Yoshifumi Kitamura*
Andrew Smith†
Haruo Takemura‡
and
Fumio Kishino*
ATR Communication Systems
Research Laboratories
2-2 Hikaridai, Seika-cho, Soraku-gun
Kyoto 617-02 Japan
kitamura@eie.eng.osaka–u.ac.jp

A Real-Time Algorithm for Accurate Collision Detection for Deformable Polyhedral Objects

Abstract

We propose an accurate collision detection algorithm for use in virtual reality applications. The algorithm works for three-dimensional graphical environments where multiple objects, represented as polyhedra (boundary representation), are undergoing arbitrary motion (translation and rotation). The algorithm can be used directly for both convex and concave objects and objects can be deformed (nonrigid) during motion. The algorithm works efficiently by first reducing the number of face pairs that need to be checked accurately for interference, by first localizing possible collision regions using bounding box and spatial subdivision techniques. Face pairs that remain after this pruning stage are then accurately checked for interference. The algorithm is efficient, simple to implement, and does not require any memory-intensive auxiliary data structures to be precomputed and updated. The performance of the proposed algorithm is compared directly against other existing algorithms, e.g., the separating plane algorithm, octree update method, and distance-based method. Results are given to show the efficiency of the proposed method in a general environment.

1 Introduction

Collision detection is one of the central problems in many application fields. In virtual environment research, for example, it is necessary to avoid interpenetration among objects and to accurately simulate several kinds of physical phenomena to enhance the reality of a virtual environment. For this purpose, it is necessary to detect the ''collisions'' among polyhedral objects in the environment. In addition, it is important to determine more precisely the colliding part of the object (e.g., colliding pair of faces and their normal vectors) to accurately simulate the physical phenomena. Furthermore, it is vital to be able to update the virtual environment at real-time rates to provide an impression of realism. Unfortunately, current collision detection algorithms, if used, are an enormous bottleneck and make real-time update impossible (Pentland, 1990; Hahn, 1988).

The difficulty in detecting the collision of polyhedral objects can be seen by examining the basic, naive way of performing it. The basic method works by performing static intersection tests at discrete time instants; the time interval between tests is assumed small enough that collisions are not missed. Then,

* Currently with the Graduate School of Engineering, Osaka University.
† Currently in the Ph.D. program at the Department of Computer Science, Yale University.
‡ Currently with the Nara Institute of Science and Technology, Japan.

Presence, *Vol. 7, No. 1, February 1998,* 36–52

interference among polyhedral objects at a time instant is detected by testing all combinations of faces and edges for the presence of an edge of one object piercing the face of another object; if such an edge-face pair exists then there is a collision (Boyse, 1979). The average time complexity for this test (for n objects) is $O(n^2EF)$, where E and F are the number of edges and faces in the average object. As can be seen from this complexity figure, the problem lies in the necessity of having to perform such a large number of computationally expensive intersection tests at every time instant, where the number of such tests increases quadratically as the number and complexity of objects increase. For anything more than a simple world with a few objects of a few hundred faces each, this method is untenable in terms of maintaining real-time performance.

The main problem with the basic, naive collision detection method is that it requires such a large number of computationally expensive edge-face intersection checks. In an actual virtual world, the number of edge-face pairs that intersect at any time instant is a small percentage of the total number of possible pairs (in fact, much of the time there are no intersections). Thus, it is desirable to have a collision detection algorithm that checks the number of edge-face pairs proportional to the number that actually intersect. In this paper, we present an algorithm that does this and can be used for general (i.e., both convex and concave), deformable polyhedral objects undergoing arbitrary motion.

The next section discusses other research efforts toward efficient collision detection. After that, the details of our collision detection algorithm are described. Next, experiments carried out using our algorithm are described, and performance results, showing the efficiency of the approach, are given. The performance of the proposed algorithm is compared directly against other existing algorithms.

2 Efficient Collision Detection Approaches

There is much literature devoted to efficient collision detection approaches and this section discusses some of them. The first subsection simply describes other approaches to efficient collision detection. The last two subsections evaluate these approaches, describing problems preventing them from being entirely suitable for practical, large-scale virtual environments and how our algorithm addresses these problems.

2.1 Related Collision Detection Research

A lot of research on collision detection for polyhedra aims to drastically reduce the number of edge-face pairs that need to be checked for intersection. A common first step in many collision detection routines is an approximate bounding region (usually an axis-aligned / box or a sphere) overlap test to quickly eliminate many objects as not interfering. An extension of this idea is to use a hierarchy of bounding regions to localize collision regions quickly (Hahn, 1988). Gottschalk, Lin, and Manocha (1996) use precomputed OBBTrees (oriented bounding box trees). Related methods use octrees and voxel sets. Garcia-Alonso, Serrano, and Flaquer (1994) store a voxel data structure with each object, along with the pointers (from the voxels to polyhedra faces) that intersect them. Collision is localized by testing for intersection of voxels between two objects. Kitamura, Takemura, and Kishino (1994) store an octree for each object and, at each time instant, check for interference of updated octrees; face pairs from the inside of interfering octree nodes are then checked for collision. Other voxel and octree methods have been proposed by Moore and Wilhelms (1988), Turk (1989), Zyda et al. (1993), Shaffer and Herb (1992), and Hayward (1986).

Another method of collision detection involves keeping track of the distance between each pair of objects in the world; if the distance between a pair goes below some small threshold then the pair has collided. A noteworthy use of this idea for the collision detection of rigid, convex objects was proposed by Lin and Canny (1991), Lin, Manocha, and Canny (1994), and Cohen et al. (1995). In this approach, the coherence of objects between time instants (i.e., object positions change only slightly) and the property of convex polyhedra are used to detect collisions among objects in a roughly constant

time per object pair. Other researchers who have used this distance-based approach include Gilbert, Johnson, and Keerth (1988), Quinlan (1994), and Chung and Wang (1996).

Briefly, some other approaches to collision detection are as follows. Bouma and Vanecek (1991) use a data structure called a ''BRep-Index'' (an extension of the well-known BSPTree) for quick spatial access of a polyhedron in order to localize contact regions between two objects. Baraff (1990) finds separating planes for pairs of objects; using object coherence, these separating planes are cached and then checked at succeeding time instants to yield a quick reply of noncollision most of the time. Shinya and Forgue (1991) use ideas from the z-buffer visible surface algorithm to perform interference detection through rasterization. Vanecek (1994) uses backface culling to prevent roughly half of the faces of objects from being checked for detailed interference; the basic idea is that polygons of a moving object that do not face in the general direction of motion cannot possibly collide. Foisy, Hayward, and Aubry (1990) use a scheduling scheme whereby object pairs are sorted by distance and only close objects are checked at each time instance. Hubbard (1993) uses four-dimensional space-time bounds to determine the earliest time that each pair of objects could collide and does not check the pair until then. Pentland (1990) models objects as superquadrics and shows how collision detection can be done efficiently using the inside / outside function of a superquadric. For coarse collision detection, Fairchild, Poston, and Bricken (1994) store bounding regions of objects in a stack of 2D structures similar to quadtrees (to reduce memory use) and uses only bit manipulations to add or delete objects to / from this stack (to reduce computation).

Finally, our algorithm uses ideas from methods for localized set operations on polyhedra (Mantyla and Tamminen, 1983; Fujimura and Kunii, 1985). These methods attempt to efficiently perform set operations, such as intersection, union, etc., on polyhedra by localizing regions where faces use spatial subdivision techniques; a set operation for a face then only needs to be done against the other faces inside the region that the face is in. As a particular example, the idea of intersecting faces with overlap regions of bounding boxes in order to localize the interference region of two objects was first described in Maruyama (1972), and we use this idea effectively in our algorithm.

2.2 Evaluation

We evaluate the above algorithms on the basis of four properties a collision detection algorithm needs for effective use in a practical, large-scale virtual environment inhabited by humans. The properties are the ability to handle deformable (nonrigid) objects, the ability to handle concave objects, the use of an inexcessive amount of memory for storing auxiliary data structures, and a better than $O(n^2)$ complexity for n objects in the world. None of the algorithms surveyed in the previous subsection have all four properties and some do not even have one of them. Our algorithm can satisfy all four of these properties.

2.2.1 Deformable Objects. In a virtual environment inhabited by humans, it is very important to be able to perform collision detection for objects that deform during motion. For example, in physical-based simulations, forces between colliding objects are determined and the colliding objects are then deformed based on these forces. In general, a user should be allowed to deform objects in a virtual environment, necessitating collision detection for deformable objects. Many of the above algorithms require precomputation and computationally expensive updating of auxiliary data structures (e.g., octrees, voxel sets, OBBTrees, BRep-indices) for each object. This limits their usefulness because it means that objects are essentially limited to being rigid; this is because when an object deforms, its auxiliary data structures must be recomputed and this is usually an expensive operation. Our collision detection algorithm handles deformable objects.

2.2.2 Auxiliary Data Structures. In addition to being expensive to recompute, auxiliary data structures can take up considerable memory when stored for each object. This limits the number of objects for some algorithms. Our algorithm does not require any auxiliary

data structures beyond simple bounding boxes and arrays.

2.2.3 Concave Objects. Another problem is that some of the above collision detection algorithms require objects to be convex (Baraff, 1990; Lin and Canny, 1991; Lin, Manocha, and Canny, 1994; Cohen et al., 1995; Gilbert, Johnson, and Keerth, 1988; Quinlan, 1994; Chung and Wang, 1996). However, it is clear that most objects of interest in the real world are concave and a virtual environment, to be useful, should allow concave objects. To solve this problem, the above authors argue that a concave object can be modeled as a collection of convex pieces. While this can in fact be done for any concave object, it adds many fictitious elements (i.e., vertices, edges, etc.) to an object. In addition, breaking a concave object up into convex pieces means that the one object becomes many objects; unfortunately, this greatly worsens the complexity problem described in the next section (because each convex piece of the concave object must be treated as a separate object for the purpose of collision detection). Most importantly, however, any algorithm that requires objects to be convex or unions of convex pieces cannot be used to detect collisions for deformable objects; this is because, in general, deformations of an object easily lead to concavities. Our algorithm deals directly with concave objects in the same way as convex ones, with no extra computational overhead.

2.2.4 Complexity. The $O(n^2)$ complexity problem becomes apparent for large-scale virtual environments. Pentland (1990) discusses problems due to computational complexity in computer-simulated graphical environments and notes that collision detection is one such problem for which, in order to simulate realistically complex worlds, algorithms that scale linearly or better with problem size are needed. To understand the problem concretely, consider a collision detection algorithm that takes 1 millisecond per pair of objects. While for very small environments this algorithm is extremely fast, the algorithm is impractical for large-scale environments. For example, for an environment with just 50 objects, 1225 pairwise checks between objects must be done, taking more than a second of computation; in this example, real-time performance cannot be maintained for the environment, should it have more than 14 objects (being able to compute something in 100 milliseconds or less is considered to be real-time performance) (Card, Moran, and Newell, 1983). All of the distance-based approaches (Lin, Manocha, and Canny, 1994; Gilbert, Johnson, and Keerth, 1988; Quinlan, 1994) and many of the others (Baraff, 1990; Pentland, 1990; Garcia-Alonso, Serrano, and Flaquer, 1994) suffer from this complexity problem. In our experiments, we did use a bounding box test among objects, which is $O(n^2)$ for n objects. The bounding box test between two objects was found to be extremely fast and thus should not become a bottleneck unless there are many objects in the environment; for such an environment, the problem can be easily solved by using a bounding box check with better complexity (Kirk [1992] describes such a method) or by skipping the bounding box stage altogether and going directly to the face octree spatial subdivision stage, which is $O(n^2)$ for n objects.

2.2.5 Other. A few other minor problems with the surveyed algorithms are as follows. Some of these algorithms (Turk, 1989; Pentland, 1990) cannot be used for polyhedra, and this limits their usefulness for current graphical applications where polyhedra dominate as the object representation. Kitamura, Yee, and Kishino (submitted) describe how accurate collision detection—to identify exactly which objects (i.e., their faces) are interfering—is useful for manipulation aid in a virtual environment. Some of the algorithms do not provide accurate collision detection among objects (Shinya and Forgue, 1991; Hubbard, 1993; Foisy, Hayward, and Aubry, 1990; Fairchild, Poston, and Bricken, 1994). While most of the algorithms described above are clearly improvements over the basic, naive collision detection algorithm, none of them provide a solution to the problem that is as general, efficient, and simple as ours.

2.3 Proposed Algorithm

Our proposed algorithm is an extension of the methods for localized set operations for use in collision detection. In particular, we extend the ideas in Mantyla

and Tamminen (1983), Fujimura and Kunii (1985), and Maruyama (1972) to a world with multiple objects; these papers describe algorithms for two objects but never precisely explain how to extend their algorithms efficiently to handle multiple objects (thus, direct use of these algorithms requires $O(n^2)$ complexity for n objects). In addition, these algorithms, in testing for intersection between a face and an axis-aligned box (while performing spatial subdivision), advocate using an approximate test between the bounding box of the face and the axis-aligned box; however, in developing our algorithm we found that using the exact intersection test described in Section 3.7 gave better performance (because it reduces the number of edge-face pairs even more, without much of an added computational cost). Also, these algorithms are used to perform static set operations; we show how they can be used for collision detection in a dynamic environment with multiple moving objects. Finally, we provide empirical evidence to show the efficiency of the proposed algorithm. The next section describes the details of our proposed collision detection algorithm.

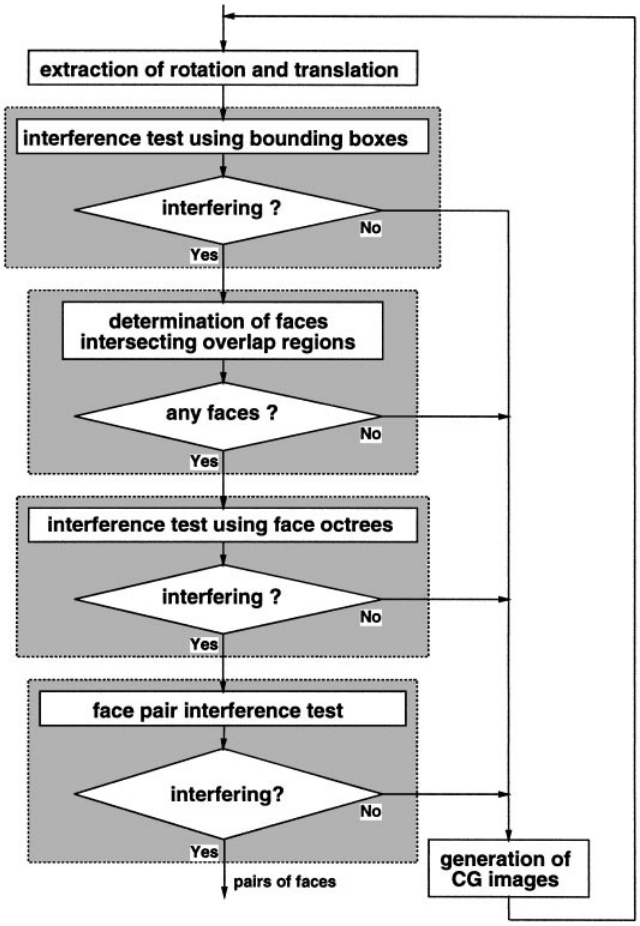

Figure 1. *Control flow of collision detection.*

3 Collision Detection Algorithm

3.1 Assumptions

All objects in the world are modeled as polyhedra (boundary representation). The faces of a polyhedron are assumed to be triangular patches without any loss of generality in the range of representations. Objects can be concave or convex. These objects undergo motion that is not predetermined (e.g., a user can move his graphical hand in a sequence of nonpredetermined, jerky motions); object motion can be both translational and rotational. Objects can be deformed during motion. Given this kind of environment, the goal is to be able to detect the colliding objects in the world and, in particular, the face pairs between objects that are interfering. Collision will be checked for all objects at discrete time instants (i.e., at each time instant the new positions of the objects will be determined, and collision will be checked for at that time instant *before* the computer graphic images of the objects are drawn to the screen). It is assumed that the speed of each object is sufficiently slow compared with the sampling interval so that collisions are not missed. Finally, it is assumed that there is a large cube that completely bounds the world (i.e., that all objects will stay inside of this cube); let the side length of this cube be L.

3.2 Outline of the Method

Figure 1 shows the control flow of our method. Suppose there are n objects in the workspace. The bounding boxes for each object are updated periodically (at discrete time instants . . . t_{i-1}, t_i, t_{i+1}, . . .) using observed object motion parameters. Updated bounding boxes are checked for interference. For each object with an interfering bounding box, all overlap regions of the

object's bounding box with those of other objects are determined. Next, for each such object all faces of the object are checked to see if they intersect with the overlap regions of the object; a list of the object's faces that intersect one or more of the overlap regions is stored. Then, if there is a list of faces for more than one object, a face octree (i.e., an octree where a node is black if and only if it intersects faces) is built for the remaining faces (for all objects' face lists, together), where the root node is the world cube of side length L and the face octree is built to some user-specified resolution. Finally, for each pair of faces that are from separate objects and that intersect the same face octree voxel (i.e., the smallest resolution cube) it is determined whether the faces intersect each other in three-dimensional space. In this way, all interfering face pairs are found. Note that the intersection of faces with overlap regions and face octree stages repeatedly test for intersection of a face with an axis-aligned box; thus, we describe an efficient algorithm for testing this intersection.

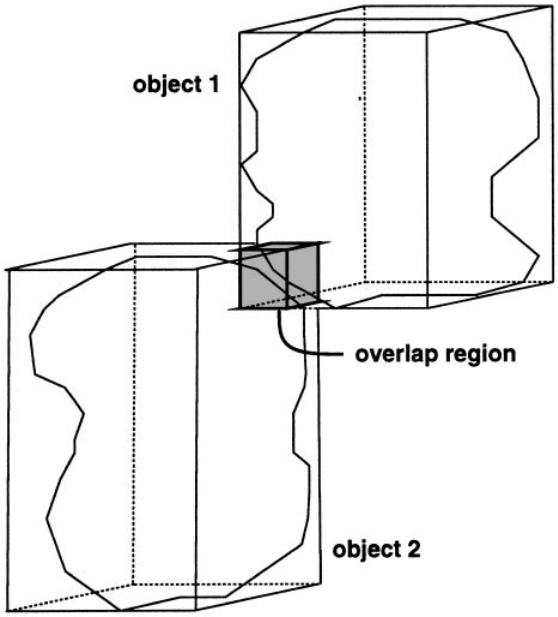

Figure 2. *An overlap region.*

3.3 Approximate Interference Detection Using Bounding Boxes

At every time instant, axis-aligned bounding boxes are computed for all objects, and all pairs of objects are compared for overlap of their bounding boxes. For each pair of objects whose bounding boxes overlap, the intersection between the two bounding boxes is determined (called an overlap region, as shown in Figure 2) and put into a list of overlap regions for each of the two objects. The overlap regions are passed to the next step.

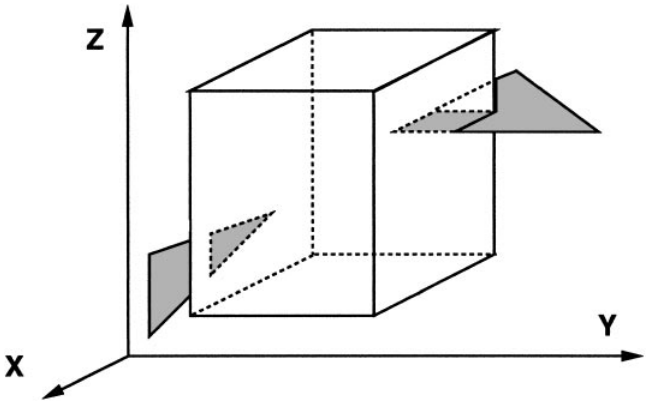

Figure 3. *Faces intersecting the overlap region.*

3.4 Determination of Faces Intersecting Overlap Regions

For every object that has a list of overlap regions, all faces of the object are compared for intersection with the overlap regions. Once a face of an object is determined to be intersecting with at least one overlap region it is placed in a face checklist for the object. If there are face checklists for two or more objects, then these are passed on to the next stage. Figure 3 shows an example of faces intersecting an overlap region.

3.5 Face Octree Spatial Subdivision Stage

A face octree is built down to a user-specified resolution for the remaining faces starting from the world cube of side length L as the root. To minimize computation, only as much of the face octree as is necessary for collision detection is built; in particular, a parent node is subdivided into its eight children only if it contains faces from two or more objects, and only the faces found to

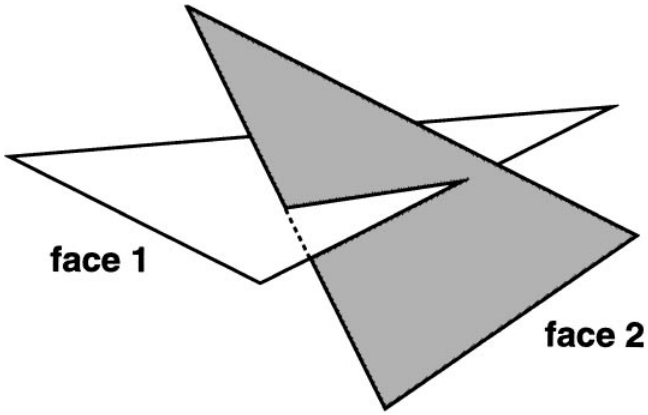

Figure 4. *Face pair intersection test.*

intersect the parent node are tested for intersection with the child nodes. Also, there is no condensation of the face octree (i.e., eight black child nodes are not erased and replaced by their single, black parent node). If there are voxels in the face octree, then in each voxel there are faces from two or more objects. For each voxel, all possible pairs of faces, where the faces are from different objects, are determined and put into a face pair checklist. However, a face pair is only put into this face pair checklist if it was not previously put there by examination of another voxel. The face pair checklist is then passed to the next stage. Note that it is not necessary to allocate memory and actually build a face octree; faces can simply be checked for intersection with the standard cubes of an octree and checked recursively for lower-level cubes (thus no memory, beyond the small amount used by the stack during recursion, is necessary for storing octrees). Also note that a face octree is built for only a very small portion of all the faces; the previous stage eliminates most faces as not interfering.

3.6 Face Pair Interference Check

A pair of faces is checked for intersection at a time instant as follows (see Figure 4). First, the bounding boxes of the faces are computed and checked for overlap; if there is no overlap in the bounding boxes then the faces do not intersect. Otherwise, the plane equation of the face plane of the first face is computed and the vertices of the second face are evaluated in this equation; if all vertices lead to the same sign (+ or −) then the second face is completely on one side of the face plane of the first face, and thus there is no intersection. The plane equation of the face plane of the second face is then determined, and the vertices of the first face are evaluated in it in the same way. If neither face is found to be completely on one side of the face plane of the other face, then more detailed checks are done as follows. The point of intersection of each edge of face 1 with the face plane of face 2 is found and checked to see if it is inside face 2 (a three-dimensional point-in-polygon check—the method used is described in Arvo [1991]); if the point is inside the face then the two faces intersect. The case where an edge and face plane are coplanar is handled by projecting the edge and face onto the two-dimensional coordinate axis most parallel to the face plane and performing a two-dimensional intersection check between the projected face and edge. In the same way, the edges of face 2 are checked for intersection with face 1. If no edges of either face are found to intersect the other face, then the two faces do not intersect.

3.7 Efficient Triangular Patch and Axis-Aligned Box Intersection Determination

To determine whether or not a triangular patch intersects with an axis-aligned box, we perform clipping against four of the face planes of the faces that comprise the box; the four face planes are the maximum and minimum extents of two of the three dimensions x, y, and z (e.g., in our implementation we arbitrarily chose to clip against the maximum and minimum x extents and the maximum and minimum y extents). For the final dimension, it is only necessary to check whether or not the remaining vertices of the clipped triangular patch are either all greater than the maximum extent or all less than the minimum extent. If either case is true then there is no intersection; otherwise there is intersection. In addition, it is often not even necessary to clip against the four planes. When clipping, whenever the intersection point of a segment with the current face plane is calculated, this point can be quickly checked to see if it is inside of the face of the face plane; if it is inside, then the

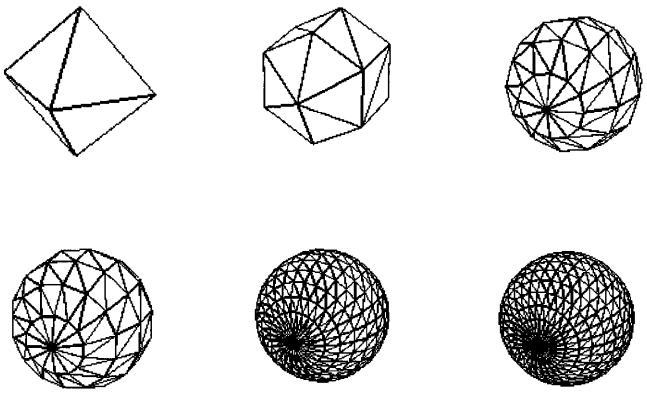

Figure 5. *Examples of experimental objects (standardized spheres with different numbers of faces).*

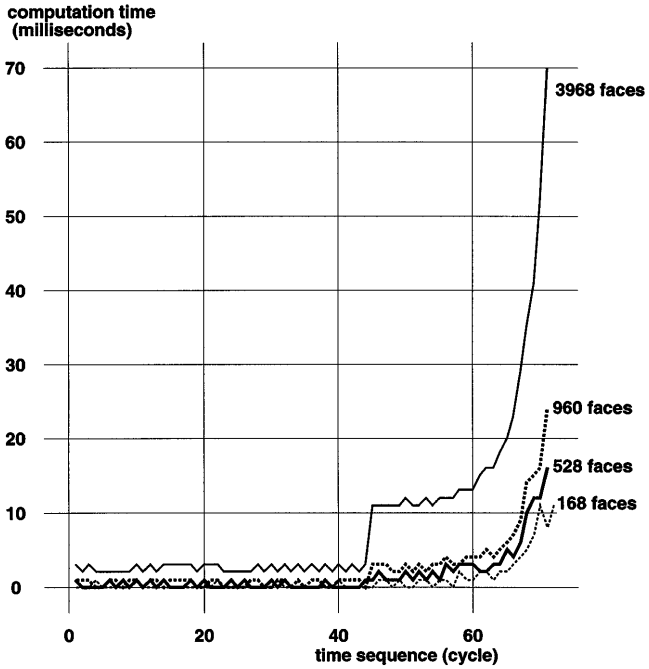

Figure 6. *Computation time for each processing cycle for two identical sphere objects with 168, 528, 960, and 3968 faces.*

triangular patch and box intersect and no more computation need be done. Finally, before performing any clipping at all, two quick tests are done. As a first step, a quick overlap check between the bounding box of the triangular patch and the axis-aligned box can be done to quickly determine nonintersection in many cases. Second, the three vertices of the triangular patch can be checked to see if one of them is inside of the axis-aligned box; if so, then the triangular patch and axis-aligned box intersect.

4 Performance of the Proposed Method

The algorithm and an experimental environment were implemented and run on a Silicon Graphics Indigo[2] (with an R4400 / 150 Mhz processor), with the exception of the experiment in Section 4.3; experiments were done to determine the efficiency of the proposed algorithm. In all experiments described in this section, face octrees were built to resolution level 6.

4.1 Standardized Objects

For performance evaluation, spherelike objects approximated by differing numbers of triangular patches were used; spheres were selected for testing because of their orientation invariance. Figure 5 shows some of the spheres that were used in the experiments. The basic experiment done was to have two identical sphere objects start at different (nonpenetrating) positions and have them move toward each other (with both translation and rotation motion) until they collided. This basic experiment was done with sphere objects having 8, 10, 24, 48, 54, 80, 120, 168, 224, 360, 528, 728, 960, and 3968 faces. Figure 6 shows the computation time (which includes updating object positions, but excludes rendering graphics) required at each processing cycle from $t = 1$ (cycle), when there is no interference, until $t = 72$ (cycle), when faces from the two sphere objects are found to be intersecting, for four of the experimental sphere objects; at the last cycle, 70, 24, 16, and 11 milliseconds of computation are required to determine the colliding faces for the spheres with 3968, 960, 528, and 168 faces. Finally, Figure 7 shows the computation required at the last stage (i.e., when faces from the two objects are found to be interfering—this requires maximum computation and is the true measure of the efficiency of a collision detection algorithm) of the pro-

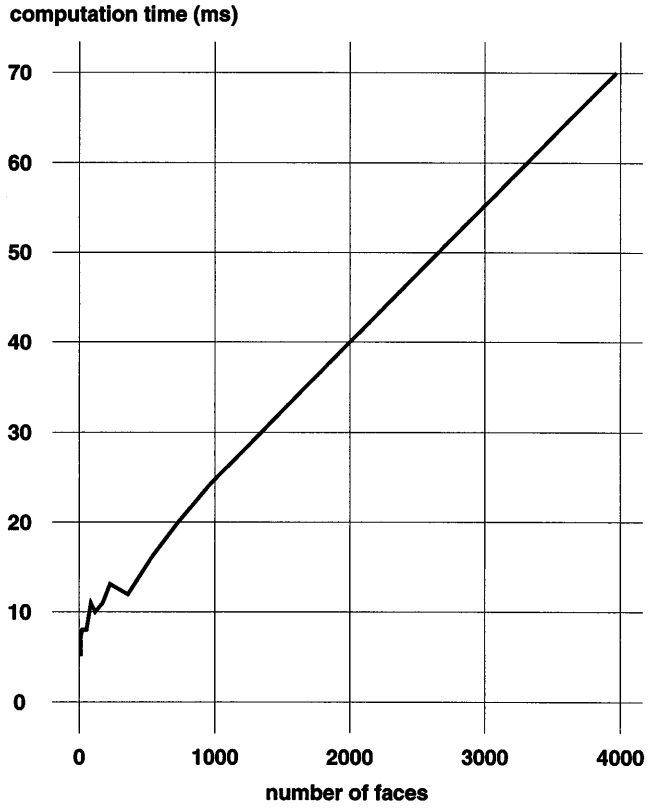

Figure 7. *Computation time at the last stage of the proposed collision detection between two identical sphere objects against the number of planar patches of the objects.*

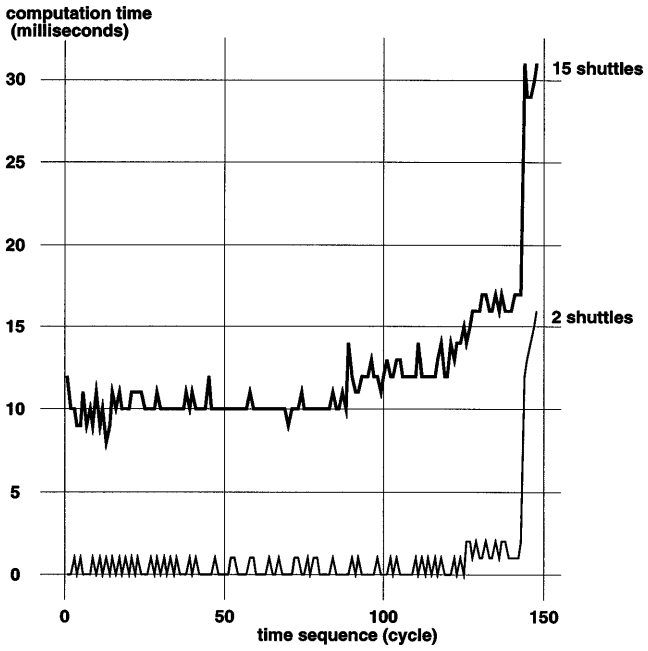

Figure 8. *Computation time at each processing cycle for 15 and for 2 space shuttle objects—collision between two objects is detected at the last cycle.*

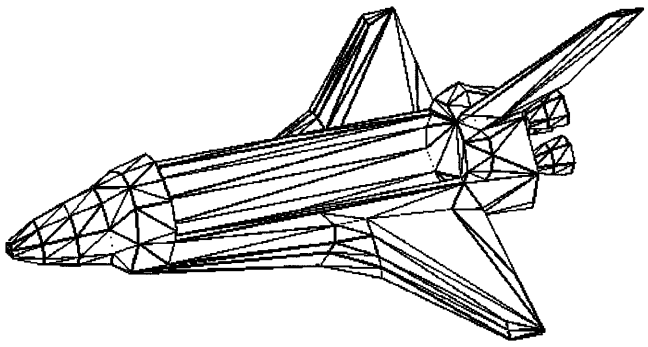

Figure 9. *The space shuttle experimental object (528 triangular patches).*

posed collision detection algorithm between two sphere objects against the number of triangular patches of the sphere objects.

4.2 Multiple General Objects

An experiment was also done with multiple general (i.e., concave, as occur in the real world) objects (see Figure 8). Specifically, 15 identical objects (space shuttles with 528 faces—see Figure 9) were moved (both translation and rotation motion) in the test environment for many processing cycles and the computation time required at each cycle to perform collision detection was measured. At every cycle, many objects' bounding boxes were overlapping; thus, many triangular patches had to be tested for intersection with overlap regions at every cycle. At the last cycle of the test, faces from two objects were found to be interfering, taking 31 milliseconds of computation. Figure 8 shows the results of this experiment. Also in this figure, to provide a basis for comparison, are the results for when only the two interfering space shuttle objects were in the test environment; here, the last step, where faces were determined to be colliding, required 16 milliseconds of computation.

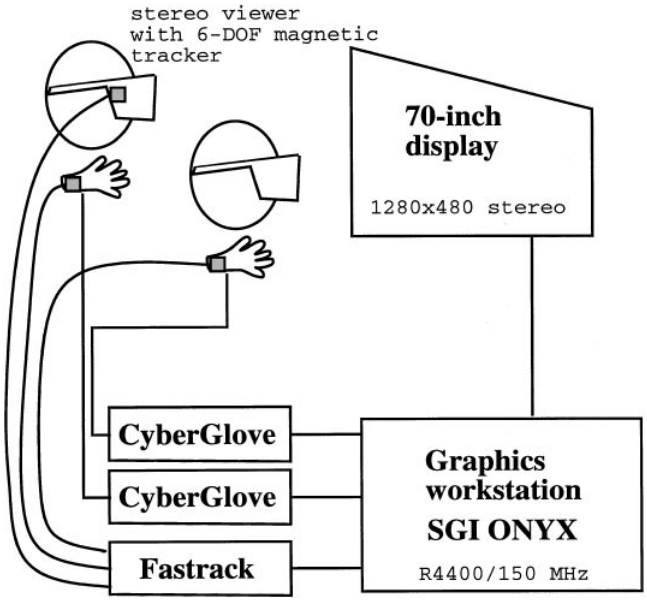

Figure 10. *Hardware configuration for interactive experiments.*

4.3 Multiple Deformable Objects in an Interactive Environment

Two users participated in the interactive experiment (Kitamura and Kishino, 1996). Both wore glasses and gloves with magnetic trackers and manipulated objects created by stereoscopic computer graphic images in the environment as shown in Figure 10. Figure 11 shows two snapshots from the experiment. The yellow hand could grasp the virtual objects directly and change the motions of the objects, while the orange hand could change the motions of the objects by touching them. Here, the detected pairs of faces are shown by their color change to red. The motions of collided objects are changed according to the normal vectors of the face pairs and their original motion vectors. The objects are also reflected at the boundary plane of the environment, meaning they continuously moved in the defined environment. The number of faces of Venus, the space shuttle, and each hand was 1816, 528, and 960, respectively. Here, a sphere and the hands deformed during motion. Even though a simple model for reflection was used in this experiment, the response after collisions are detected should be considered according to the application.

Figure 12 shows the computation time for only the collision detection routine required at each processing cycle during about 800 cycles with some random conditions. During the cycles without any collisions, the algorithm took negligible (rarely more than 5 milliseconds) computation time. It increased only when collisions were found. The average computation time for a cycle throughout this experiment was 11.5 milliseconds, as shown in Figure 12. The method detects colliding pairs of faces quite efficiently in a complicated environment; therefore, it is possible to design a virtual object manipulation aid using dynamic constraints among faces that provides the user with a natural impression of motion (Kitamura, Yee, and Kishino, submitted).

4.4 Application to a Virtual Cooperative Workspace

The proposed method was applied to a ‘‘virtual space teleconferencing’’ system. This system offers an environment for teleconferencing with realistic sensations, in which participants at different sites can engage in a conference with the sensation of sharing the same space (Miyasato, Kishino, and Terashima, 1995). It creates an image of a virtual conference room and the remotely located conference participants using computer graphics in real time. The users can work cooperatively in the virtual workspace (Takemura and Kishino, 1992).

An example of virtual cooperative work is shown on the front cover. It is a snapshot of constructing a ‘‘mikoshi’’ (a portable shrine). In this environment, multiple complicated objects are independently manipulated by different participants, with the proposed collision detection method providing users with sophisticated feedback. The detected colliding pairs of faces are shown by their color change to blue, and collision sound is also generated. Though the mikoshi has 5263 faces, it could present such feedback in real time (Miyasato, Ohya, and Kishino, 1996 [video]).

5 Comparison Against Existing Algorithms

The performance of the proposed algorithm is compared directly against three other existing algo-

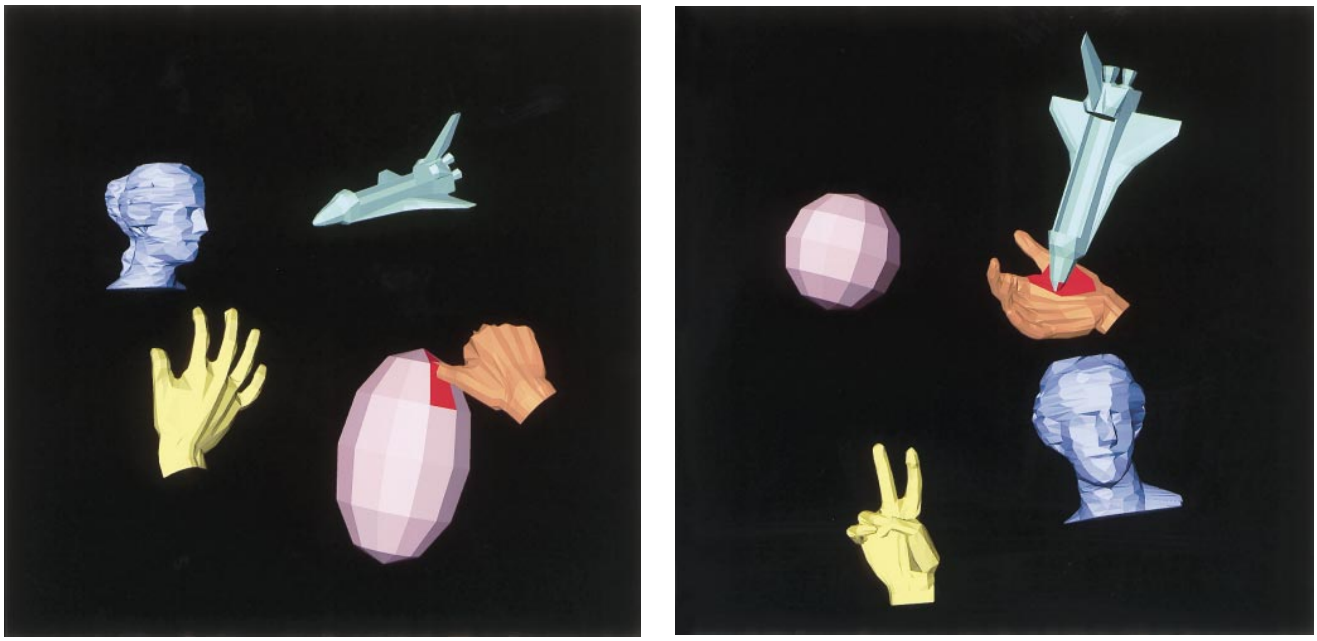

Figure 11. *Two snapshots from the interactive environment, which includes multiple deformable objects.*

rithms. In general, it is difficult to make such direct comparisons because authors of collision detection papers do not normally give out the code that they used to get their experimental results. Fortunately, however, we found the C language code for the first existing collision detection algorithm in Heckbert 1994, the second existing algorithm is a slight modification of the algorithm proposed in this paper, and the library for the third one is available on the internet.

5.1 Separating Plane Algorithm

The first algorithm is based on the separating plane algorithm (Gilbert, Johnson, and Keerth, 1988; Baraff, 1990). This algorithm can only be used for convex, rigid objects and it does not return the list of face pairs that are interfering, as ours does. The details of this algorithm are given in Heckbert 1994. However, briefly, the algorithm works by initially finding a separating plane between each pair of objects. A separating plane is found for two objects by finding the two closest vertices on the objects (using the method in Gilbert, Johnson, and Keerth, 1988); the vector between these two points is the normal vector of the plane, and the plane passes

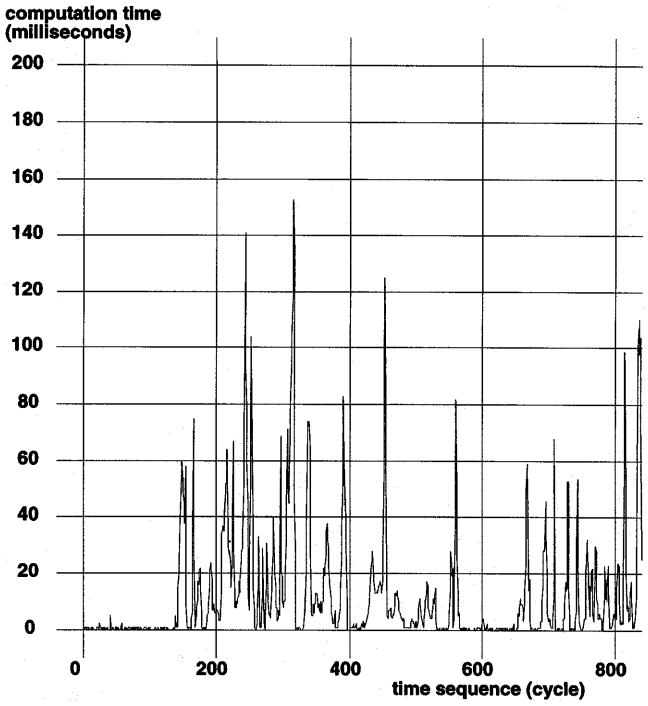

Figure 12. *Computation time at each processing cycle for the interactive environment.*

through one of the two points. Separating planes are cached between time instants and the previous time instant's separating plane is checked at the current time instant to see if it still separates the two objects; if it no longer separates them, an attempt is made to find a new separating plane, which is then cached. If no new separating plane can be found then there is collision. Note that the complexity for this test (for n objects) is $O(n^2)$.

We compared our proposed algorithm against this algorithm for environments containing differing numbers of the same sphere objects (528 triangular patches). In particular, we tested both algorithms in environments with 10, 20, 30, and 40 moving sphere objects; at the last cycle of the tests two of the sphere objects collided. For 10 sphere objects, our algorithm performed roughly the same as the separating plane algorithm; in particular, our algorithm required 16 milliseconds of computation at the last cycle, while the separating plane algorithm required approximately 10 milliseconds per cycle. However, for 20, 30, and 40 objects our algorithm performed better. In particular, for 20, 30, and 40 objects, our algorithm required 21, 22, and 41 milliseconds at the last cycle; against this, the separating plane algorithm required approximately 35, 76, and 140 milliseconds per cycle. The results of these experiments can be seen in Figure 13 (the separating plane algorithm's times are drawn with dotted lines, while the proposed algorithm's are drawn with solid lines). The computation times were measured by a Silicon Graphics Indigo[2].

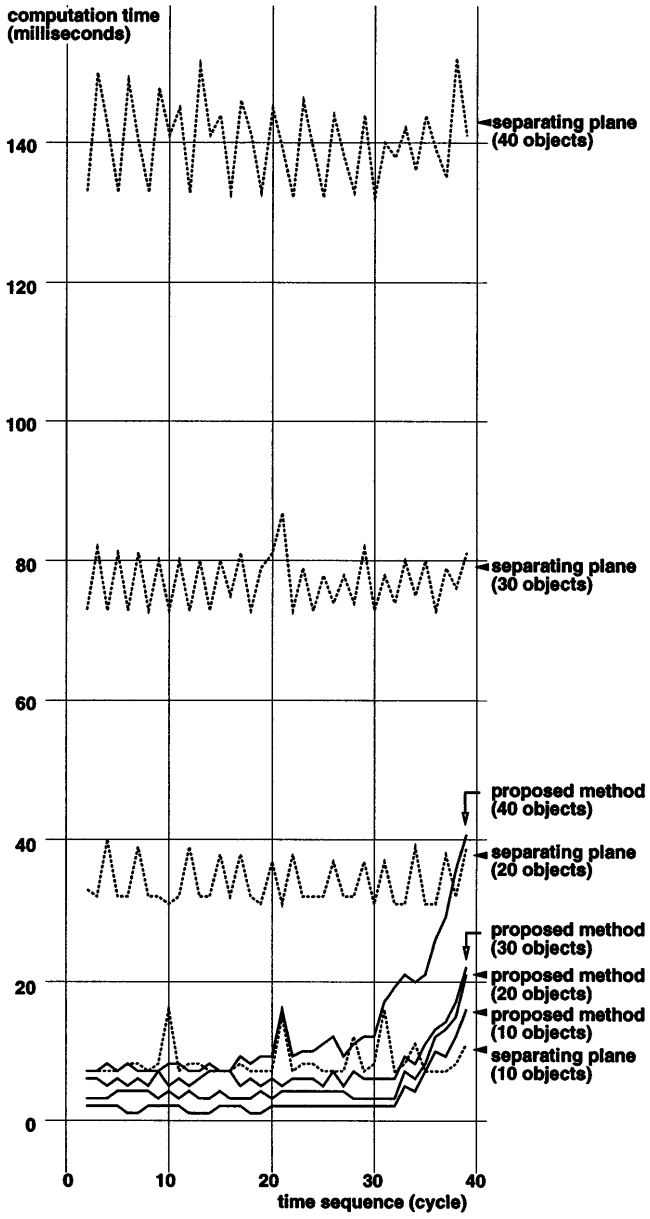

Figure 13. *Computation time for each processing cycle for the proposed algorithm (solid lines) and the separating plane algorithm (dotted lines) for 10, 20, 30, and 40 identical sphere objects (528 triangular patches each).*

5.2 Octree Update Algorithm

The second algorithm (Kitamura, Smith, Takemura, and Kishino, 1994) is a slight modification of the algorithm proposed in this paper, and is representative of the bounding region hierarchy, octree, and voxel approaches described in Section 2. Essentially, the modification is to precompute complete face octrees for all of the polyhedral objects, and to store a list for each black node of the faces that intersect that black node. Then, the proposed collision detection algorithm is modified as follows. Instead of determining the polyhedral faces that intersect with overlap regions, the octree update algorithm determines the black nodes from the precomputed face octrees that intersect with the overlap regions. These intersecting black nodes are then put into a ''node checklist'' (as opposed to a ''face checklist''). Then, in the next stage (the face octree spatial subdivision stage), instead of creating a face octree by testing for intersec-

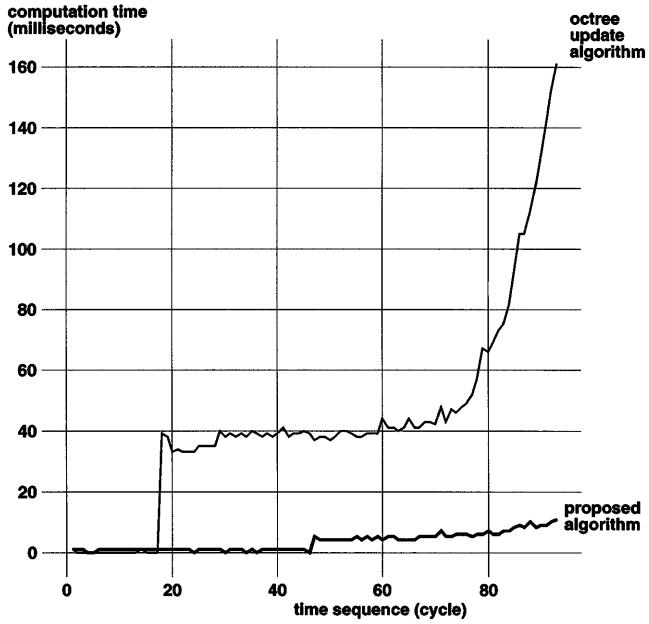

Figure 14. *Computation time for each processing cycle for the proposed algorithm and the octree update algorithm for the environment of Figure 15.*

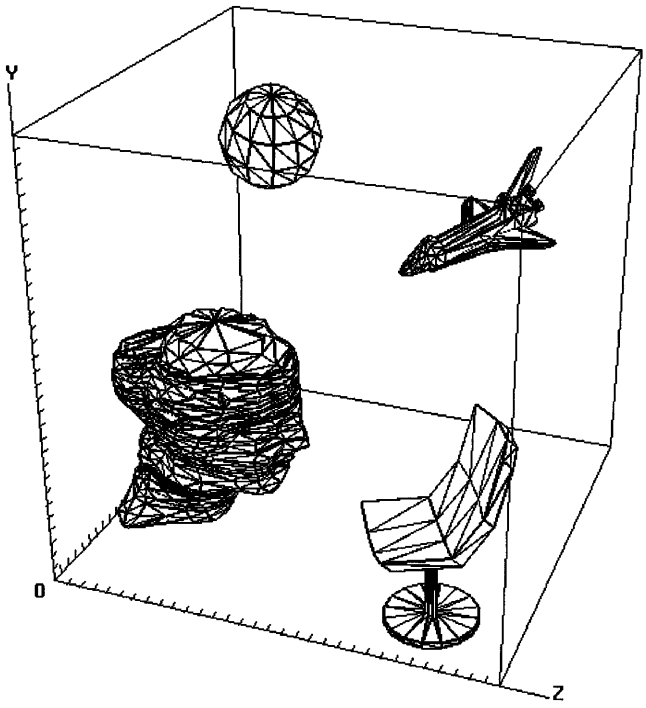

Figure 15. *The experimental environment used to obtain the data for Figure 14.*

tions between the polyhedral faces in the face check list and the standard octree nodes, the octree update algorithm builds an octree by testing for intersections between the transformed black nodes (using the same transformation matrix as for the polyhedral objects) of the node checklist and the standard octree nodes. Finally, for each standard octree voxel found to contain transformed black nodes from more than one object, all unique pairs of faces, where the faces are inside a precomputed face list of one of the transformed black nodes and the faces are from different objects, are enumerated and checked for intersection (using the method described in Section 3.6). Basically, the octree update algorithm substitutes precomputed face octree black nodes for faces in checking for intersection with overlap regions and standard octree nodes. Note that this algorithm can be used for concave objects, but that the objects must be rigid; thus, it is not as general as the proposed algorithm.

Figure 14 shows the results of an experiment in which we tested the proposed algorithm against this octree update algorithm for the environment of Figure 15; this environment contains a sphere (120 faces), a space shuttle (528 faces), a chair (146 faces), and a head of Venus (1816 faces). The experiment that was performed was to move the Venus head and the space shuttle toward each other (with translation and rotation) until they collided at the last cycle; the other two objects also translated and rotated slightly (without any collision). The proposed algorithm performed much better than the octree update algorithm for all cycles after cycle 17 (when bounding boxes first overlapped); in particular, at the last cycle the octree update algorithm required 161 milliseconds of computation, while the proposed algorithm required only 11 milliseconds (roughly 16 times better performance).

5.3 Distance-based Algorithm

The third algorithm is a distance-based approach (Cohen, et al., 1995), which involves keeping track of

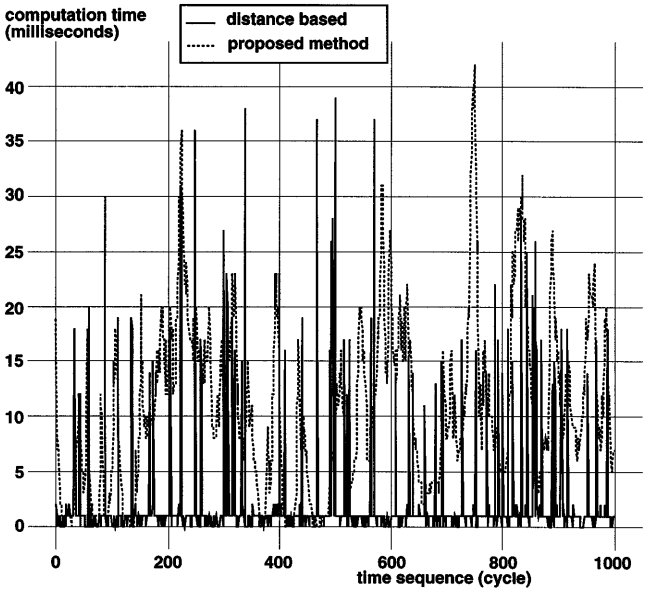

Figure 16. *Computation time at each processing cycle for the proposed algorithm and the distance-based algorithm for 15 spheres having 168 polygons.*

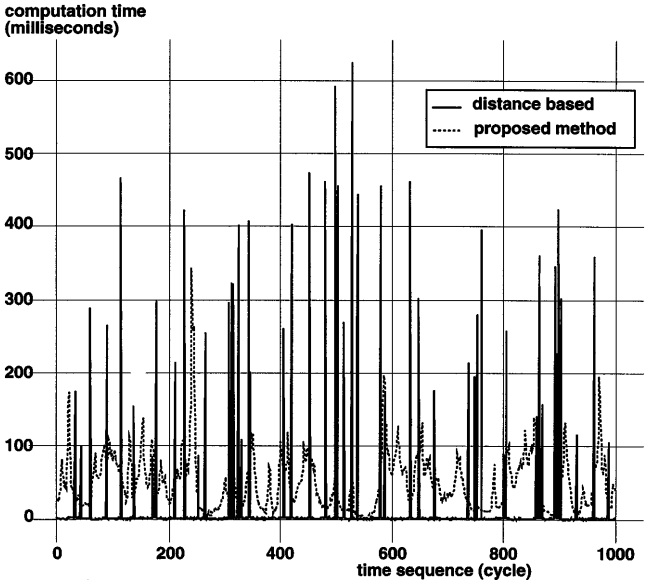

Figure 17. *Computation time at each processing cycle for the proposed algorithm and the distance-based algorithm for 15 spheres having 960 polygons.*

the distance between each closest feature pair of rigid, convex polyhedral objects in the world. It uses the coherence of objects between time instants (i.e., object positions change only slightly) and the property of convex polyhedra, so it cannot be directly used for deformable convex / concave objects. We got the program through the internet (http: / / www. cs.unc.edu / ~geom / I_COLLIDE.html) and compared it with our proposed method under the same condition with convex and rigid multiple objects.

Figures 16 and 17 show the computation time for only the collision detection routine required at each processing cycle (frame) during 1000 cycles in which many random collisions among 15 identical spheres having 168 and 960 faces occur. Measurements were made by a Silicon Graphics Onyx with an R4400 / 150 MHz processor. Both conditions correspond to 5% of the environment volume the objects occupy. The translation velocity is 5% of the object radius for each cycle, and the rotational velocity is 10 degrees per cycle for each object. Here, the motion vectors of collided objects are exchanged, and the objects are reflected at the boundary plane of the environment, so they continuously move in the defined environment. The distance-based method (which uses fixed size bounding cubes) required much computation time only when collision occurred, but it performed quite fast computation for other cycles. Local maximums were almost the same for both methods with objects having 168 faces, but our proposed method had faster local maximums than the distance-based method with more complicated objects. Figure 18 shows the average computation time for the proposed and distance-based methods throughout the experimented cycles[1] (1000 cycles), and the average computation time for only the cycles where collisions were found, with sphere objects having 48, 80, 168, 360, 528, 728, and 960 faces. Though these graphs indicate only one aspect of performance, our proposed method showed a faster computation time for the cycles where collisions occurred.

[1] Only data of this type are listed in Cohen, Lin, Manocha, and Ponamgi, 1995.

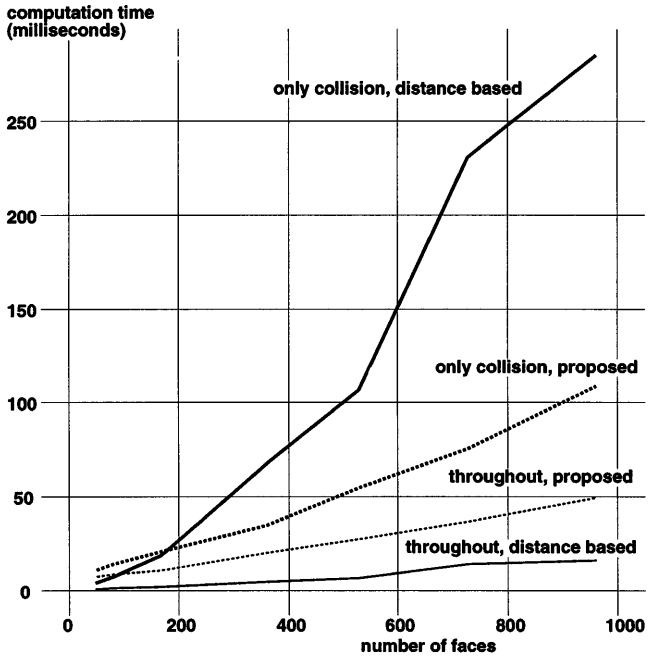

Figure 18. *Average computation time for the proposed and distance-based methods throughout the experimented cycles (1000 cycles), and average computation time for only the cycles where collisions were found, with sphere objects having 48, 80, 168, 360, 528, 728, and 960 faces.*

6 Discussion

As can be seen from the various graphs given, our collision detection algorithm is quite efficient. Card, Moran, and Newell (1983) write that if a graphic interface finishes all procedures within the cycle time of the human's perceptual processor (100 milliseconds) it provides the user with a natural impression of motion. Using this knowledge, our algorithm is able to perform collision detection for objects having up to approximately 5936 faces (extrapolated from Figure 7) within this cycle time.

We did not implement the basic, naive collision detection algorithm in order to compare it to our algorithm. Our algorithm is clearly better—see Kitamura, Takemura, and Kishino (1994) and Shinya and Forgue (1991) to see how ludicrously long the naive algorithm can take for even very simple environments. The important basis of comparison should be with other authors' accurate collision detection algorithms for general, deformable polyhedra; as shown in Section 2, there are very few collision detection algorithms that provide this generality. We were not able to compare our algorithm directly against another existing algorithm that is as general as ours; however, even against the more restrictive algorithms of the previous section our algorithm gave better performance.

Our proposed algorithm would perform quite well in many applications. Unfortunately, however, we cannot assert, based solely on the above experiments, that our algorithm is the fastest for all possible applications. More comprehensive research, which does more complete comparisons and tests variations and combinations of the various algorithms in situations that mimic real applications, might be necessary. For the time being, however, we feel that, considering the generality of our algorithm, its ease of implementation, its small memory requirements, and its proven efficiency, we have provided a practical solution to the problem of real-time collision detection.

7 Summary and Conclusion

In this paper, we have presented an efficient algorithm for accurate collision detection among polyhedral objects. The algorithm can be used for both convex and concave objects; both types of objects are dealt with in the same way and there is no performance penalty for concave objects. The algorithm can be used for objects whose motion is not prespecified, and both translation and rotation motion are allowed. The algorithm can also be used for objects that deform during motion. Thus, the algorithm is very general. The algorithm is fairly straightforward and should be easy to implement. The algorithm does not require the precomputation and update of memory-intensive auxiliary data structures, which some collision detection algorithms require and which can sap the memory resources of an application, making it impossible to perform collision detection for a large number of objects. Finally, and most importantly, even though the algorithm is very general it is extremely fast; adding many objects to the environment does not

require much more computation and the algorithm can run in real-time on a graphics workstation for polyhedra containing several thousand faces.

The performance of the proposed algorithm is compared directly against three other existing algorithms, and the results are given to show the efficiency of the proposed method in a general environment. Further speedup can be achieved by using various optimizations of this algorithm, such as using face bintrees instead of face octrees, using a more efficient bounding box check (to reduce the $O(n^2)$ complexity for n objects), and determining the optimal level for face octree subdivision (the PM-octree [Samet, 1990] might be useful for this). In addition, because the algorithm calculates many independent intersections, further speedup can be easily achieved by parallelization (Kitamura et al., 1995). The algorithm is already sufficiently fast for most applications. However, with anticipated speedups from optimization and parallelization, our algorithm should be suitable for very large, practical virtual environments.

References

Arvo, J. (Ed.) (1991). *Graphics gems II*. Boston: Academic Press Professional.

Baraff, D. (1990). Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics, 24*(4), 19–28.

Bouma, W., & Vanecek, G. (1991). Collision detection and analysis in a physical based simulation. *Proceedings of Eurographics Workshop on Animation and Simulation,* 191–203.

Boyse, J. W. (1979). Interference detection among solids and surfaces. *Communications of the ACM, 22*(1), 3–9.

Card, S. K., Moran, T. P., & Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Chung, K., & Wang, W. (1996). Quick collision detection of polytopes in virtual environments. *Proceedings of Symposium on Virtual Reality Software and Technology,* 125–132, ACM.

Cohen, J. D., Lin, M. C., Manocha, D., & Ponamgi, M. (1995). I-COLLID: An interactive and exact collision detection system for large-scale environments. *Proceedings of the Symposium on interactive 3D Graphics,* 189–196, ACM.

Fairchild, K. M., Poston, T., & Bricken, W. (1994). Efficient virtual collision detection for multiple users in large virtual spaces. *Proceedings of Virtual Reality Software and Technology,* 271–285, ACM.

Foisy, A., Hayward, V., & Aubry, S. (1990). The use of awareness in collision prediction. *Proceedings of International Conference on Robotics and Automation,* 338–343, IEEE.

Fujimura, K., & Kunii, T. (1985). A hierarchical space indexing method. *Proceedings of Visual Technology and Art (Computer Graphics Tokyo),* 21–33.

Garcia-Alonso, A., Serrano, N., & Flaquer, J. (1994). Solving the collision detection problem. *Computer Graphics and Applications, 14*(3), 36–43.

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

Gottschalk, S., Lin, M. C., & Manocha, D. (1996). OBBTree: a hierarchical structure for rapid interference detection. *Computer Graphics Proceedings, Annual Conference Series,* 171–180, ACM.

Hahn, J. K. (1988). Realistic animation of rigid bodies. *Computer Graphics, 22*(4), 299–308.

Hayward, V. (1986). Fast collision detection scheme by recursive decomposition of a manipulator workspace. *Proceedings of the International Conference on Robotics and Automation,* 1044–1049, IEEE.

Heckbert, P. (Ed.) (1994). *Graphics gems IV.* Boston: Academic Press Professional.

Hubbard, P. M. (1993). Interactive collision detection. *Proceedings of the Symposium on Research Frontiers in Virtual Reality,* 24–31, IEEE.

Kirk, D. (Ed.) (1992). *Graphics gems III.* Boston: Academic Press Professional.

Kitamura, Y., & Kishino, F. (1996). Real-time colliding face determination in a general 3-D environment. *Video Proceedings of the Virtual Reality Annual International Symposium,* IEEE.

Kitamura, Y., Takemura, H., & Kishino, F. (1994). Coarse-to-fine collision detection for real-time applications in virtual workspace. *Proceedings of the International Conference on Artificial Reality and Tele-Existence,* 147–157.

Kitamura, Y., Yee, A., & Kishino, F. A sophisticated manipulation aid in a virtual environment using the dynamic constraints among object faces. *Presence, Teleoperators and Virtual Environments.* (Submitted).

Kitamura, Y., Smith, A., Takemura, H., & Kishino, F. (1994). Optimization and parallelization of octree-based collision detection for real-time performance. *Proceedings of IEICE Conference 1994, Autumn,* D-323 (in Japanese).

Kitamura, Y., Smith, A., Takemura, H., & Kishino, F. (1995). Parallel algorithms for real-time colliding face detection. *Proceedings of the International Workshop on Robot and Human Communication,* 211–218, IEEE.

Lin, M. C., & Canny, J. F. (1991). A fast algorithm for incremental distance calculation. *Proceedings of the International Conference on Robotics and Automation,* 1008–1014, IEEE.

Lin, M. C., Manocha, D., & Canny, J. F. (1994). Fast contact determination in dynamic environments. *Proceedings of the International Conference on Robotics and Automation,* 602–608, IEEE.

Mantyla, M., & Tamminen, M. (1983). Localized set operations for solid modeling. *Computer Graphics, 17*(3), 279–288.

Maruyama, K. (1972). A procedure to determine intersections between polyhedral objects. *International Journal of Computer and Information Sciences, 1*(3), 255–266.

Miyasato, T., Kishino, F., & Terashima, N. (1995). Virtual space teleconferencing: Communication with realistic sensations. *Proceedings of the International Workshop on Robot and Human Communication,* 205–210, IEEE.

Miyasato, T., Ohya, J., & Kishino, F. (1996). Virtual space teleconference—communication with realistic sensations. *Video Proceedings of the Virtual Reality Annual International Symposium,* IEEE.

Moore, M., & Wilhelms, J. (1988). Collision detection and response for computer animation. *Computer Graphics, 22*(4), 289–298.

Pentland, A. P. (1990). Computational complexity versus simulated environments. *Computer Graphics, 24*(2), 185–192.

Quinlan, S. (1994). Efficient distance computation between non-convex objects. *Proceedings of the International Conference on Robotics and Automation,* 3324–3329, IEEE.

Samet, H. (1990). *The design and analysis of spatial data structures.* Reading, MA: Addison-Wesley.

Shaffer, C. A., & Herb, G. M. (1992). A real-time robot arm collision avoidance system. *Transactions on Robotics and Automation, 8*(2), 149–160, IEEE.

Shinya, M., & Forgue, M. (1991). Interference detection through rasterization. *The Journal of Visualization and Computer Animation, 2,* 132–134.

Takemura, H., & Kishino, F. (1992). Cooperative work environment using virtual workspace. *Proceedings of the Annual Conference on Computer Supported Cooperative Work,* 226–232, ACM.

Turk, G. (1989). Interactive collision detection for molecular graphics. M.Sc. thesis, Department of Computer Science, University of North Carolina at Chapel Hill.

Vanecek, G. (1994). *Back-face culling applied to collision detection of polyhedra.* Technical report, Purdue University Department of Computer Science.

Zyda, M. J., Pratt, D. R., Osborne, W. D., & Monahan, J. G. (1993). NPSNET: Real-time collision detection and response. *The Journal of Visualization and Computer Animation, 4*(1), 13–24.