A Simple and Efficient Method for Accurate Collision Detection Among Deformable Polyhedral Objects in Arbitrary Motion

Andrew Smith Yoshifumi Kitamura Haruo Takemura* Fumio Kishino

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

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 (non-rigid) 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. Since polyhedral shape representation is one of the most common shape representation schemes, this algorithm should be useful to a wide audience. Performance results are given to show the efficiency of the proposed method.

1 Introduction

In a virtual environment, we can simulate various kinds of physical phenomena. An important example of this is being able to determine when moving objects collide; this is called the "collision detection problem." It is vitally important to be able to update a virtual environment at real-time rates to engender realism for a user. Unfortunately, current collision detection algorithms, if used, are an enormous bottleneck and make real-time update impossible [1, 2]. The difficulty of collision detection for 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 so that collisions are not missed. Then, 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 [3]. 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 which checks a number of edge-face pairs proportional to the number that actually intersect. In this paper, we present an algorithm that realizes this and can be used for general (i.e., the environment can contain both convex and concave objects), deformable polyhedral objects undergoing arbitrary motion.

The remainder of the paper is organized as follows. The next section discusses other research efforts towards 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. Finally, the last section concludes the paper.

2 Efficient Collision Detection Approaches

There is much literature devoted to efficient collision detection approaches and this section discusses this research. The first subsection simply describes other approaches to efficient collision detection. Then, the last two subsections evaluate these other approaches, describing the problems with them which make

*Currently with the Nara Institute of Science and Technology, Japan (takemura@is.aist-nara.ac.jp) them not entirely suitable for practical, large-scale virtual environments and how our algorithm addresses these problems.

2.1 Related Collision Detection Research

Much 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 [2]. Related methods use octrees and voxel sets. [4] stores a voxel data structure with each object, with pointers from voxels to polyhedra faces that intersect them. Collision is localized by testing for intersection of voxels between two objects. [5] stores an octree for each object and, at each time instant, checks the interference of objects' updated octrees; face pairs from inside of interfering octree nodes are then checked for collision. Other voxel and octree methods include [6–10].

Another method for 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 collision detection of rigid, convex objects is [11], where 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 roughly constant time per object pair. Other research which uses this distance based approach include [12, 13].

Briefly, some other approaches to collision detection are as follows. [14] uses a data structure called a "BRep-Index" (an extension of the well-known BSP tree) for quick spatial access of a polyhedron in order to localize contact regions between two objects. [15] 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 non-collision most of the time. [16] uses ideas from the z-buffer visible surface algorithm to perform interference detection through rasterization. [17] uses back-face culling to remove roughly half of the faces of objects from being checked for detailed interference; the basic idea is that polygons of a moving object which do not face in the general direction of motion cannot possibly collide. [18] uses a scheduling scheme, whereby object pairs are sorted by distance and only close objects are checked at each time instance. [19] 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. [1] 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, [20] stores 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 this (to reduce computation).

Finally, our algorithm uses ideas from methods for localized set operations on polyhedra [21, 22]. These methods attempt to perform efficiently set operations, such as intersection, union, etc., on polyhedra by localizing the regions where faces are using 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 [23] and we use this idea effectively in our algorithm.

2.2 Evaluation

We evaluate the above algorithms on the basis of four properties of a collision detection algorithm necessary for effective use in a practical, large-scale virtual environment inhabited by humans. These are the ability to handle deformable (non-rigid) objects, the ability to handle concave objects, not using excessive amounts of memory for storing auxiliary data structures, and having better than $O(n^2)$ complexity for n objects in the world. None of the algorithms surveyed in the previous subsection has 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 which 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, which necessitates 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, BRep-indices, etc.) 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, storing auxiliary data structures for each object can take up considerable memory. This limits the number of objects for which such algorithms can be effectively used. 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 [11–13, 15]. 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 purposes 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. [1] 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 which 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 environments with more than 14 objects (being able to compute something in 100 milliseconds or less is considered to be real-time performance [24]). All of the distance based approaches [11–13] and many of the others [1, 4, 15] 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. However, the bounding box test between two objects is extremely fast and thus should not become a bottleneck unless there are many objects in the environment; for such an environment, however, the problem can be easily solved by using a bounding box check with better complexity ([25] describes such a method) or by skipping the bounding box stage altogether and going directly to the face octree spatial subdivision stage which is $O(n)$ for n objects.

2.2.5 Other

A few other minor problems with the surveyed algorithms are as follows. Some of these algorithms [1, 7] cannot be used for polyhedra, which limits their usefulness for current graphical applications where polyhedra dominate as the object representation. Some of the algorithms do not provide accurate collision detection (i.e., identify exactly which objects are interfering and which faces of the objects interfere— [5] describes how this is useful for operator assistance) among objects [16, 18–20]. 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. The details of our collision detection algorithm are presented next.

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 [21–23] to a world with multiple objects; these papers describe algorithms for 2 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 a later section (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 for performing 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.

3 Collision Detection Algorithm

3.1 Assumptions

All objects in the world are modeled as polyhedron (boundary representation). The faces of a polyhedron are assumed to be triangular patches without any loss of generality of range of representation. Objects can be concave or convex. The objects are undergoing motion which is not predetermined (e.g., a user can move his graphical hand in a sequence of non-predetermined, jerky motions); object motion can be both translation and rotation. 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 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 speeds of objects are sufficiently slow compared with the sampling interval so that collisions are not missed. Finally, it is assumed that there is a large cube which 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 $\cdots$, t_{i-1}, t_i, t_{i+1}, $\cdots$) 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 other objects' bounding boxes are determined. Next, for each such object all faces of the object are checked for intersection with the overlap regions of the object; a list of the object's faces which 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 which are from separate objects and which intersect the same face octree voxel (i.e., 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.

3.3 Approximate Interference Detection Using Bounding Boxes

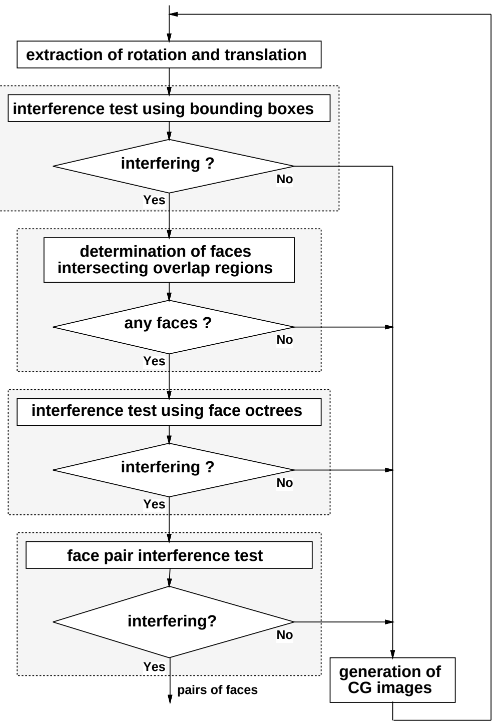
At every time instant, axis-aligned bounding boxes are computed for all objects and all pairs of objects are compared for overlap of their bounding boxes. For each pair of objects whose bounding boxes overlap, the intersection between the two bounding boxes is determined (called an overlap region as 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.

3.4 Determination of Faces Intersecting Overlap Regions

For every object which 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 check list for the object. If there are face check lists 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 8 children only if it contains faces from two or more objects, and only the faces which were found to intersect the parent node are tested for intersection with the children nodes. Also, there is no condensation of the face octree (i.e., 8 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



extraction of rotation and translation
interference test using bounding boxes
interfering ?
Yes
No
determination of faces intersecting overlap regions
any faces ?
Yes
No
interference test using face octrees
interfering ?
Yes
No
face pair interference test
interfering?
Yes
No
pairs of faces
generation of CG images

Figure 1: Control flow of collision detection.

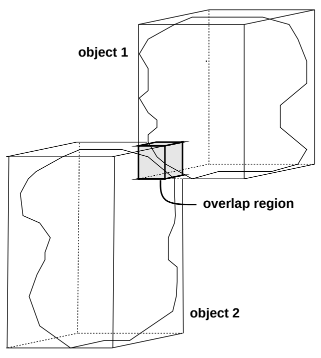


Figure 2: An overlap region

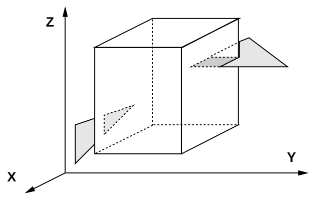


Figure 3: Faces intersecting the overlap region

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, for storing octrees is necessary). 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. 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. For each edge, in turn, of face 1 the intersection point of it with the face plane of face 2 is found and checked to see if it is inside face 2 (i.e., three-dimensional point-in-polygon check—the method used is described in [26]); if the point is inside the face then the two faces intersect. The case when an edge and face plane are coplaner 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 4 of the face planes of the faces that comprise the box; the 4 face planes are the maximum and minimum extents of two of the three x,y,z dimensions (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 four planes. During 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 triangular patch and box intersect and no more computation needs to 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 non-intersection 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.

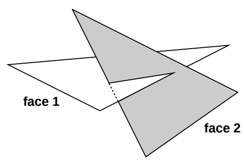


Figure 4: Face pair intersection test.

4 Experiments

The algorithm and an experimental environment were implemented and run on a Silicon Graphics Indigo^2 (this has an R4400/150 MHz processor); 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, sphere-like 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 which were used in the experiments. The basic experiment done was to have two identical sphere objects start at different (non-penetrating) positions and have them move towards each other (with both translation and rotation motion) until they interfere. This basic experiment was done with sphere objects having respectively 8, 10, 24, 48, 54, 80, 120, 168, 224, 360, 528, 728, 960, and 3968 triangular patches. Figure 6 shows the computation time 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 proposed collision detection algorithm between two sphere objects against the number of triangular patches of the sphere objects.

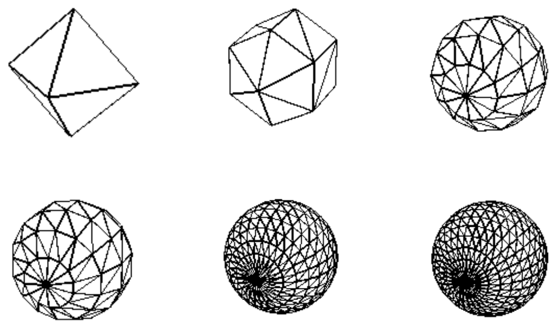


Figure 5: Examples of experimental objects (standardized spheres with different numbers of faces)

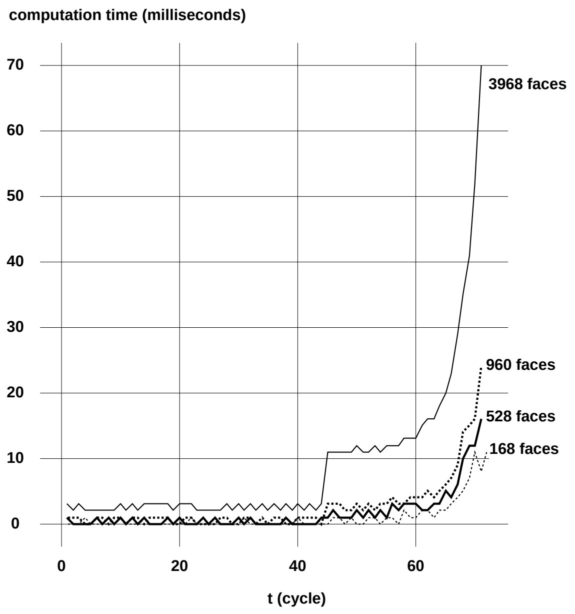


Figure 6: Computation time for each processing cycle for two identical sphere objects with 168, 528, 960 and 3968 faces.

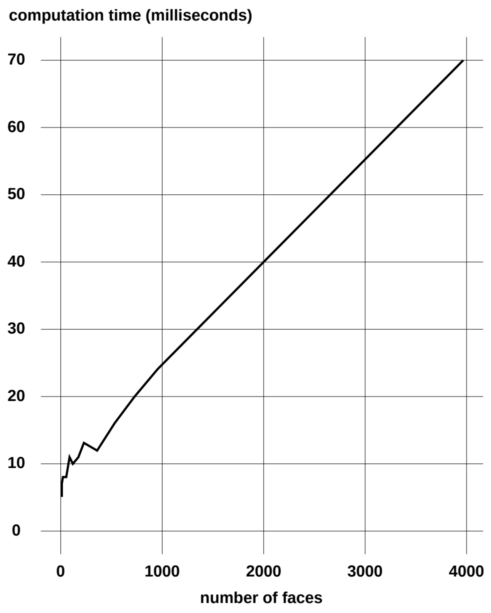


Figure 7: Computation time at the last stage of the proposed collision detection between two identical sphere objects against the number of planar patches of the objects.

4.2 Multiple General Objects

An experiment was also done with multiple general (i.e., concave—a real-world type of object) objects. Specifically, 15 identical objects (space shuttles with 528 triangular patches—see figure 9) were moved (both translation and rotation) 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, in order to provide a basis for comparison, are the results for when only the two interfering space shuttle objects are in the test environment; here, the last step, where faces are determined to be colliding, required 16 milliseconds of computation.

4.3 Comparison Against Competing Algorithms

In order to show that our algorithm is truly efficient, we directly compared the performance of our algorithm against two other competing algorithms. 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 experimental results. Fortunately, however, we found the C language code for the first competing collision

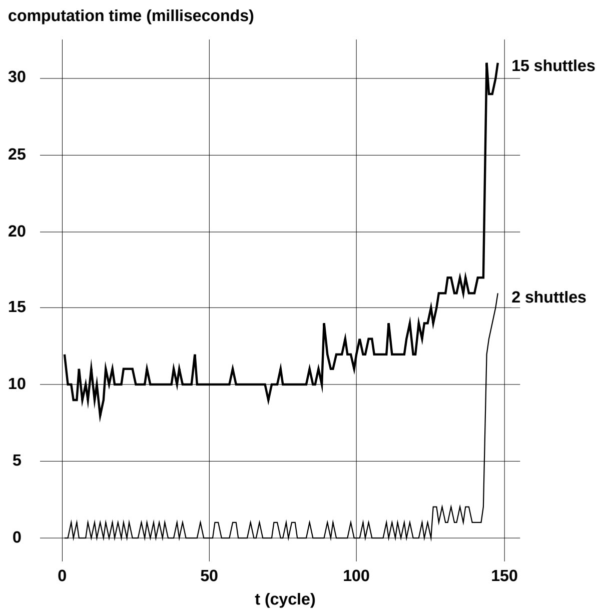


Figure 8: Computation time at each processing cycle for 15 space shuttle objects-collision between two objects is detected at the last cycle.

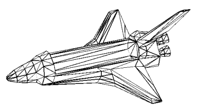


Figure 9: The space shuttle experimental object (528 triangular patches).

detection algorithm in [27], and the second competing algorithm is a slight modification of the algorithm proposed in this paper.

4.3.1 Separating Plane Algorithm

The first competing algorithm is based on ideas from [12] and [15]. 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. Thus, it is not completely fair to compare our algorithm against this algorithm because our algorithm is more general and gives more complete collision analysis. Even so, however, our algorithm gives better performance for non-trivial virtual environments.

The details of this competing algorithm are given in [27]. 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 two objects (using the method in [12]); the vector between these two points is the normal vector of the plane and the plane passes 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 then 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 algorithm against this competing algorithm for environments containing differing numbers of 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 were interfering. For 10 sphere objects, our algorithm performed roughly the same as the competing algorithm; in particular, our algorithm required 16 milliseconds of computation at the last cycle, while the competing 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 competing algorithm required approximately 35, 76, and 140 milliseconds per cycle. The results of these experiments can be seen in figure 10 (the competing algorithm's times are drawn with dotted lines, while the proposed algorithm's are drawn with solid lines).

4.3.2 Octree Update Algorithm

The second competing algorithm [28] 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 the section on related work (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 which intersect that black node. Then, the proposed collision detection algorithm is modified as follows. Instead of determining the polyhedral faces which intersect with overlap regions, the octree update algorithm determines the black nodes from the precomputed face octrees which intersect with the overlap regions; these intersecting black nodes are then put into a "node check list" (as opposed to a "face check list"). Then, in the next stage (face octree spatial subdivision stage), instead of creating a face octree by testing for intersections 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 (i.e., using the same transformation matrix as for the polyhedral objects) black nodes of the node check list and the standard octree nodes. Finally, for each standard octree voxel which was 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 objects must be rigid; thus, it is not as general as the proposed algorithm.

We tested the proposed algorithm against this competing algorithm for the environment of figure 12; this environment contains a sphere (120 faces), a space shuttle (528 faces), a chair (146 faces), and a Venus head (1816 faces). The experiment that was performed was to move the venus head and the space shuttle towards 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 competing algorithm for all cycles after cycle 17 (when bounding boxes first overlapped); in particular, at the last cycle the competing algorithm required 161 milliseconds of computation, while the proposed algorithm required only 11 milliseconds (roughly 16 times better performance). Figure 11 shows the results of this experiment.

5 Discussion

As can be seen from the various graphs given, our collision detection algorithm is quite efficient. A common definition for "real-time" performance of a computer graphics application is being able to render 10 frames per second [24]. Using this definition, our algorithm is able to perform real-time collision detection for objects having up to approximately 5936 faces (extrapolated from figure 7). Also important is the fact that the algorithm takes negligible compute time (rarely more than 10 milliseconds) when no objects in the environment are interfering. In addition, adding many objects to the environment increases computation time only slightly (i.e., for the case that only two objects at a time interfere—if more objects interfere at the same time then computation time will increase, but not greatly).

We did not implement the basic, naive collision detection algorithm in order to compare it to our algorithm (because our algorithm is clearly better—see [5] and [16] 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 the section on related work (section 2), there are very few collision detection algorithms which providerevised this generality. We were not able to compare directly our algorithm against another competing algorithm which is as general as ours; however, even against the more restrictive algorithms of the previous section our algorithm gives better performance.

Based on these experiments, it seems reasonable to conclude that our algorithm would perform quite well in many applications. Unfortunately, however, we cannot assert, based solely on these experiments, that our algorithm is the fastest for all possible applications. There has already been much research into efficient collision detection, and many different efficient approaches have been proposed. We feel that, in addition to exploring new collision detection approaches, "comparative collision detection" would be a worthy new research topic. We feel that our proposed algorithm would fare well in such a comparative study, and we have made a start towards such research with our comparisons against two competing algorithms. However, more comprehensive research, which does more complete comparisons and which tests variations and combinations of the various algorithms in situations that mimic real applications, is 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.

6 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. And 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 thousands of faces.

We are currently exploring various optimizations to 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 [29] might be useful for this). In addition, we are implementing a parallel version of the algorithm, which should be quite effective because of the many independent intersection calculations done by the algorithm. 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

[1] Pentland, Alex P. Computational complexity versus simulated environments. *Computer Graphics*, Vol. 24, No. 2, pp. 185–192, 1990.

[2] Hahn, James K. Realistic animation of rigid bodies. *Computer Graphics*, Vol. 22, No. 4, pp. 299–308, 1988.

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

[4] Garcia-Alonso, A., Serrano, N., and Flaquer, J. Solving the collision detection problem. *Computer Graphics and Applications*, Vol. 14, No. 3, pp. 36–43, May 1994.

[5] Kitamura, Y., Takemura, H., and Kishino, F. Coarse-to-fine collision detection for real-time applications in virtual workspace. In *International Conference on Artificial Reality and Tele-Existence*, pp. 147–157, July 1994.

[6] Moore, M. and Wilhelms, J. Collision detection and response for computer animation. *Computer Graphics*, Vol. 22, No. 4, pp. 289–298, 1988.

[7] Turk, Greg. Interactive collision detection for molecular graphics. M.sc. thesis, Department of Computer Science, University of North Carolina at Chapel Hill, 1989.

[8] Zyda, M. J., Pratt, D. R., Osborne, W. D., and Monahan, J. G. NPSNET: Real-time collision detection and response. *The Journal of Visualization and Computer Animation*, Vol. 4, No. 1, pp. 13–24, 1993.

[9] Shaffer, C. A. and Herb, G. M. A real-time robot arm collision avoidance system. *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 2, pp. 149–160, 1992.

[10] Hayward, V. Fast collision detection scheme by recursive decomposition of a manipulator workspace. In *International Conference on Robotics and Automation*, pp. 1044–1049. IEEE, 1986.

[11] Lin, M. C., Manocha, D., and Canny J. F. Fast contact determination in dynamic environments. In *International Conference on Robotics and Automation*, pp. 602–608. IEEE, 1994.

[12] Gilbert, Elmer G., Johnson, Daniel W., and Keerth, S. Sathiya. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE Journal of Robotics and Automation*, Vol. 4, No. 2, pp. 193–203, 1988.

[13] Quinlan, Sean. Efficient distance computation between non-convex objects. In *International Conference on Robotics and Automation*, pp. 3324–3329. IEEE, 1994.

[14] Bouma, W. and Vanecek, G. Collision detection and analysis in a physical based simulation. In *Eurographics Workshop on Animation and Simulation*, pp. 191–203, September 1991.

[15] Baraff, David. Curved surfaces and coherence for non-penetrating rigid body simulation. *Computer Graphics*, Vol. 24, No. 4, pp. 19–28, 1990.

[16] Shinya, M. and Forgue, M. Interference detection through rasterization. *The Journal of Visualization and Computer Animation*, Vol. 2, pp. 132–134, 1991.

[17] Vanecek, George. Back-face culling applied to collision detection of polyhedra. Technical report, Purdue University Department of Computer Science, 1994.

[18] Foisy, A., Hayward, V., and Aubry, S. The use of awareness in collision prediction. In *International Conference on Robotics and Automation*, pp. 338–343. IEEE, 1990.

[19] Hubbard, Philip M. Interactive collision decision. In *Symposium on Research Frontiers in Virtual Reality*, pp. 24–31. IEEE, 1993.

[20] Fairchild, K. M., Poston, Timothy, and Bricken, William. Efficient virtual collision detection for multiple users in large virtual spaces. In *Virtual Reality Software and Technology*, 1994.

[21] Mantyla, M. and Tamminen, M. Localized set operations for solid modeling. *Computer Graphics*, Vol. 17, No. 3, pp. 279–288, July 1983.

[22] Fujimura, K. and Kunii, T. A hierarchical space indexing method. In *Visual Technology and Art (Computer Graphics Tokyo)*, pp. 21–33, 1985.

[23] Maruyama, K. A procedure to determine intersections between polyhedral objects. *International Journal of Computer and Information Sciences*, Vol. 1, No. 3, pp. 255–266, 1972.

[24] Card, S. K., Moran, T. P., and Newell, A. *The Psychology of Human-Computer Interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

[25] Kirk, David, editor. *Graphics Gems III.* Academic Press Professional, 1992.

[26] Arvo, James, editor. *Graphics Gems II.* Academic Press Professional, 1991.

[27] Heckbert, Paul, editor. *Graphics Gems IV.* Academic Press Professional, 1994.

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

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

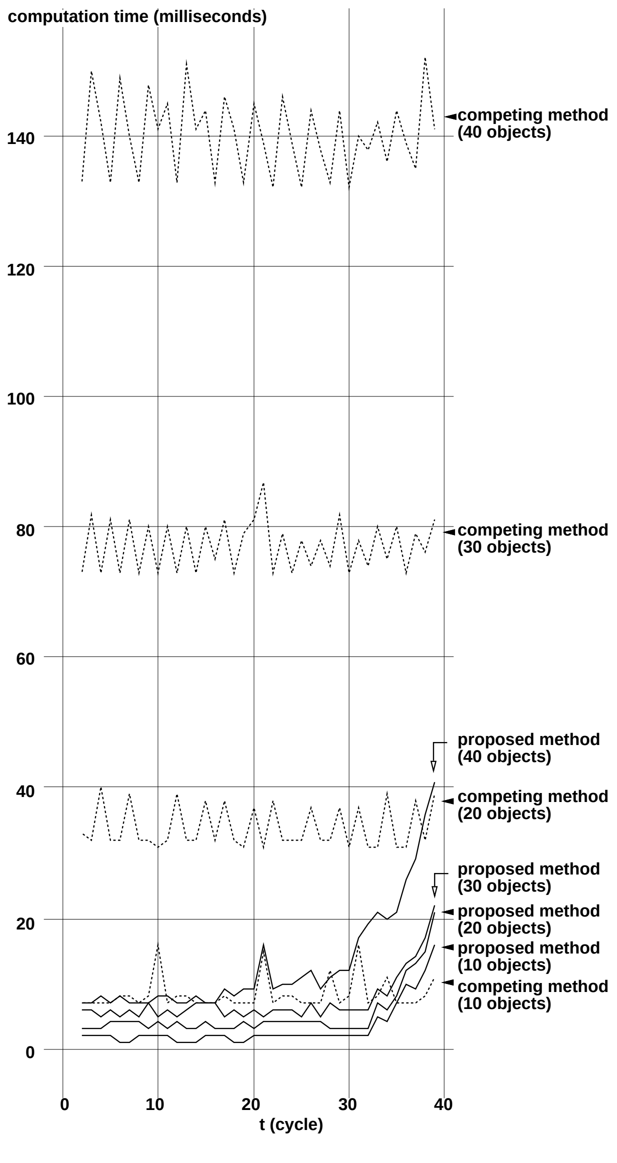


Figure 10: Computation time for each processing cycle for the proposed algorithm (solid lines) and the separating plane competing algorithm (dotted lines) for 10, 20, 30, and 40 identical sphere objects (528 triangular patches each).

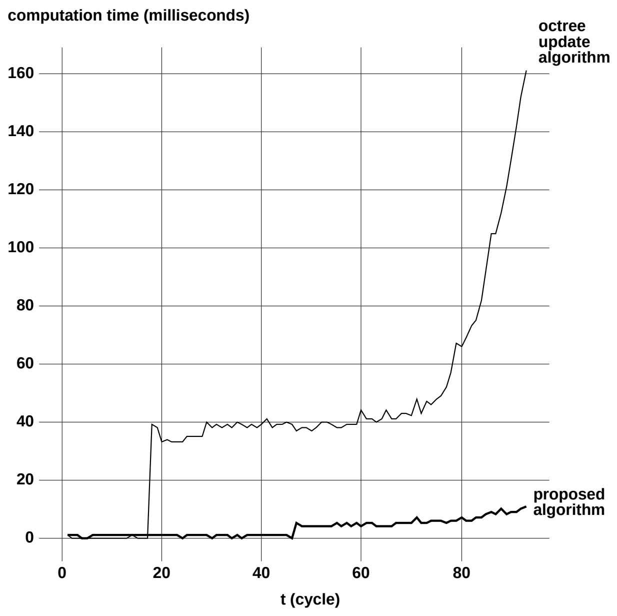


Figure 11: Computation time for each processing cycle for the proposed algorithm and the octree update competing algorithm for the environment of figure 12.

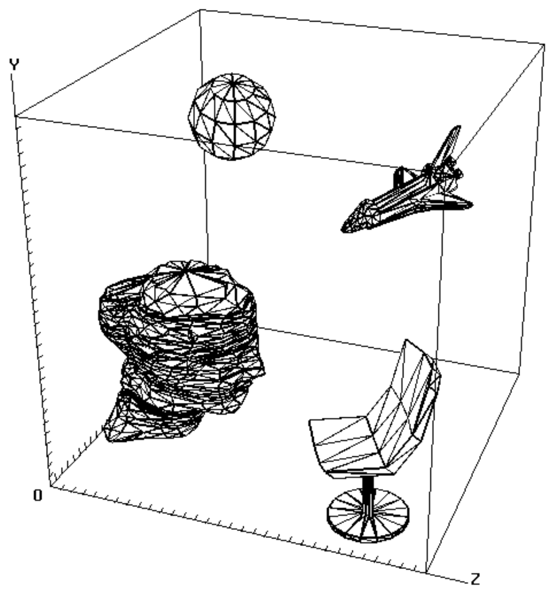


Figure 12: The experimental environment used to obtain the data for figure 11.