Quick Collision Detection of Polytopes in Virtual Environments

Kelvin Chung and Wenping Wang
tlchung@cs.hku.hk *wenping@cs.hku.hk*
Department of Computer Science
University of Hong Kong
Pokfulam Road, Hong Kong

Abstract

The problem of collision detection is fundamental to interactive applications such as computer animation and virtual environments. In these fields, prompt recognition of possible impacts is important for computing real-time response. We present a simple exact collision detection algorithm for convex polytopes. The algorithm finds quickly a separating plane between two polytopes if they are non-colliding, or else reports collision if it cannot possibly find a separating plane. In the case of non-collision, the separating plane found for one time frame is cached as a witness for the next time frame, an idea borrowed from [10]; this use of time coherence further speeds up the algorithm in dynamic applications. Both temporal and geometric coherences are exploited to make this algorithm run in expected constant time empirically.

1 Introduction

The problem of collision detection has been extensively studied in many fields. Most of the research makes use of rectangular bounding boxes or a hierarchy of them as the first step to quickly eliminate non-interference objects. For n bounding boxes, a sweep and prune technique [1] can achieve an expected $O(n+e)$ time by projecting the corner points of three-dimensional bounding boxes onto the x, y, z axes and sorting them at each time instant. Other methods to reduce the complexity of the bounding box tests include spatial subdivision [3], octree [2], scheduling [4], and progressive refinement [14].

When the bounding boxes of objects overlap, usually an exact collision detection algorithm is called. In [5], a face octree is built for the faces of objects that intersect the overlapping region of bounding boxes to check for possible intersection. In [6] the rectangular box of an object is subdivided into cells with each cell containing a list of facets intersecting the cell. Intersection is done by considering only the facets in the overlapped cells. In [9], a data structure, called a “BRep-Index”, is used for quick spatial access of polyhedra in order to localize contact regions between two objects. In [15], an expected linear time algorithm which computes the minimum distance and the separating plane of two objects is proposed. In [11] the separating planes for all pairs of non-interference objects are found by the above expected linear time algorithm and cached [10] to facilitate collision detection using temporal coherence. However, it still takes linear time in the following time frame to test the validity of the cached separating plane. In [16], an algorithm with sub-quadratic running time algorithm to detect collision between polytopes is proposed. When the motion is restricted to be translational only, the best theoretical time for detecting collision between two polytopes is $O(\log^2 n)$, using the hierarchical representation of convex polyhedra [12], which needs $O(n)$ preprocessing time to build. In [17], the ideas of [1] and [8] are extended to deal with concave polytopes. Other methods to detect collision usually decompose the object into hierarchical structure and can deal with concave polyhedra. They include the octree [1, 7], the BSP tree [3] and the OBB-Tree[20] techniques. Among them, OBB-Tree is more efficient than others. However, for convex polyhedra geometric coherence can be exploited to achieve better performance without decomposing the polyhedra.

The method in [8] maintains a pair of closest features for each pair of polytopes and calculates the Euclidean distance between the features to detect collision based on Voronoi regions. This method takes advantage of geometric coherence and runs in expected constant time if the polytopes do not move swiftly. Since this algorithm needs to compute and store the Voronoi region for each feature (vertex, edge, or face) on the boundary, and to handle different cases when walking around on the boundary in order to find the closest features pairs, the implementation is not trivial. Moreover, in most applications the closest features are not of great interest to the program when the polytopes do not collide. So it is not worth continuing to compute the closest features once it is known that a separating plane exists between the two polytopes.

Our algorithm is a major improvement on existing algorithms in terms of running time, implementation simplicity, and memory requirement. It extends the idea in [11] of searching for a separating plane between two polytopes. But we search a separating plane with a different method and verify its validity in expected constant time instead of linear time as in [11]. If there is no collision, our algorithm will find a proper separating plane quickly, or else it will report collision after testing some simple conditions. It makes use of temporal coherence by caching a separating plane for successive time frames. Our algorithm does not compute the closest features as done in [8], although such features may be useful in animation for computing collision impulse when a collision is detected. Our algorithm considers polyhedral vertices only, instead of all boundary features (vertices, edges, and faces) as in [8], so it is more efficient and simpler to implement. Temporal and geometric coherences are exploited to make the algorithm run in expected constant time.

2 Separating Vector Searching Algorithm

2.1 Algorithm Overview

Our algorithm is an exact collision detection algorithm between convex polytopes. The idea is to detect collision between polytopes quickly using the fact that two polyhedra do not collide if and only if there exists a separating plane between the two objects [18]. At each iteration, the algorithm finds a candidate plane and uses constant time to verify whether this plane is a separating plane. If it is a separating plane then the polytopes do not collide, and this plane is cached to be used as the initial plane in the search for a separating plane in the next time frame; otherwise the algorithm continues to search for a separating plane. If the algorithm has determined that a separating plane cannot possibly exist (to be explained later), it reports collision.

2.2 The Algorithm

Definition 1: *Let* $\mathbf{V}(P)$ *denote the set of vertices of polytope* P*. A supporting vertex of* P *in the direction* $\mathbf{S}$ *is* $\mathbf{p} \in \mathbf{V}(P)$ *such that* $\mathbf{S} \cdot \mathbf{p} = \max\{\mathbf{S} \cdot \mathbf{p}' | \mathbf{p}' \in \mathbf{V}(P)\}$*.*

In fact, for a supporting vertex $\mathbf{p}$ of P, we have $\mathbf{S} \cdot \mathbf{p} = \max\{\mathbf{S} \cdot \mathbf{p}' | \mathbf{p}' \in P\}$.

Lemma 1: *For a vector* $\mathbf{S}$*, let* $\mathbf{p}$ *be a supporting vertex of polytope* P *in the direction* $\mathbf{S}$ *and* $\mathbf{q}$ *be a supporting vertex of polytope* Q *in the direction* $-\mathbf{S}$*. If* $\mathbf{S} \cdot (\mathbf{q} - \mathbf{p}) > 0$*, then* P *and* Q *do not intersect.*

Proof: Since $\mathbf{S} \cdot (\mathbf{q} - \mathbf{p}) > 0$, we have $\mathbf{S} \cdot \mathbf{q} > \mathbf{S} \cdot \mathbf{p}$, which implies $\mathbf{S} \cdot \mathbf{q} > \mathbf{S} \cdot (\mathbf{p} + \mathbf{q})/2 > \mathbf{S} \cdot \mathbf{p}$. By definition, $\mathbf{S} \cdot \mathbf{p} \geq \mathbf{S} \cdot \mathbf{p}'$ for any $\mathbf{p}' \in P$ and $\mathbf{S} \cdot \mathbf{q}' \geq \mathbf{S} \cdot \mathbf{q}$ for any $\mathbf{q}' \in Q$. So $\mathbf{S} \cdot \mathbf{q}' > \mathbf{S} \cdot (\mathbf{p} + \mathbf{q})/2 > \mathbf{S} \cdot \mathbf{p}'$. Hence the plane containing the point $(\mathbf{p} + \mathbf{q})/2$ with normal vector being $\mathbf{S}$ separates properly P and Q. □

To understand the idea of our algorithm, a 2D version is first presented. Figure 1(i) shows two non-overlapping convex polygons P and Q. Note that Definition 1 and Lemma 1 also hold for the 2D case, with the term "polytope" being replaced by "convex polygon".

Briefly, the algorithm works as follows. Given two convex polygons P and Q, initially a unit vector $\mathbf{S_0}$ is chosen and a supporting vertex $\mathbf{p_0}$ of P in the direction $\mathbf{S_0}$ is found. Similarly, a supporting vertex $\mathbf{q_0}$ of Q in the direction $-\mathbf{S_0}$ is found. Then the following criterion is tested. By Lemma 1, with $i = 0$, P and Q do not collide if

$$\mathbf{S_i} \cdot (\mathbf{q_i} - \mathbf{p_i}) \geq 0. \tag{1}$$

Any vector $\mathbf{S_i}$ satisfying the above condition is called a *separating vector* of P and Q, or just a separating vector since P and Q are often clear from the context. Note that we consider the case where $\mathbf{S_0} \cdot (\mathbf{q_0} - \mathbf{p_0}) = 0$ to be non-collision, although in this case P and Q may touch each other. A *separating vector* $\mathbf{w}$ of P and Q has the property that $\mathbf{w} \cdot (\mathbf{q}' - \mathbf{p}') \geq 0$ for any $\mathbf{p}' \in P$ and any $\mathbf{q}' \in Q$.

In general, if the above test fails for $\mathbf{S_i}$, P and Q may still not collide. In this case we find a new direction $\mathbf{S_{i+1}}$ from $\mathbf{S_i}$ by

$$\mathbf{S_{i+1}} = \mathbf{S_i} - 2(\mathbf{r_i} \cdot \mathbf{S_i})\mathbf{r_i}, \quad i = 0, 1, \ldots, \tag{2}$$

where $\mathbf{r_i} = (\mathbf{q_i} - \mathbf{p_i})/\|(\mathbf{q_i} - \mathbf{p_i})\|$. See Figure 1(iii). Note that $\mathbf{S_{i+1}}, \mathbf{S_i}$, and $\mathbf{r_i}$ lie on the same plane, and the angle between $\mathbf{S_{i+1}}$ and $\mathbf{S_i}$ is bisected by a vector perpendicular to $\mathbf{r_i}$.

This choice of $\mathbf{S_{i+1}}$ from $\mathbf{S_i}$ is based on the following observation. Consider two non-intersecting circular disks P and Q in the plane. See Figure 1(ii). It can be verified that if $\mathbf{S_0}$ is not a separating vector, the $\mathbf{S_1}$ given by (2) is a separating vector. (This argument is also true of two non-intersecting balls in 3D.) So in the general 2D case we choose $\mathbf{S_{i+1}}$ by (2) in the hope that the $\mathbf{S_{i+1}}$ thus chosen converges quickly to some separating vector, provided that P and Q do not collide.

If (1) does not hold and collision conditions (to be given later) are not satisfied, the above procedure is repeated. The first $\mathbf{S_k}$ that satisfies (1) is a separating vector of P and Q, and k is the number of iterations performed by the algorithm.

This algorithm works exactly the same way in 3D case. It is proved in the next section that if the two polytopes do not collide and the condition (1) does not hold, $\mathbf{S_i}$ will get closer to any fixed separating vector by each iteration. In each time frame, if the two polytopes do not collide, the separating vector and the two supporting vertices found are cached. The separating vector is used as the initial vector $\mathbf{S_0}$ in the next time frame. As objects usually do not move swiftly in virtual environment, so this vector is likely to be the separating vector in the next time frame or as an initial vector it can help get a report on collision more quickly. Similarly, the supporting vertices found in the previous time frame are used as initial points to search for the new supporting vertices in the next time frame. Because of convexity, local search is sufficient to locate the supporting vertices. Therefore the separating vector searching step runs in expected constant time due to temporal and geometric coherences.

Conditions for reporting collision when the two polytopes collide will be discussed in a later section.

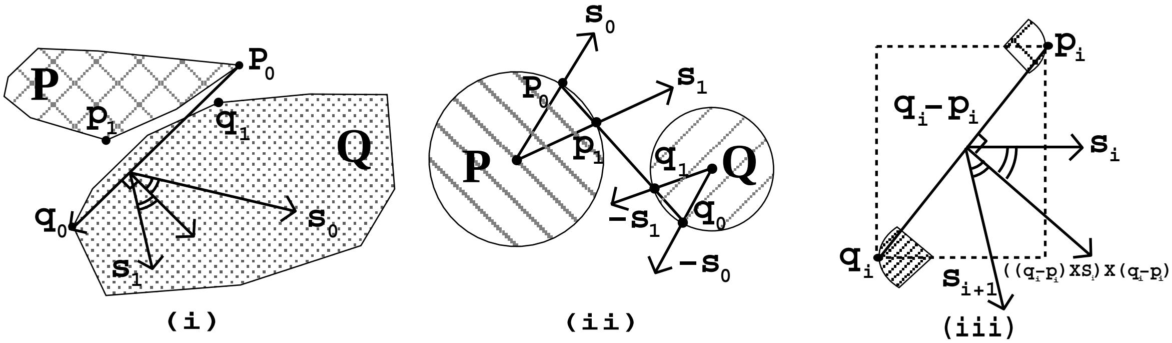

Figure 1: Searching for a separating vector (i) the idea (ii) in the case of circles (iii) choosing $\mathbf{S}_{i+1}$.

2.3 Searching for Supporting Vertices

The searching algorithm outlined in [1] is used to find a supporting vertex $\mathbf{p}_i$ on P and $\mathbf{q}_i$ on Q, with respect to $\mathbf{S}_i$ and $-\mathbf{S}_i$, respectively. In the search the current vertex $\mathbf{p}'$ on P is compared to its neighboring vertices to see if $\mathbf{S}_i \cdot \mathbf{p}'$ is the largest. If yes, the current vertex is a supporting vertex; if not, this vertex is replaced by a neighboring vertex $\mathbf{p}''$ with the largest $\mathbf{S}_i \cdot \mathbf{p}''$. This process is repeated until a supporting vertex is found. Notice that the supporting vertex may not be unique but this does not affect our algorithm. Because of convexity, this search can always find a supporting vertex eventually. If we assume that polytopes move slowly between time frames (which is usually the case in virtual environment), then the initial vertex for the search is close to the required supporting vertex usually. So empirically the searching step takes expected constant time because the search is performed locally on the surface of the polytopes. This has been verified by experiments. A supporting vertex $\mathbf{q}$ on Q can be found similarly.

In implementaton, there is no need to transform each vertex of polytope P or Q from its defining coordinate system to the world coordinate system and then take the dot product with $\mathbf{S}_i$ in order to find a supporting vertex. Instead, a more efficient way is to transform vector $\mathbf{S}_i$ to the defining coordinate system of the polytope by the inverse of the rotation matrix of the polytope, and the search is performed in the defining coordinate system. After a supporting vertex is found, it is transformed to the world coordinate system. Thus only two coordinate transformations are required for locating each supporting vertex.

2.4 Preprocessing

In a virtual environment, most collision detection algorithms use bounding boxes as the first step to eliminate non-interference polytopes. When bounding boxes of polytopes overlap for the first time, we can choose $\mathbf{S}_0 = (\mathbf{q}_c - \mathbf{p}_c)/||\mathbf{q}_c - \mathbf{p}_c||$ where $\mathbf{p}_c$ and $\mathbf{q}_c$ are the *centroids* of P and Q respectively. A centroid can be approximated by the average of all vertices of the polytope. We choose this initial $\mathbf{S}_0$ because the separating vector is likely to be close to this direction. Then an arbitrary vertex can be used as an initial vertex for searching supporting vertices of P and Q.

For better efficiency, we pre-compute supporting vertices in a number of pre-defined directions and store them in a 2D table. Then, for any given direction $\mathbf{S}_0$, a supporting vertex in the table with the direction close to $\mathbf{S}_0$ is retrieved in constant time and is used as the initial vertex to search for a supporting vertex with respect to $\mathbf{S}_0$. The larger is the size of this 2D table, the better approximation does this initial point provide, and the more quickly does this searching algorithm locate a supporting vertex. A table of size 8 × 16 is used in our implementation.

3 Proof of Convergence

Lemma 2: *If polytopes P and Q do not collide and* $\mathbf{S}_i \cdot \mathbf{r}_i < 0$ *in the i-th searching step, then for any separating vector* $\mathbf{w}$ *of P and Q,*

$$\mathbf{S}_{i+1} \cdot \mathbf{w} > \mathbf{S}_i \cdot \mathbf{w}, \quad i = 0, 1, \ldots. \tag{3}$$

Proof: By Eqn. (2),

$$\mathbf{S}_{i+1} \cdot \mathbf{w} = \mathbf{S}_i \cdot \mathbf{w} - 2(\mathbf{r}_i \cdot \mathbf{S}_i)(\mathbf{r}_i \cdot \mathbf{w}).$$

Since $\mathbf{r}_i \cdot \mathbf{w} > 0$ as $\mathbf{w}$ is a separating vector,

$$\mathbf{S}_{i+1} \cdot \mathbf{w} - \mathbf{S}_i \cdot \mathbf{w} = -2(\mathbf{r}_i \cdot \mathbf{S}_i)(\mathbf{r}_i \cdot \mathbf{w}) > 0. \quad \Box$$

Hence if the two polytopes do not collide and $\mathbf{S}_i$ is not a separating vector, then $\mathbf{S}_{i+1}$ given by Eqn. (2) is closer to any separating vector $\mathbf{w}$ than $\mathbf{S}_i$ is, since by Lemma 2 the angle between $\mathbf{S}_{i+1}$ and $\mathbf{w}$ is smaller than the angle between $\mathbf{S}_i$ and $\mathbf{w}$.

Another property of the algorithm is that if the pair $\mathbf{p}_i$ and $\mathbf{q}_i$ appear in two consecutive steps i.e. $\mathbf{r}_{i+1} = \mathbf{r}_i$, then P and Q do not collide, as indicated by the following lemma.

Lemma 3: *If* $\mathbf{S}_i \cdot (\mathbf{q}_i - \mathbf{p}_i) < 0$ *and* $\mathbf{p}_{i+1} = \mathbf{p}_i$, $\mathbf{q}_{i+1} = \mathbf{q}_i$, *then* $\mathbf{S}_{i+1} \cdot \mathbf{r}_{i+1} > 0$, *i.e. P and Q do not collide.*

Proof: Since

$$\begin{aligned}\mathbf{S}_{i+1} \cdot \mathbf{r}_{i+1} &= \mathbf{S}_i \cdot \mathbf{r}_{i+1} - 2(\mathbf{r}_i \cdot \mathbf{S}_i)(\mathbf{r}_i \cdot \mathbf{r}_{i+1}) \\ &= \mathbf{S}_i \cdot \mathbf{r}_i - 2(\mathbf{r}_i \cdot \mathbf{S}_i)(\mathbf{r}_i \cdot \mathbf{r}_i) = -\mathbf{S}_i \cdot \mathbf{r}_i > 0,\end{aligned}$$

by Lemma 1, P and Q do not collide. □.

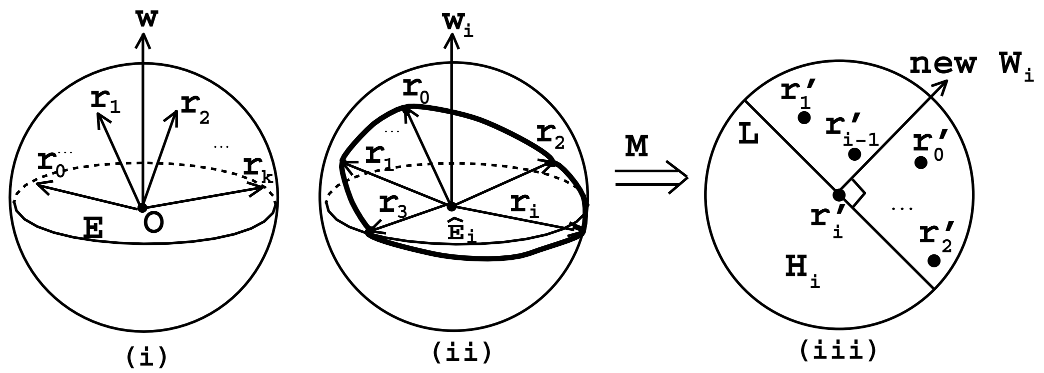

Figure 2: Determine the existence of a separating vector.

4 Collision Condition

4.1 Minkowski Sum

It can be shown [15] that P and Q collide iff $O \in M$ where O is the origin and $M = Q + (-P) = \{\mathbf{q} - \mathbf{p} \mid \mathbf{p} \in P, \mathbf{q} \in Q\}$ is the Minkowski sum of P and $-Q$. So, as M is convex, if the origin is outside M there exists $\mathbf{w}$ such that

$$\mathbf{m} \cdot \mathbf{w} \geq 0 \quad \text{for all} \quad \mathbf{m} \in M \qquad (4)$$

and this $\mathbf{w}$ is a separating vector. Conversely, $\mathbf{w}$ does not exist if there is a collision. Geometrically, Eqn. (4) implies that there exists a separating plane E passing through the origin such that all the points $\mathbf{r}_0, \ldots, \mathbf{r}_k$ on the unit sphere lie on one side of the plane E (see Figure 2(i)), where $\mathbf{r}_i = \mathbf{m}_i/||\mathbf{m}_i|| = (\mathbf{q}_i - \mathbf{p}_i)/||\mathbf{q}_i - \mathbf{p}_i||$. Therefore, if it is not possible to find such a plane E, then the two polytopes collide.

4.2 Existence of a Separating Plane

When a point $\mathbf{r}_i$ is added, an incremental algorithm is used to find a plane E_i with normal vector $\mathbf{w}_i$ such that $\mathbf{r}_0 \ldots, \mathbf{r}_i$ all lie on the positive half space of E_i. Initially, $\mathbf{w}_1$ is chosen to be the midpoint of $\mathbf{r}_0$ and $\mathbf{r}_1$. At the i-th iteration, if $\mathbf{r}_i \cdot \mathbf{w}_{i-1} > 0$, then $\mathbf{r}_i$ also lies on the positive half space of E_{i-1}, so we set $\mathbf{w}_i = \mathbf{w}_{i-1}$; otherwise, $\mathbf{r}_i$ must be one of the boundary points on the convex hull (a spherical polygon) formed by $\mathbf{r}_0, \ldots, \mathbf{r}_i$ on the surface of the sphere (see Figure 2(ii)). If there exists a plane $\hat{E}_i$ passing through the origin such that all the above points lie on one side of $\hat{E}_i$, then we can always rotate $\hat{E}_i$ into a plane $\tilde{E}_i$ such that $\tilde{E}_i$ touches $\mathbf{r}_i$ and all the points $\mathbf{r}_j$, $j = 0, 1, \ldots, i$, are on one side of $\tilde{E}_i$.

Let H_i denote the plane passing through the origin with normal vector $\mathbf{r}_i$. Then project the $\mathbf{r}_0, \ldots, \mathbf{r}_i$ along the vector $\mathbf{r}_i$ into points $\mathbf{r}'_0, \ldots, \mathbf{r}'_i$ on the plane H_i. If there exists a line L on the plane H_i that passes through the origin such that all the points $\mathbf{r}'_0, \ldots, \mathbf{r}'_i$ on H_i lie on one side of the line L, then $\mathbf{w}_i$ is taken to be the vector on the plane H_i perpendicular to L (see Figure 2(iii)). Conversely, if such a line L does not exist, then there does not exist a vector $\mathbf{w}_i$ such that Eqn. (4) is satisfied; that is, the two polytopes collide. Note that the existence of the line L can be determined in $O(i)$ time.

4.3 Termination

In the above searching process, by Lemma 3, if $\mathbf{m}_i = \mathbf{q}_i - \mathbf{p}_i \in Q - P$ repeats itself in two consecutive steps, P and Q do not collide. However, if $\mathbf{m}_i$ reoccurs after more than one steps before Eqn. (1) is satisfied, we cannot conclude that P and Q do not collide. In this case, in order to prevent the algorithm from running without stop, we set $\mathbf{S}_{i+1} = \mathbf{w}_i$ which is found in subsection 4.2. Then the vector $\mathbf{m}_{i+1} = \mathbf{q}_{i+1} - \mathbf{p}_{i+1}$ thus found with $\mathbf{S}_{i+1} = \mathbf{w}_i$ has the following property.

Lemma 4: *If $\mathbf{m}_{i+1} = \mathbf{m}_j$ for some j, $0 \leq j \leq i$, then $\mathbf{S}_{i+1} = \mathbf{w}_i$ is a separating vector of P and Q, that is, P and Q do not collide.*

Proof: Since

$$\mathbf{S}_{i+1} \cdot \mathbf{m}_{i+1} = \mathbf{w}_i \cdot \mathbf{m}_j \geq 0,$$

by Lemma 1, P and Q do not collide. □

Lemma 4 implies that either the algorithm stops with $\mathbf{S}_{i+1}$ being a separating vector or the $\mathbf{m}_{i+1}$ is a new vertex of M that has not been visited before. This gaurantees that the total number of vertex pairs repeated during the search is at most the number of vertices in M. So the algorithm will terminate in a finite number of steps.

To summarize, the vector $\mathbf{S}_{i+1}$ is either generated from $\mathbf{S}_i$ by Eqn. (2) or set to be $\mathbf{w}_i$ when there is a reoccurrence of $\mathbf{m}_i = \mathbf{q}_i - \mathbf{p}_i$. For a sequence of vectors $\{\mathbf{S}_i\}$ thus defined, when $\mathbf{S}_i \cdot \mathbf{r}_i \geq 0$ for some i for the first time, we can conclude that $\mathbf{S}_i$ is a separating vector, and the polytopes P and Q do not collide. The polytopes P and Q collide if there does not exist $\mathbf{w}_i$ such that $\mathbf{w}_i \cdot \mathbf{r}_j \geq 0$, $j = 0, 1, \ldots, i$ for some i. Note that when $\mathbf{m}_i$ reoccurs in two consecutive steps, P and Q do not collide by Lemma 3. Note that Lemma 3 and Lemma 4 are used because of numerical errors in implementation. For simplicity, they do not appear in the pseudo code of the algorithm (see the appendix).

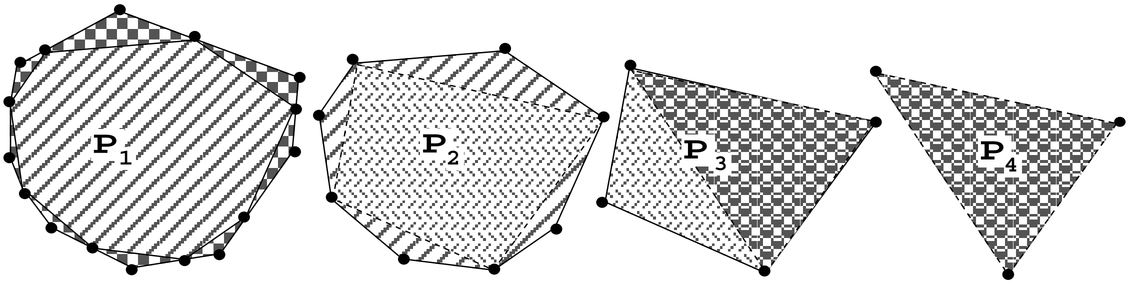

Figure 3: A hierarchical representation of a polygon.

5 Complexity

As pointed out earlier, the dynamic version of our algorithm makes use of time coherence between successive frames so to run in expected constant time. Let the number of vertices in P and Q be n and m respectively. Without making use of coherence, our implementation uses $O(n+m)$ time to search for new supporting vertices in P and Q, and $O(i)$ time to detect collision at the i-th iteration. So the worst time complexity is $O((n+m+k)*k)$, where k is the total number of iterations performed.

The time complexity for searching a new supporting vertex in P can be reduced to $O(\log n)$ with $O(n)$ preprocessing time, using the hierarchical representation of polytopes [12]. A hierarchical representation of polytope P with vertex set $\mathbf{V}(P)$ is defined as a sequence of polytopes $\text{hier}(P) = \{P_1, \ldots, P_h\}$ such that

(i) $P_1 = P$ and P_h is a simplex;
(ii) $P_{i+1} \subset P_i$, for $1 \leq i < h$;
(iii) $\mathbf{V}(P_{i+1}) \subset \mathbf{V}(P_i)$, for $1 \leq i < h$; and
(iv) the vertices in $\mathbf{V}(P_i) - \mathbf{V}(P_{i+1})$ form an independent set in P_i for $1 \leq i < h$.

An example of hierarchical representation in 2D case is shown in Figure 3. Note that this representation P is not unique. It is proved in [12] that the height of the hierarchical representation, h, is $O(\log n)$. Using this representation, we can find a supporting vertex of polytope P in direction $\mathbf{S}$ by first finding the supporting vertex $\mathbf{v}_\mathrm{h}$ of polytope P_h, which takes constant time since P_h is a simplex. Then local search is used to find the supporting vertex $\mathbf{v}_\mathrm{i}$ of P_i in direction $\mathbf{S}$ starting from $\mathbf{v}_{\mathrm{i+1}}$, for $i = h-1, h-2, \ldots, 1$.

Lemma 5: *For $i = 1, \ldots, h-1$, let $\mathbf{v}_{\mathrm{i+1}}$ be a supporting vertex of P_{i+1} with respect to direction $\mathbf{S}$. Then either $\mathbf{v}_{\mathrm{i+1}}$ is also a supporting vertex P_i with respect to direction $\mathbf{S}$ or a supporting vertex $\mathbf{v}_\mathrm{i}$ of P_i with respect to $\mathbf{S}$ is a neighbor of $\mathbf{v}_{\mathrm{i+1}}$ in P_i.*

Proof: Let $\mathbf{v}_\mathrm{i}$ be a supporting vertex of P_i with respect to $\mathbf{S}$. Since $P_{i+1} \subset P_i$, $\mathbf{S}\cdot\mathbf{v}_\mathrm{i} \geq \mathbf{S}\cdot\mathbf{v}_{\mathrm{i+1}}$. When $\mathbf{S}\cdot\mathbf{v}_\mathrm{i} = \mathbf{S}\cdot\mathbf{v}_{\mathrm{i+1}}$, clearly, $\mathbf{v}_{\mathrm{i+1}}$ is also a supporting vertex of P_i with respect to $\mathbf{S}$.

When $\mathbf{S}\cdot\mathbf{v}_\mathrm{i} > \mathbf{S}\cdot\mathbf{v}_{\mathrm{i+1}}$, we must have $\mathbf{v}_\mathrm{i} \notin P_{i+1}$; for otherwise $\mathbf{v}_\mathrm{i}$ would be a supporting vertex in P_{i+1}, instead of $\mathbf{v}_{\mathrm{i+1}}$. Suppose $\mathbf{v}_\mathrm{i}$ is not a neighbor of $\mathbf{v}_{\mathrm{i+1}}$ in P_i. By property (iv) of the hierarchical representation, all the neighbors of $\mathbf{v}_\mathrm{i}$ are vertices of P_{i+1}, and these neighbors do not include $\mathbf{v}_{\mathrm{i+1}}$. Therefore the open line segment $\overline{\mathbf{v}_\mathrm{i}\mathbf{v}_{\mathrm{i+1}}}$ has nonempty intersection with the polytope P_{i+1}. Let a point in this nonempty intersection $\overline{\mathbf{v}_\mathrm{i}\mathbf{v}_{\mathrm{i+1}}} \bigcap P_{i+1}$ be $\mathbf{v}_\lambda = (1-\lambda)\mathbf{v}_\mathrm{i} + \lambda\mathbf{v}_{\mathrm{i+1}}$ for some λ with $0 < \lambda < 1$. Then

$$\mathbf{S}\cdot\mathbf{v}_\lambda = (1-\lambda)\mathbf{S}\cdot\mathbf{v}_\mathrm{i} + \lambda\mathbf{S}\cdot\mathbf{v}_{\mathrm{i+1}} > \mathbf{S}\cdot\mathbf{v}_{\mathrm{i+1}}.$$

This contradicts that $\mathbf{v}_{\mathrm{i+1}}$ is a supporting vertex of P_{i+1} with respect to $\mathbf{S}$. Hence $\mathbf{v}_\mathrm{i}$ is a neighbor of $\mathbf{v}_{\mathrm{i+1}}$ in P_i. □

Since $h = O(\log n)$, a supporting vertex of $P = P_1$ can be found in $O(\log n)$ time, assuming that the degrees of vertices in P_i are bounded by a constant.

Besides, it takes constant time to check whether a pair of supporting vertices has been visited previously in the algorithm. The method is to keep track of a 2D array with each entry being a counter for a pair of vertices. Initially all the entries are reset to zero. There is also a variable called *timestamp*, which is incremented every time the collision detection algorithm is called. During the search for a separating vector, if the counter for a pair of supporting vertices is not equal to the timestamp, that counter is set to the timestamp; if it is equal to the timestamp, the pair has been visited before. When the maximum limit for the counter is reached, which is the maximum long integer of the language used, all the entries are reset to zero.

Hence the worst case running time of the separating vector searching algorithm can be reduced to $O((\log n + \log m + k) * k)$. So far the only upper-bound known to us for k is $O(mn)$. However, with temporal coherence takes effect in virtual environment, it is found empirically that k is very small even for very large n. For an ellipsoid-shaped polytope, $k < 25$ when $n < 1000$. The empirical running time of this algorithm in a dynamic environment is almost constant.

6 Experiments

Experiments have been carried out to investigate the number of the searching steps k for polytopes with different number of vertices n. The simulation uses 500 polytopes of the same number of vertices moving in a closed environment.

Polytopes of three different shapes are used: ellipsoid, a thin rod, and flat plate, obtained by randomly sampling points on the surface of an ellipsoid, a thin rod, and a flat plate, respectively. They provide a variety of different shapes for testing. Each object has its translational velocity equal to 5% of its radius and rotational velocity 10 degrees per time frames. When there is a collision between two polytopes, their rotational and translational velocities are

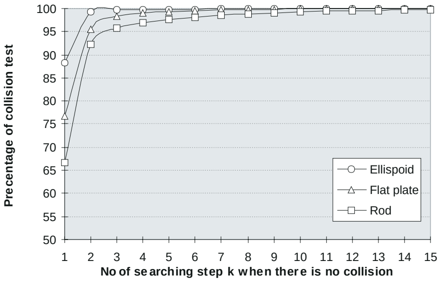

Figure 4(i)

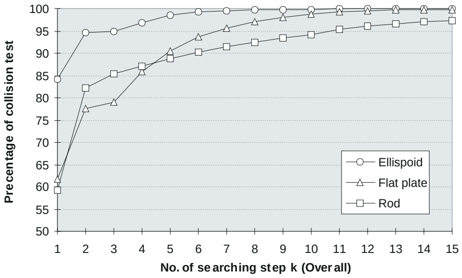

Figure 4(ii)

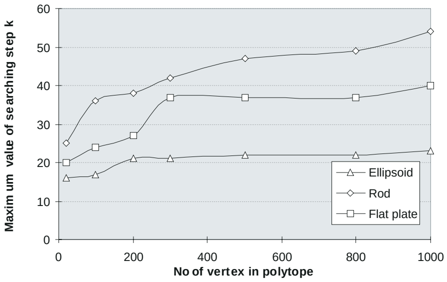

Figure 4(iii)

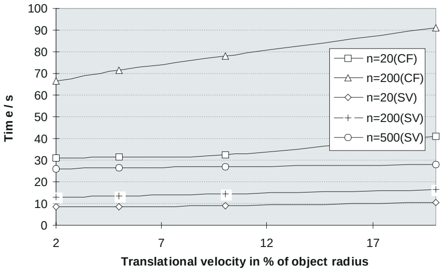

Figure 4(iv)

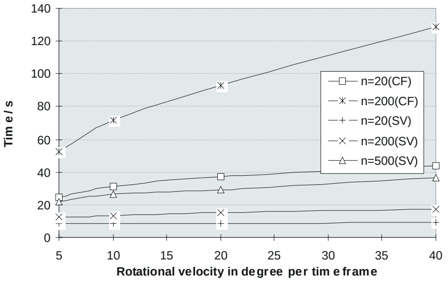

Figure 4(v)

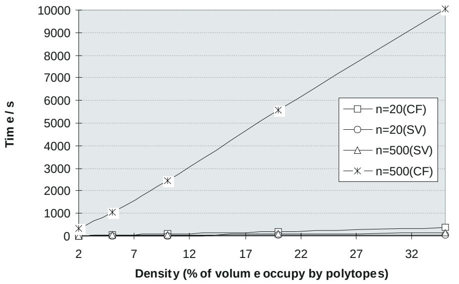

Figure 4(vi)

reversed. In the experiment, a precomputed 8×16 table for the supporting vertices is used as explained in section 2.4.

Figure 4(i) measures the value of k when collision test is called between non-colliding objects of 500 vertices. This collision test is called only when the tightest rectangular bounding boxes (as found in [1]) of two polytopes overlap. The results show that more than 95% of non-colliding objects are identified in first three steps for all three shapes. Moreover, for the case of ellipsoid more than 99% of non-colliding objects can be identified in the first four steps. Figure 4(ii) measures the value of k when collision test is called for both colliding and non-colliding objects of 500 vertices. The results show that on average more than 80% of collision tests can be completed within first three searching steps. Moreover, for polytopes of different number of vertices, a similar curve to that in Figure 4(i), Figure 4(ii) is obtained (not shown in the figure). This indicates that the algorithm runs in almost expected constant time. We also noticed that there are reoccurences of supporting vertices during the search for separating planes in less than 0.1% of collision tests for the polytopes which do not collide.

In Figure 4(iii), the maximum value of k for each case recorded is shown. From this figure, the maximum value of k is around 22 for ellipsoid, increases slightly from 20 to 40 for flat plate and increases from 25 to 55 for rod when n increases from 10 to 1000. It is noticed that the algorithm performs best for ellipsoid-shaped objects, and becomes less efficient for objects of plate-shape or rod-shape. The results also indicate that even in the worst example we construct as for thin rods and flat plates, the maximum value of k is small as compared to n. Besides, it is noted that this worst case value of k happens very rarely ($\leq 0.01\%$ of collision test) in the experiments. This explains why our algorithm runs significantly faster than others on average as shown in the next experiment, especially in virtual environments where we can make use of temporal coherence.

We have compared our algorithm with the closest features tracking algorithm, which is the fastest algorithm so far [1]. To measure the performance of our algorithm, the lowest layer of the I_COLLIDE [1] source code, which detects collision between two polytopes using the closest feature tracking algorithm (CF), is replaced by our separating vector algorithm (SV). The simulation is done on SGI/Indy machine (R4600), with a total of 100 polytopes of the above three shapes and the same number of vertices in the environment. Figure 4(iv) shows that, when the translational velocity is changed from 2% to 20% of object radius per time frame, the simulation time of SV algorithm increases slightly; however, the simulation time of CF algorithm increases substantially. That is because when the translational velocity increases, CF algorithm needs more time to locate the closest points between polytopes and travel from one feature to another, while finding a supporting vertex is faster in SV algorithm. Moreover, the I_COLLIDE library needs to call another linear programming algorithm when there is a recycling of features. The simulation time for CF algorithm when $n = 500$ is around 1000 seconds so it is not included in Figure 4(iv) and Figure 4(v). As a result, for velocity that is 20% of object radius per time frame and the number of vertices of each polytope $n = 500$, nearly 28 times speedup by SV algorithm is achieved.

Figure 4(iv) shows that, when the rotational velocity is increased from 5 degrees to 40 degrees per time frame, SV algorithm takes only a little longer time, while the CF algorithm takes substantially longer time. Here the set up is the same as above.

Lastly, Figure 4(v) shows the comparison when only the density of the environment changes. Again, SV algorithm is faster and more efficient than CF algorithm in all cases.

7 Conclusion

We have proposed an efficient exact collision detection algorithm for polytopes in virtual environments. The algorithm is based on a simple technique to quickly locate a separating plane between two polytopes if they do not collide, or otherwise test some simple conditions to report collision. Our algorithm is fast and simple to implement. Taking advantage of geometric and temporal coherences in a dynamic environment, our algorithm uses caching, preprocessing, and local search to run in expected constant time. These results have been verified by experiments.

As the contact points between two objects when they collide provide useful information for impulse computation, one of the remaining research problems is to consider reporting efficiently the contact points between colliding objects in our algorithm.

Also, we noticed that the number of searching steps k for some polytopes is far greater than average. So another problem we are investigating is to characterize those polytopes which our algorithm has to take many search steps to process.

References

1. D. J. Cohen, M.C. Lin, D. Manocha, and M. Ponamgi, I-Collide: An interactive and exact collision detection system for large-scale environments, *Proceeding of Symposium of Interactive 3D Graphics*, pp. 189-196, 1995.
2. M. Moore and J. Wilhelms, Collision detection and response for computer animation, *Computer Graphics*, Vol. 22, No. 4, pp. 289-298, 1988.
3. W. Thibault and B. Naylor, Set operations on polyhedra using binary space partitioning trees, *ACM Computer Graphics*, 4, pp. 153-162, 1987.
4. A. Foisy, V. Hayward, and S. Aubry, The use of awareness in collision prediction, *International Conference on Robotics and Automation*, pp. 338-343. IEEE, 1990.
5. A. Smith, Yoshifumi Kitamu, Haruo Takemura, and Fumio Kishino, A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion, *Virtual Reality Annual International Symposium*, pp. 136-145, IEEE, 1995.
6. A. Garcia-Alonso, N. Serrano and J. Flaquer, Solving the collision detection problem, *IEEE Computer Graphics and Applications*, 13(3), pp. 36-43, 1994.
7. Y. Yang and N. Thalmann, An improved algorithm for collision detection in cloth animation with human body, *First Pacific Conference on Computer Graphics and Application*, pp. 237-251, 1993.
8. M. Lin and J. Canny, Efficient collision detection for animation, *Proceedings of the Third Eurographics Workshop on Animation and Simulation*, Cambridge, 1991.
9. W. Bouma and G. Vanecek, Collision detection and analysis in a physical based simulation, *Proceedings of Eurographics Workshop on Animation and Simulation*, pp. 191-203, September, 1991.
10. D. Baraff, Curved surfaces and coherence for non-penetrating rigid body simulation, *Computer Graphics*, Vol. 24, No. 4, pp. 19-28, 1990.
11. Rich Rabbitz, Fast collision detection of moving convex polyhedra, *Graphics Gem IV*, AP Professional, pp. 83-109, 1994.
12. D.P. Dobkin and D.G. Kirkpatrick, A linear algorithm for determining the separation of convex polyhedra, *Journal of Algorithms* Vol. 6, pp. 381-392, 1985.
13. M. C. Lin, *Efficient Collision Detection for Animation and Robotics*, PhD thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, December 1993.
14. Philip M. Hubbard, Collision Detection for Interactive Graphics Applications, *IEEE Transactions on Visualization and Computer Graphics*, Vol. 1, No. 3, pp. 218-228, 1995.
15. E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, A fast procedure for computing the distance between complex object in three-dimensional space, *IEEE Journal of Robotics and Automation*, 4(2):193-203, 1988.
16. Elmar Schomer and Christian Thiel, Efficient collision detection for moving polyhedra, *Proceedings of the Eleventh Annual Symposium on Computational Geometry*, pp. 51-60, 1995.
17. M. Ponamgi, D. Manocha, and M. Lin, Incremental algorithms for collision detection between solid models, *Proceedings of ACM/Siggraph Symposium on Solid Modeling*, pp. 293-304, 1995.
18. R. T. Rockafellar, *Convex Analysis.* Princeton University Press, 1970.
19. B. Chazelle and D. Dobkin, Detection is easier than computation, *ACM Symposium on Theory of Comput.*, 12 , pp. 146-153, 1980.
20. S. Gottschalk, M. Lin and D. Manocha, OBB-Tree: A Hierarchical Structure for Rapid Interference Detection, to appear in *Proceedings of SIGGRAPH '96.*

APPENDIX

```
/*
  Detection collision between P and Q with rotation matrix Rp, Rq and
  translation Tp, Tq respectively. invRp and invPq are the inverse of the
  matrix Rp and Rq, respectively.
  Assume that the centers of P and Q are at the origin in the local coordinate
 system.
*/
Quick_Collision(P, Q, Rp, Rq, invRp, invRq, Tp, Tq)
{
  If (bounding boxes overlap for the first time) {
     S = <Tq - Tp>;    /* where <x> is the normalized vector of vector x */
     use S to get vertices p and q from the precomputed table;   /* section 2.4 */
  }
  else {
     retrieve S, p, q from cache;
  }
  k = 0;
  do {
     k++;
     p = SearchSupportVertex(P, p, invRp*S);     /* section 2.3 */
     q = SearchSupportVertex(Q, q, invRq*(-S));
     r_k = <(q*Rq + Tq) - (p*Rp + Tp)>;
     dp = dotproduct(S, r_k);
     if ( dp >= 0) {                             /* Lemma 1 */
        save S, p, q;                            /* cache the values */
        return non-collision;
     }
     if (r_k has appeared before) {
      S = w;                                   /* Lemma 4 */
     }
     else
     {
        if (k = 2)
           w = <r_1 + r_2>
        if ((dotproduct(w, r_k) < 0) and !FindSeparatePlane(r_k, w))
            return collision;                    /* section 4.2 */
        save r_k;
        S = S - 2*dp*r_k;                        /* Eqn. (2) */
     }
  } while (TRUE)
}
```