose there are n objects in the workspace, and that for each object both the octree and polyhedral representations are known. Both of these representations for each object are updated periodically (at discrete time instants $\cdots$, t_{i-1}, t_i, t_{i+1}, $\cdots$) using the observed object motion parameters. To avoid collisions, future object positions and orientations are extrapolated using these motion parameters. Potential collisions are then checked using the extrapolated representations to detect the interference at each discrete time instant t_i, using the following steps.

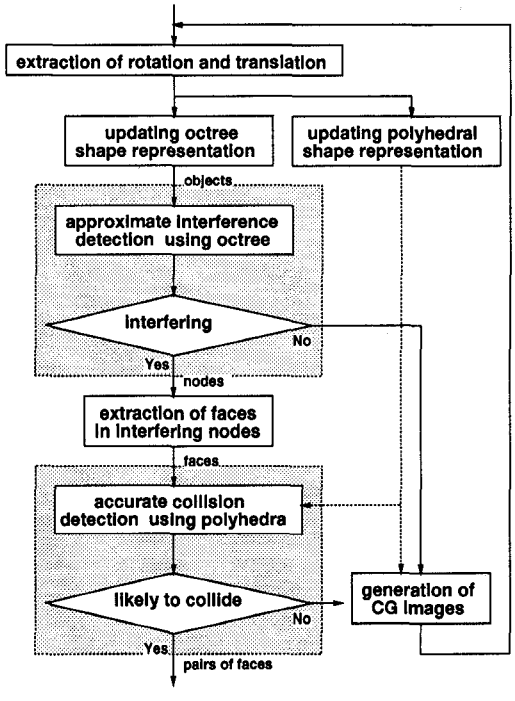


Figure 1: Proposed method for collision detection.

3.2 Procedure

It is assumed that the motion of each object during each time interval is small compared to the sampling interval. It is also assumed that the objects are rigid, and their motions between successive time instants are translation and rotation.

3.2.1 Updating Representations

Both the octree and polyhedral shape representations are updated periodically for each object using the observed object motion parameters. Note that updating polyhedral representations is relatively straightforward. An algorithm capable of updating the octree of a three-dimensional object for arbitrary rotation and translation is described in [7]. This algorithm moves each of the black leaf nodes and constructs subtrees corresponding to the displaced black nodes. It tests the intersections of moved cubes with octree tessellation. Since octree and polyhedral shape representations are updated independently for each object in this application, differences occur between the two representations. A 2-D example of translation is shown in Figure 2. An object located at the position shown in (a) moves toward the right. For the initial state in (a), nodes a and c are black. When a 1/4-voxel movement toward the right occurs, for example, a polygonal object (polyhedral object in 3-D) moves as in (b), while the octree nodes b and d continue to be white because their centers are outside the displaced black cubes a and c that have been moved by 1/4-voxel. When the motion is more than 1/2 voxel, node b or d becomes black.

Therefore, in the algorithm used here, a partially-occupied voxel is colored black if any part of its boundary is on or inside a displaced black cube. Though this reduces the computational efficiency since the number of black nodes increases, it is necessary so as not to miss possible collisions. An accelerated method that generates octrees from polyhedral shape representations for each processing cycle will lessen this problem.

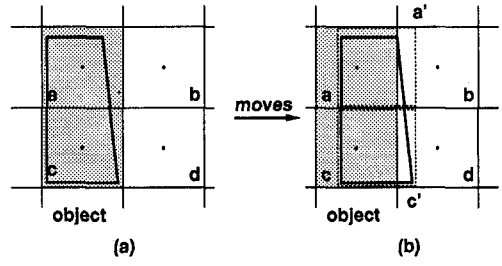


Figure 2: An example of white nodes occupied partially by an object

3.2.2 Approximate Interference Detection Using Octrees

Interference in the entire workspace is detected by traversing the objects' octrees in parallel. If black node exists whose corresponding node in another tree is also black, these nodes are considered to be interfering. The traversal is performed down to a predetermined lowest level. At this lowest level, for safety, any pair of corresponding gray nodes is considered to be interfering. After the traversal is finished, all interfering nodes and corresponding objects are identified. This determines the objects and their approximate parts that are likely to collide in the near future.

3.2.3 Extraction of Faces in Interfering Nodes

The faces of objects that intersect with their octree nodes are extracted (marked as interfering with other objects). For each such interfering node in an object's octree, the coordinates of the eight vertices of the corresponding cube C are substituted in the equation of the plane T of each face F of the interfering object. If all vertices do not lead to the same sign (positive or negative) for the value obtained after substitution, then a face F is judged to be possibly causing the interference detected by octree representation. To determine if the face actually does cause interference, the polygon S of the intersection between C and T is found. A simple two-dimensional interference detection is then done for S and F. If an intersection is found, F is labeled as interfering; otherwise, it is labeled as noninterfering.

3.2.4 Accurate Collision Detection Using Polyhedra

The faces identified above are checked for collisions. At any time instant t_i, the possibility of a collision between t_i and t_{i+1} is tested by considering the volume expected to be swept by each face during the interval $[t_i,\ t_{i+1}]$ (see Figure 3). This is how collisions between discrete time instants are avoided. To be conservative, collision is assumed if these volumes intersect even though such intersections are a necessary, but not sufficient, condition for the occurrence of collisions.

For each moving face A, the convex hulls $V_A^{t_i}$ of a set of vertex points of A^{t_i} (i.e. $a_0^{t_i}$, $a_1^{t_i}$, $a_2^{t_i}$,...) and $A^{t_{i+1}}$ (i.e. $a_0^{t_{i+1}}$, $a_1^{t_{i+1}}$, $a_2^{t_{i+1}}$,...) (chapter 3 in [8]) expected to be swept by face A during the interval $[t_i,\ t_{i+1}]$ are computed. For each face B^{t_i} with which intersection of A^{t_i} is to be tested during the interval $[t_i,\ t_{i+1}]$, the convex hulls $V_B^{t_i}$ of a set of vertex points of B^{t_i} and $B^{t_{i+1}}$ are computed. Here, face A and face B at time $t = t_i$ are specified by A^{t_i} and B^{t_i}, respectively.

Next, the intersection between $V_A^{t_i}$ and $V_B^{t_i}$ is tested. An intersection is detected by testing whether one of the following positional relationships of all combinations of faces and edges exists: both endpoints of an edge lie on the same side of the plane containing the face (Edge 1), an edge intersects the outside of the face plane (Edge 2), or an edge intersects the inside of the face plane (Edge 3). An intersection is detected in the case of Edge 3.

This identifies all pairs of faces that are expected to collide during the time interval $[t_i,\ t_{i+1}]$ by testing for collisions among faces extracted from each node of the octree in which interference has been found. This method is not efficient when the number of vertices of each face is large. In this case, a more efficient method (such as the Muller-Preparata method in chapter 7 of [8]) might be useful to test for intersection of convex polyhedra. Figure 3 shows the simplest case (triangles).

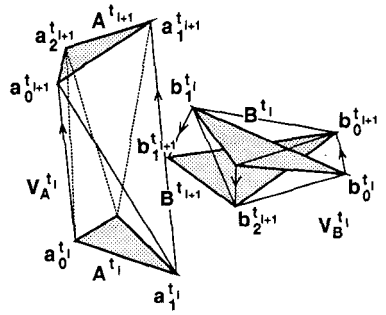


Figure 3: Collision detection between moving faces identified by octree representation as potentially colliding.

4 Experiments

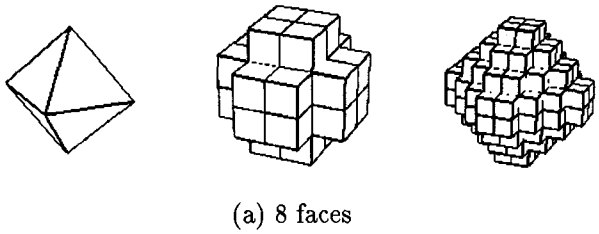
This section is concerned with the performance of the proposed collision detection algorithm and presents experimental results.

4.1 Performance Evaluation

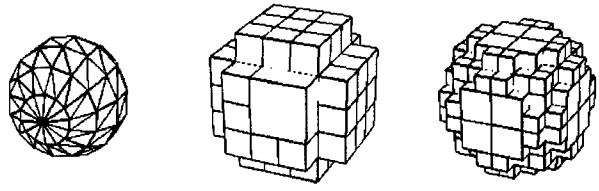
For performance evaluation, a sphere-like object represented by triangular patches was used. An octree shape representation of this object is shown in Figure 4. Spheres were selected because of their orientation invariance. The octree representation hierarchy is considered to have 4 and 5 levels in this figure, and the root node of the octree corresponds to the entire workspace.

4.1.1 Experiments with Two Moving Objects Having a Different Numbers of Faces

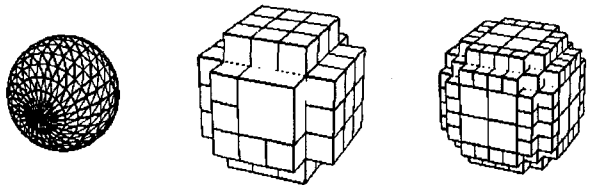
Interference and collision detection between two identical objects (spheres) represented by several kinds of polyhedral shape representations, each having a different number of planar patches, was tested. Each sphere has a corresponding octree shape representation as shown in Figure 4. The initial positions, directions and motions of the centers of the two objects (A and B) are shown in Table 1. Here, the units are equal to the length of the side of the smallest octree cube (or voxel), i.e. level 0 (depth $d = 4$). The workspace is divided into M^3 ($M = 2^4$) voxels at the lowest level,



(a) 8 faces



(b) 168 faces



(c) 728 faces

Figure 4: Examples of experimental objects represented using polyhedra (left column), 4-level octrees (center) and 5-level octrees (right column).

and the diameters of the objects are 3.8 voxels, with respect to 4-level octrees.

During every processing cycle, after the positions of the octree and polyhedral shape representations of each object had been updated, an approximate test was performed to identify interfering objects in the entire workspace using octree representation of object shapes. If any interfering nodes were found, the polyhedral representation of object shapes was used to more accurately identify any object parts causing interference and collisions. The computation time (CPU time) of each processing cycle was measured on a workstation (Silicon Graphics CRIMSON with 96M memory).

The results for two moving objects with 24, 80 and 168 triangular patches are shown in Figure 5. In the case of a collision between two spheres with 168 faces, no interference between octree nodes could be found until time $t = 6$ (*cycles*), Between times $t = 6$ and $t = 62$, interfering octree nodes were found but no faces could be detected in these nodes; therefore, the computational cost was relatively small. Once faces in interfering nodes were found at $t = 62$, more computation time was needed to test for collisions among them. Though faces in interfering nodes were found, they did not collide until $t = 116$. Finally, at $t = 116$, collisions among detected faces were found, and this experiment was terminated. At the last stage of collision detection, about 1.28 seconds was needed to identify 14 pairs of faces from 31 interfering octree nodes that were going to collide. Here, a test was done for collisions among the faces identified by each node of the octree for which interference had been found. Collisions were checked at discrete time instants, but collisions between time instants were not missed.

On the other hand, the computation time for collision detection without any constraints caused by octree shape representations, i.e., using only polyhedral shape representation, between two objects with 168 faces, was 41 seconds. The polyhedral method tests for collisions among all combinations of faces at every time instants; therefore, considerable computation is necessary throughout. For collision detection between spheres, constraints such as the distance between centers against the sum of diameters, or other parametric methods might be useful for eliminating the candidate faces. However, since these types of constraints are not always good for concave or complex objects, the proposed method was compared with the polyhedral representation collision detection algorithm, which does not use the above constraints.

The scale is very different for the results of each experiment shown in Figure 5; however, the shapes of the graphs are very similar. In Figure 6, the computation time for the last stage of the proposed collision detection method, which requires maximum computation, is compared with that of the conventional collision detection method using only polyhedral shape representation. The experimental results show the efficiency of the proposed method when the number of object faces increases.

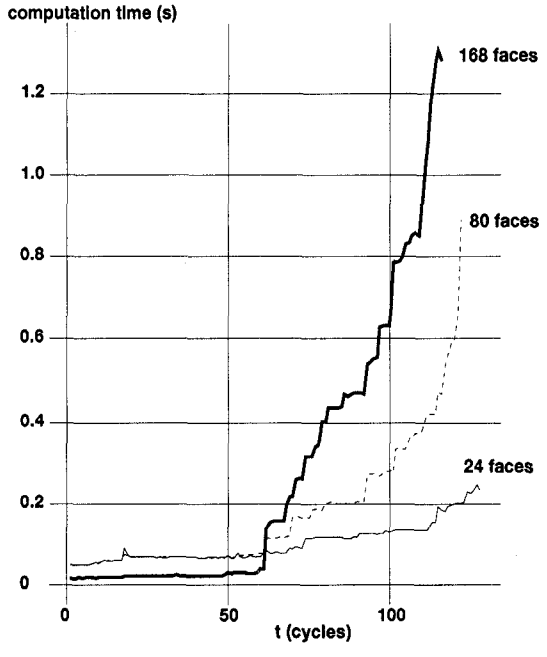


Figure 5: Computation time for each processing cycle of collision detection for two identical objects with 24, 80 and 168 faces.

Table 1: Initial positions, directions and motions given to the objects (unit: 4-level octree)

object	initial positions (*voxels*)	translation (*voxels/cycle*)	rotation (*degrees/cycle*)
A	(2.5, 2.5, 2.5)	(0.015, 0.0075, 0.01)	(0.0, 0.25 0.0)
B	(8.5, 6.5, 6.5)	(-0.015, -0.0075, -0.01)	(0.0, 0.25, 0.0)
C	(13.5, 13.5, 6.0)	(-0.0005, -0.0005, -0.0005)	(0.05, 0.1, 0.15)
D	(6.0, 2.5, 13.5)	(-0.0005, -0.0005, -0.0005)	(0.1 0.15, 0.05)
E	(2.5, 13.5, 2.5)	(0.0005, -0.0005, -0.0005)	(0.15, 0.05, 0.1)
F	(13.5, 8.0, 13.5)	(0.0005, -0.0005, -0.0005)	(-0.05, -0.1, -0.15)
G	(13.5, 2.5, 2.5)	(0.0005, -0.0005, -0.0005)	(-0.1, -0.15, -0.05)
H	(6.0, 13.5, 13.5)	(0.0005, -0.0005, -0.0005)	(-0.15, -0.05, -0.1)

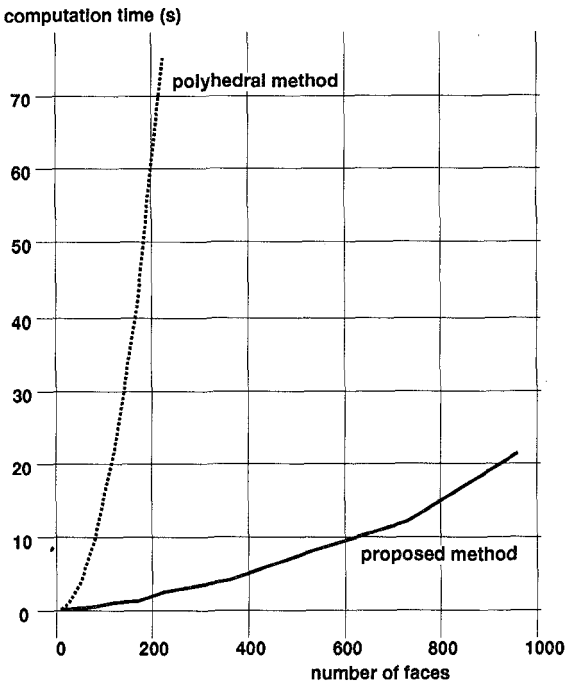


Figure 6: Computation time for each processing cycle of collision detection between two identical objects against the number of object faces

4.1.2 Experiments with Multiple Moving Objects

Several instances of an object were used to obtain a multiple moving object. The experimental object had 168 faces in a polyhedral representation, and the octree representation hierarchy had 4 levels. Constant translation and rotation comprise the motion of each object A, B, C, D, ...; and only object A and B collide. The initial position and motion of each object is listed in Table 1. The computation time for each processing cycle was measured on a workstation as described above.

In the results of experiments for various numbers of objects, the scale is very different for each experiment; however, the shapes of the graphs are quite similar. Figure 7 shows these results. In this figure, the computation time for the last stage which requires maximum computation is compared to that of the collision detection method using only polyhedral shape representation. The experimental results show the efficiency of the proposed method when the number of objects increases.

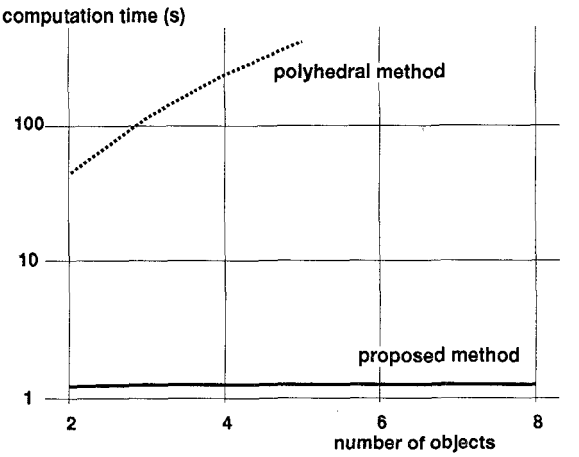


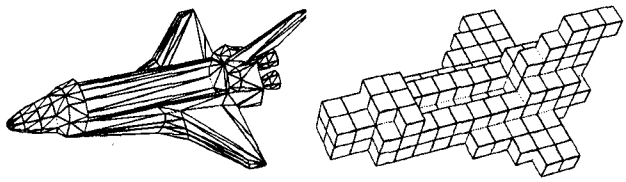
Figure 7: Computation time for each processing cycle of collision detection among multiple moving objects with 168 faces against the number of objects

4.2 Experiments with a Complex Object

The proposed algorithm was applied to a practical environment. A space shuttle represented by triangular patches and an octree shape representation of this object generated by the method of [6] shown in Figure 8 was used.

Interference and collisions among n ($n = 2, 3, 4, 5$) identical objects (space shuttles) were detected. The initial positions of the objects (from A to E) in the experimental workspace are shown in Figure 9. Constant translation and rotation movements were given to each object, and only object A and B collided.

Here, a level-5 octree was used. In this experiment, the faces of polyhedral shape representations are all triangles, and the octree shape representation of Figure 8 (b) was used by integrating its octree root node into the level-1 (depth 4) octree node whose root node corresponds to the entire space.



(a) polyhedra (528 faces) (b) octree

Figure 8: The space shuttle represented using (a) polyhedra and (b) octree representations.

The results from multiple moving objects (space shuttles) with 528 triangular patches were shown in Figure 10. Until time $t = 71$ (*cycles*), no interference could be found; therefore, the computation cost was relatively low. Once interfering nodes were found at $t = 71$, more computation time was needed to detect faces in the interfering nodes and to test for collisions among them. Though interfering nodes were found, the detected faces did not collide until $t = 123$. Finally, at $t = 123$, collisions among detected faces were found, and the experiment was terminated just before the collision shown in Figure 11. This Figure shows a snapshot from the viewpoint in Figure 9. In the last stage, about 1.2 seconds were needed to identify 14 pairs of faces from 62 interfering octree nodes that were going to collide. Every collision detection involved all objects in the environment, and determined the pairs of faces that were likely collide.

On the other hand, the computation time for collision detection for two space shuttles without using an octree, i.e., using only polyhedral shape representation, was 415 seconds. In an environment that included five moving objects, the computation for each time instant of the polyhedral method required 2470 seconds, while the proposed method required only 1.56 seconds for the last stage which requires maximum computation. In this case, the proposed method detects collisions in about 0.06% of the time required by the polyhedral method. Though the proposed method is efficient compared to the polyhedral method, this is slow for on-line (real time) monitoring and path planning. However, since the method can be easily implemented on a parallel processor, real-time collision detection in a 3-D environment containing multiple independently-moving complicated objects could be achieved.

5 Summary

This paper has presented an efficient method for detecting interference and potential collisions among multiple objects. Collisions are checked at discrete time instants, but collisions between time instants are not missed. Every collision detection involves all objects in the environment, and determines the pairs of faces that are likely collide. The experimental results have shown the efficiency of the proposed method, especially when there are multiple, complicated objects with arbitrary motion in an environment.

An accelerated method that generates octrees from polyhedral shape representation for each processing cycle will lessen the problem of updating octrees. This approach makes it possible to detect collisions among deformable objects.

It might be possible to make a more efficient algorithm by using a special octree ([9] or chapter 5 of [10]) which associates (using pointers) the black nodes of octrees with the faces of polyhedral shape representation that are contained inside of or intersect with them. Real-time collision detection in a 3-D environment could be achieved by implementing the proposed method on a parallel computer.

References

[1] Boyse, John W. Interference decision among solids and surfaces. *Communications of the ACM*, Vol. 22, No. 1, pp. 3–9, 1979.

[2] Canny, John. Collision decision for moving polyhedra. *IEEE Transactions on PAMI*, Vol. 8, No. 2, pp. 200–209, 1986.

[3] Kawabe, S., Okano, A., and Shimada, K. Collision detection among moving objects in simulation. *Robotics Research*, Vol. 4, pp. 489–496, 1988.

[4] Ahuja, N., Chien, R. T., and Bridwell, N. Interference detection and collision avoidance among three dimensional objects. In *International Conference on Artificial Intelligence*, pp. 44–48, 1980.

[5] Tamminen, Markku and Samet, Hanan. Efficient octree conversion by connectivity labeling. *Computer Graphics*, Vol. 18, No. 3, pp. 43–51, 1984.

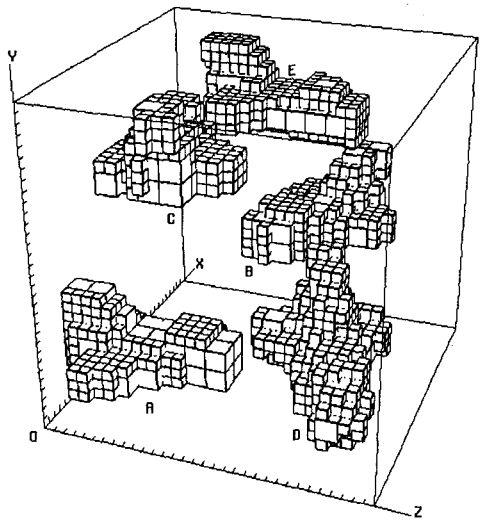
[6] Kitamura, Y., Bayle, M., Takemura, H., and Kishino, F. Generation of an octree from a polyhedral shape representation. In *IEICE Conference 1994*. D-604, 1994. (in Japanese).

[7] Weng, J. and Ahuja, N. Octree of objects in arbitrary motion: Representation and efficiency. *Computer Vision, Graphics, and Image Processing*, Vol. 39, No. 2, pp. 167–185, 1987.

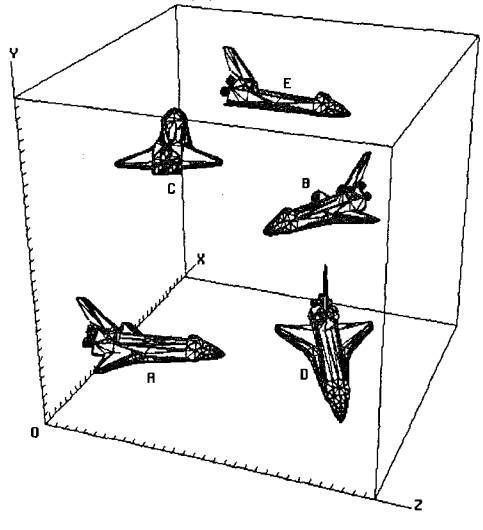
[8] Preparata, Franco P. and Shamos, Michael Ian. *Computational geometry, an introduction.* Springer-Verlag, 1988.

[9] Ayala, D., Brunet, P., Juan, R., and Navazo, I. Object representation by means of nonminimal division quadtrees and octrees. *ACM Transactions on Graphics*, Vol. 4, No. 1, pp. 41–59, 1985.

[10] Hanan Samet. *The design and analysis of spatial data structures.* Addison-Wesley, 1990.



(a) octree



(b) polyhedra

Figure 9: Experimental space including five objects (space shuttles) at initial positions.

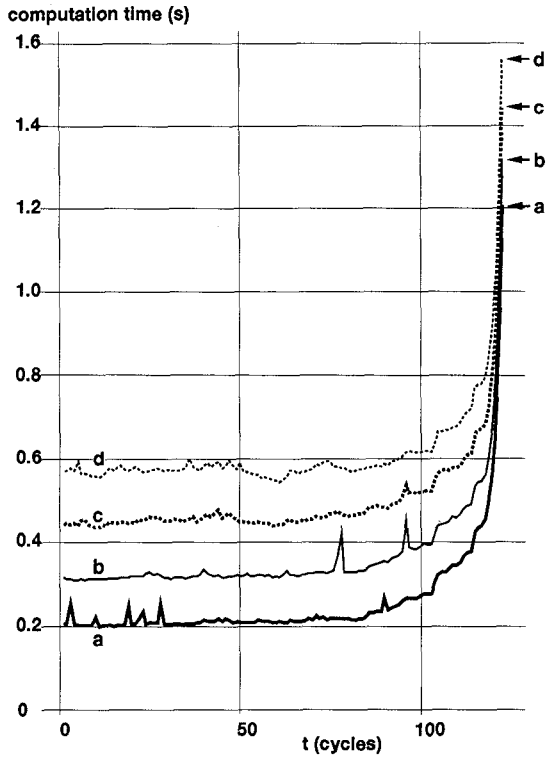


Figure 10: Computation time for each processing cycle of collision detection among objects (space shuttle) — a: two objects, b: three objects, c: four objects, d: five objects. Arrows indicate the end point of each graph.

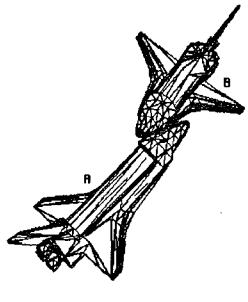


Figure 11: A snapshot of the experiment when collisions are detected.