Detect Collision of Polytopes Using a Heuristic Search for Separating Vectors

Xueqing Li, Xiangxu Meng, C.Y.Wang
College of Computer Science and Technology of Shandong University
Jinan, Shandong, China,250100
liyou@jn-public.sd.cninfo.net
Wenping Wang, Kelvin Chung, Siu Ming Yiu
University of Hong Kong
Pokfulam Road, Hong Kong
wenping@csis.hku.hk

1 Nov., 2001

Abstract

Collision detection is fundamental to computer simulation of a physical environment, and has applications in computer graphics, CAD/CAM, and robotics. In this paper we propose an new method, which we call *HS-jump*, for collision detection for polytopes. HS-jump combines an efficient scheme to report collision for two colliding polytopes and a fast heuristic strategy to search for a *separating vector* of two separated polytopes; a *separating vector* is the normal vector of a separating plane of two disjoint polytopes. Due to the particular nature of search scheme used, HS-jump gets more efficient when the convex polyhedra are more sphere-shaped. Hence, in the case of applying HS-jump to convex polyhedra with a high number of vertices, HS-jump delivers the maximum efficacy if the objects are on average not very elongated.

KeyWords: Computational Geometry, Computer Graphics, Virtual Reality, Collision Detectioin

1 Introduction

The collision detection problem is to determine whether two moving objects collide or not at any moment. It is often sufficient, as a typical and efficient approach in practice, to consider whether the two polytopes intersect in any of a sequence of discrete time frames. This technique is applied especially when the motion path of an object is not prespecified but is defined interactively, such

as in virtual reality applications, as opposed to motion planning in robotics. Such a treatment has two implications: 1) Collision occurring between two consecutive frames may be missed; normally, the chance of such possible errors is reduced by sampling with a smaller time interval to get more frames. 2) Efficient collision detection can be achieved by exploiting the fact that the position and orientation of moving objects under consideration change little between consecutive time frames, which is called *between-frame coherence*.

Collision detection algorithms that exploit between-frame coherence are more efficient than repeated applications of a conventional polytope intersection algorithm in consecutive frames. To make use of between-frame coherence, one may compute a *witness*, such as a separating plane or a pair of closest points, whose existence confirms the separation (or non-collision status) of two polytopes in the current frame. Because of between-frame coherence, the *witness* from the current frame can be used in the next time frame to either facilitate the computation of a new witness or test quickly if the two polytopes in updated positions are still separated.

Another feature of collision detection algorithms is the type of output information. Some applications require only collision status of boolean type, i.e., *separated* or *colliding*. But it is often useful to return a separating plane or even a pair of closest features of two separated polytopes.

We will describe a new collision detection algorithm, called *HS-jump*, for 3D polytopes. HS-jump uses a fast heuristic to compute a separating plane of two separated polytopes; a formula that yields a separating vector for two separated sphere is extended into an iterative heuristic searching scheme for a separating vector of two general convex polyhedra. Colliding polytopes are detected efficiently by maintaining a spherical convex polygon to yield balanced time performances for both cases of separated and colliding polytopes. A pair of closet feature can also be computed with little extra computation within the framework of HS-jump. Furthermore, unlike some existing methods, the termination conditions of HS-jump are established rigorously.

2 Review

There are many techniques proposed in the literature to solve the problem of collision detection. One of them is spatial decomposition. For instances, Octrees [22], $k-d$ trees [14], BSP-trees [24], brep-indexes [29], tetrahedral meshes [14], and grids [10, 14] are examples of spatial decomposition. The idea of this technique is to divide the space occupied by the objects into cells. During the collision detection process, only objects in the same or nearby cells are checked. The other commonly used method is to represent an object by a hierarchy of bounding volumes [15, 16, 25]. In this multi-resolution representation, one can obtain increasingly more accurate approximations of the objects, until the exact geometry of the object is reached through the hierarchy. The detection starts with the highest level and can stop once a conclusion is reached without going into details of the objects every time. The choices for bounding volumes are usually spheres or axis-aligned bounding boxes

due to the simplicity in checking overlaps (intersections) for two such volumes. In addition, it is easy to transform these volumes as an object rotates and translates.

Another bounding volume that has become popular is the oriented bounding box (OBB), which surrounds an object with a bounding box (hexahedron with regular facets) whose orientation is arbitrary with respect to the coordinate axes. This volume has the advantage that it can, in general, yield a better (tighter) outer approximation of an object, as its orientation can be chosen to make the volume as small as possible. One such example is the publicly available "RAPID" system for performing collision detection among arbitrary polygonal models. The system uses a type of oriented bounding boxes, called "OBBTree", implemented by Ottschalk, Lin and Manocha [13]. The efficiency of this method is partially due to a fast algorithm for determining whether two oriented bounding boxes overlap. This algorithm is based on examining projections along a small set of "separating axes" and is claimed to be an order of magnitude faster than previous algorithms.

Another technique used by collision detection is the space-time bounds [15] and four-dimensional geometry [3]. The idea is to bound the positions of objects within the near future using a fourth dimension as time. Contacts can be pin-pointed exactly. However, these methods are restrictive in the sense that they usually require the motion to be pre-specified as a closed-form function of time. Hubbards space-time bounds [15] do not have such a requirement but rely on an assumption on the acceleration of the objects.

Using the techniques of computational geometry to study collision detection is also an important approach. One method is to utilize Voronoi diagrams [6, 19] to keep track of the closet features between pairs of objects. The popular system, I-COLLIDE [6], uses spatial and temporal coherence in addition to a "sweep-and-prune" technique to reduce the pairs of objects that need to be considered for collision. Although this software works well for many simultaneously moving objects, the objects are restricted to be convex. More recently, Ponamgi, Manocha, and Lin have generalized this work to include non-convex object [26]. Another method is to solve the collision detection by computing the intersection or the minimum distance. Using hierarchical representation, an $O(\log^2 n)$ algorithm is given in [8] for the polytope-polytope collision detection problem, where n is the number of vertices. Good theoretical and practical approaches based on the linear programming problem are also known [20, 27]. Minkowski difference and convex optimization techniques are used in [12] to compute the distance between convex protopes by finding the closest points.

Other variations of these techniques appear in [11, 1, 18, 21, 22, 28]. In [18], a method, based on bounding-volume hierarchies, for efficient collision detection for objects moving within highly complex environments is proposed by Klosowski. The choice of bounding volume is a "discrete orientation polytope" ($K-DOPs$). A $k-DOP$ is a convex polytope whose facets are determined by halfspaces of which the outward normals come from a small *fixed* set of k orientations. In [1], G. Barequet et al. use oriented bounding boxes to compute the hierarchical representation of surfaces for performing collision detection. Colin [28] gives a method to solve the collision detection of

surfaces by computing the distance between convex objects defined by $NURBS$ curves or patches. Brian Mirtich describes a method for fitting the pieces together which produces the V_Clip collision detection algorithm in [21]. This algorithm can rapidly and robustly locate pairs of closest features between polyhedral models. In [11] and [23], an image-based algorithm using rasterizing graphics hardware is proposed for collision detection between complex solids. In this method, the $3D$ collision detection problem is reduced to $1D$ interval testing problem along the $z-$axis. One drawback of this approach, inherent in all rasterizing graphics techniques, is the possibility of missing collision that occurs between samples. There are also some comparison studies on these algorithms. For example, in [4], Stephen Cameron has modified the algorithm of Gilbert, and he shows that his algorithm has $O(1)$ time cost under the reasonalbe assumptions.

Most of these methods are efficient in the case non-collision, but not very efficient in the case of collision. Furthermore the termination conditions of these methods are not established rigorously. In [5] a method, calld Q-Collide, based on seperating vectors is proposed for collision detection for polytopes. In this paper we improve the original method in [5] by establishing rigorous termination conditions, providing a deeper theoretical analysis, and presenting a more complete experimental results. A new collision detection system, called HS-jump, has been built on these new development.

3 The basic idea

A polytope is the intersection of a collection of (closed) half spaces in E^3, the 3D Euclidean space. We assume throughout in this paper that the polytopes considered have a finite number of vertices and are bounded. Let $\mathcal{P}$ be a bounded polytope in E^3. Let $V(\mathcal{P})$ denote the set of vertices of P. Let $(\mathbf{x}\cdot\mathbf{y})$ denote the inner product of 3D vectors $\mathbf{x}$ and $\mathbf{y}$. Given a vector $\mathbf{S}\neq 0$, a vertex $\mathbf{p}\in V(\mathcal{P})$ is called a *supporting vertex* of $\mathcal{P}$ with respect to $\mathbf{S}$ if $\mathbf{S}\cdot\mathbf{p}=\max\{\mathbf{S}\cdot\mathbf{x}|\mathbf{x}\in V(\mathcal{P})\}$; as a matter of fact, $\mathbf{S}\cdot\mathbf{p}=\max\{\mathbf{S}\cdot\mathbf{x}|\mathbf{x}\in\mathcal{P}\}$ for the supporting vertex $\mathbf{p}$. Such a supporting vertex $\mathbf{p}$ may not be unique for a given polytope $\mathcal{P}$ and vector $\mathbf{S}$.

Given a vector $\mathbf{S}\neq 0$ and two polytopes $\mathcal{P}$ and $\mathcal{Q}$, let $\mathbf{p}$ be a supporting vertex of $\mathcal{P}$ with respect to $\mathbf{S}$ and $\mathbf{q}$ a supporting vertex of $\mathcal{Q}$ with respect to $-\mathbf{S}$. The ordered pair $(\mathbf{p},\mathbf{q})$ is called a *supporting vertex pair* of $\mathcal{P}$ and $\mathcal{Q}$ with respect to $\mathbf{S}$. The pair $(\mathbf{p},\mathbf{q})$ may not be unique for a given vector $\mathbf{S}$ and polytopes $\mathcal{P}$ and $\mathcal{Q}$. See Figure 1.

The vector $\mathbf{q}-\mathbf{p}$ is called a *supporting vertex vector* of $\mathcal{P}$ and $\mathcal{Q}$ with respect to $\mathbf{S}$.

The polytopes $\mathcal{P}$ and $\mathcal{Q}$ are said to be *disjoint*, or *separated*, or *non-colliding* if $\mathcal{P}\cap\mathcal{Q}=\emptyset$. For two disjoint polytopes $\mathcal{P}$ and $\mathcal{Q}$ there exists a plane $\mathcal{L}$ such that $\mathcal{P}$ and $\mathcal{Q}$ are on the opposite sides of $\mathcal{L}$, and $\mathcal{P}\cap\mathcal{L}=\emptyset$ and $\mathcal{Q}\cap\mathcal{L}=\emptyset$. See Figure 2. The plane $\mathcal{L}$ is called a *separating plane* of $\mathcal{P}$ and $\mathcal{Q}$. A separating vector of $\mathcal{P}$ and $\mathcal{Q}$ is defined to be a normal vector $\mathbf{S}$ of $\mathcal{L}$ that points from the side of $\mathcal{P}$ to the side of $\mathcal{Q}$, i.e., $\mathbf{S}\cdot(\mathbf{y}-\mathbf{x})>0$ for any $\mathbf{x}\in\mathcal{P}$ and any $\mathbf{y}\in\mathcal{Q}$.

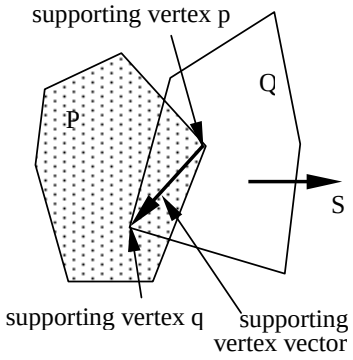

Figure 1: the supporting vertex pair

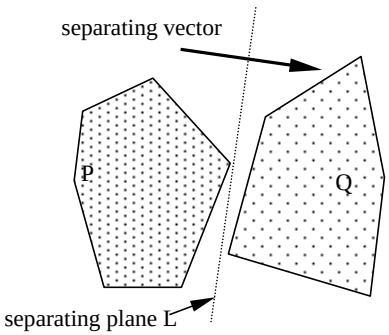

Figure 2: P and Q is disjoint

Theorem 1: *A vector* $\mathbf{S}$ *is a separating vector of polytopes* $\mathcal{P}$ *and* $\mathcal{Q}$ *if and only if* $\mathbf{S}\cdot(\mathbf{q}-\mathbf{p}) > 0$, *where* $(\mathbf{p}, \mathbf{q})$ *are a supporting vertex pair of* $\mathcal{P}$ *and* $\mathcal{Q}$ *with respect to* $\mathbf{S}$.

Proof: Firstly, we suppose that $\mathbf{S}$ is an seperating vector of polytopes $\mathcal{P}$ and $\mathcal{Q}$, by defining of $\mathbf{S}$, for any $x \in \mathcal{P}$ and $y \in \mathcal{Q}$, we have $S \cdot (y - x) > 0$. Since $p \in \mathcal{P}$ and $q \in \mathcal{Q}$, we have $S \cdot (q - p) > 0$.

Now suppose we have $S \cdot (q - p) > 0$, so $S \cdot q > S \cdot p$, since p and q are the supporting vertex of $\mathcal{P}$ and $\mathcal{Q}$ with respect to $\mathbf{S}$, by defining of supporting vectex, we have $S \cdot p \geq S \cdot x, x \in \mathcal{P}$, $-S \cdot q \geq -S \cdot y, y \in \mathcal{Q}$, then $S \cdot y \geq S \cdot q > S \cdot p > S \cdot x$, at last we have $S \cdot (y - x) > 0$, i.e. $\mathbf{S}$ is a seperating vector of polytopes $\mathcal{P}$ and $\mathcal{Q}$. This completes the proof.

Theorem 1 states a property of the separating vector that suggests solving the collision detection problem by searching for a separating vector, which, if exists, can be found by checking the condition $\mathbf{S}\cdot(\mathbf{q}-\mathbf{p}) > 0$. HS-jump is based on this idea and performs iterations to resolve the collision detection problem between two polytopes $\mathcal{P}$ and $\mathcal{Q}$ in a given frame. Within a finite number of steps, HS-jump can either find a separating vector, so declaring $\mathcal{P}$ and $\mathcal{Q}$ to be separated, or find $\mathcal{P}$ and $\mathcal{Q}$ to be colliding via an analysis of the supporting vertex pairs produced in all preceding iterations.

Briefly, HS-jump performs for each frame an iteration consisting of the following two major steps, until a definite collision status is determined. Suppose that the present frame is the i-th iteration.

(1) Generate a new candidate separating vector S_i, and its corresponding pair of supporting vertices $\mathbf{p}_i \in V(\mathcal{P})$ and $\mathbf{q}_i \in V(\mathcal{Q})$. (The rules of generating candidate separating vectors will be introduced in Section 4 and Section 6.) If $\mathbf{S}_i$ is a separating vector, i.e., $\mathbf{S}_i \cdot (\mathbf{q}_i - \mathbf{p}_i) > 0$, declare that $\mathcal{P}$ and $\mathcal{Q}$ do not collide in the present frame; otherwise, go to step (2).

(2) Use all the pairs of vertices $\{\mathbf{p}_j, \mathbf{q}_j\}$, $j = 0, 1, 2, \ldots, i$, that have been produced so far, to check if $\mathcal{P}$ and $\mathcal{Q}$ collide. (This checking will be explained in Section 5.) The outcome can be

collision or *undetermined*. If it is *collision*, declare collision between P and Q for the present frame; otherwise, set $i := i + 1$ and go to (1).

In the first step above, HS-jump attempts to detect the separation of two polytopes. The polytopes are declared to be separated if the current candidate separating vector can be confirmed to be a separating vector; so the outcome is either *separation* or *undetermined*. In Section 4 we will present a fast heuristic that is used in step 1 to generate new candidate separating vectors.

It is clearly not possible to find any separating vectors if the input polytopes intersect. Hence, with no prior knowledge about the collision status of the two input polytopes, we need an efficient check in step (2) to detect the collision of two colliding polytopes; the outcome is either *collision* or *undetermined*. This scheme will be presented in Section 5. When *undetermined* status is reached in both step 1 and step 2, HS-jump enters the next iteration.

Overall, there are four important issues regarding the correctness and efficiency of HS-jump:

(1) generation of new candidate separating vectors;

(2) efficient computation of supporting vertex pairs with respect to a given candidate separating vector;

(3) confirming collision by analyzing supporting vertex pairs;

(4) establishing correct termination conditions.

These issues will be discussed in Section 6 and Section 8, respectively.

4 Detecting separation

In this section we will introduce the scheme of generating candidate separating vectors in HS-jump. Suppose that two polytopes $\mathcal{P}$ and $\mathcal{Q}$ are given. Let $\mathbf{S}_0$ be an initial candidate separating vector. Let $(\mathbf{p}_i, \mathbf{q}_i)$ be a supporting vertex pair of $\mathcal{P}$ and $\mathcal{Q}$ with respect to the candidate separating vector $\mathbf{S}_i$ in the i-th iteration. Assume that $\mathbf{S}_i \cdot (\mathbf{q}_i - \mathbf{p}_i) \leq 0$, for otherwise we can declare $\mathcal{P}$ and $\mathcal{Q}$ to be separated and terminate HS-jump for the present frame. Denote $\mathbf{r}_i = (\mathbf{q}_i - \mathbf{p}_i)/|(\mathbf{q}_i - \mathbf{p}_i)|$; here we may assume $\mathbf{q}_i - \mathbf{p}_i \neq 0$, for otherwise, we will know that $\mathbf{p}_i \in \mathcal{P} \cap \mathcal{Q} \neq \emptyset$, i.e., $\mathcal{P}$ and $\mathcal{Q}$ collide. The vector $\mathbf{r}_i$ is called a *unit supporting vertex vector* of $\mathcal{P}$ and $\mathcal{Q}$ with respect to $\mathbf{S}_i$.

Finding quickly a separating vector for two separated polytopes is key to identifying their separation. There are two rules that are used in HS-jump to generate new candidate separating vectors. Suppose that a candidate separating vector $\mathbf{S}_i$ has been found not to be a separating vector, then the first rule of generating the new candidate separating vector $\mathbf{S}_{i+1}$ in the next frame is by the formula

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

For the sake of avoiding infinite looping of HS-jump, We will introduce later in Section 6 the second rule of generating a new candidate separating vector. In the rest of this section, we will present some properties of formula (1) for its justification.

Theorem 2: *Let $\mathcal{P}_s$ and $\mathcal{Q}_s$ be two disjoint spheres in E^3. Let $\mathbf{S}_0$ be an any nonzero vector which is not a separating vector of $\mathcal{P}_s$ and $\mathcal{Q}_s$. The vector $\mathbf{S}_1$ derived from $\mathbf{S}_0$ through formula (1) is a separating vector of $\mathcal{P}_s$ and $\mathcal{Q}_s$.*

Proof: Consider the sphere $M = \mathcal{Q}_s \oplus (-\mathcal{P}_s) = \{y - x | x \in \mathcal{Q}_s, y \in \mathcal{P}_s\}$, i.e. the Minkowski sum of $\mathcal{Q}_s$ and -$\mathcal{P}_s$. It is clear that, for any vector S, S is a separating vector of $\mathcal{P}_s$ and $\mathcal{Q}_s$ if and only if S is a separating vector of M and $R = O$, where $O = (0, 0)$ is the origin; furthermore, the supporting vertex vector of $\mathcal{P}_s$ and $\mathcal{Q}_s$ with respect to S is equal to the supporting vertex vector of R and M with respect to S. Hence, we just need to prove the conclusion of the theorem by considering R and M in place of $\mathcal{P}_s$ and $\mathcal{Q}_s$.

Since $\mathcal{P}_s$ and $\mathcal{Q}_s$ are separated, R and M are also separated. Therefore, the origin O is not contained in sphere M. Let m_0 be the supporting vertex vector of R and M with respect to S_0. Then m_0 can be regarded as a point on the boundary of sphere M. Let $\mathcal{L}$ be the line passing through the origin O and m_0. Let m_0 and m_1 between the two intersections of $\mathcal{L}$ and sphere M. Since S_0 is not a separating vector of R and M, m_1 is between m_0 and O on line $\mathcal{L}$. Figure 3 shows the sectional view of sphere M on the plane determined by the origin O, m_0, and vector S_0.

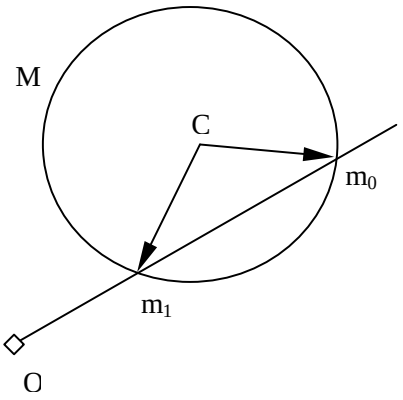

Figure 3: the proof of theorem 2

Let C denote the center of sphere M. Without the loss of generality, we may assume $S_0 = C - m_0$, since a common scaling factor to S_i and S_{i+1} in formula (1) is immaterial . Now the unit supporting vertex vector with respect to S_0 is $r_0 = m_0/|m_0|$. Since $(m_0 + m_1)/2 = (C \cdot r_0)r_0$, it follows that $m_1 = 2(C \cdot r_0)r_0 - m_0$. Thus, according to formula (1),

$$S_1 = S_0 - 2(S_0 \cdot r_0)r_0 = C - m_0 - 2[(C - m_0) \cdot r_0]r_0 = C - [2(C \cdot r_0)r_0 - m_0] = C - m_1.$$

Hence, the supporting vertex vector with respect to the next candidate separating vector $S_1 = C - m_1$ is m_1, i.e. S_1 is the normal vector of the tangent plane of M at m_1, and this tangent plane is clearly a separating plane of R and M. Hence, S_1 is a separating vector of R and M, and, according to the argument at the beginning of the proof, S_1 is also a separating vector of $\mathcal{P}_s$ and $\mathcal{Q}_s$. This completes the proof.

Theorem 2 asserts that HS-jump uses at most two iterations to produce a separating vector of two disjoint spheres. In addition, a result to be proved in Theorem 4 will show that, using formula (1), HS-jump also reports collision within two iterations for two colliding spheres. These properties provide useful insights into the behavior of the candidate separating vectors produced by formula (1), though they relate only to the theoretically interesting case of HS-jump being applied to spheres. As a matter of fact, one may regard formula (1) as a heuristic scheme that possesses the above properties, and expect naturally this scheme to perform efficiently when the input polytopes are nearly sphere-shaped. However, in order to devise a collision detection algorithm for general polytopes, we need study, theoretically or experimentally, if the merits of formula (1) are preserved in the general setting. For example, the next theorem states the convergence behavior of the S_i produced by (1) for two separated polytopes.

Theorem 3: *Let $\mathcal{P}$ and $\mathcal{Q}$ be two separated polytopes. Let $\hat{\mathbf{S}}$ be an arbitrary separating vector of $\mathcal{P}$ and $\mathcal{Q}$. Then the angle between $\hat{\mathbf{S}}$ and the $\mathbf{S}_i$ produced by (1) decreases monotonically as i increases.*

Proof: By formula (1), we known

$$S_{i+1} \cdot \hat{\mathbf{S}} = (S_i - 2(S_i \cdot r_i)r_i) \cdot \hat{\mathbf{S}} = S_i \cdot \hat{\mathbf{S}} - 2(S_i \cdot r_i)r_i \cdot \hat{\mathbf{S}}$$

since $\hat{\mathbf{S}}$ is a seperating vector , $r_i \cdot \hat{\mathbf{S}} > 0$, but S_i is not a seperating vector, so $S_i \cdot r_i < 0$, otherwise, we know that $\mathcal{P}$ and $\mathcal{Q}$ is disjoint and the algorithm ends, so we have $-2(S_i \cdot r_i)r_i \cdot \hat{\mathbf{S}} > 0$, at last we have $S_{i+1} \cdot \hat{\mathbf{S}} > S_i \cdot \hat{\mathbf{S}}$. This completes the proof.

Although Theorem 3 indicates that the vector $\mathbf{S}_i$ generated by (1) gets closer and closer to an arbitrary but fixed separating vector of two separated polytopes, there is no guarantee that the $\mathbf{S}_i$ produced by (1) alone will lead to termination of HS-jump in a finite number of steps. We will address this issue in Section 6 by introducing an auxiliary rule of obtaining $\mathbf{S}_{i+1}$ from $\mathbf{S}_i$.

5 Detecting collision

When two polytopes intersect, the attempt to search for a separating vector will eventually fail. To avoid spending much time searching for a separating vector in this case, we will discuss in this section the scheme used in HS-jump to detect the intersection between the polytopes.

Recall that $\mathcal{P} \cap \mathcal{Q} \neq \emptyset$ if and only if zero vector $0 \in \mathcal{Q} \oplus (-\mathcal{P}) \equiv \{\mathbf{y} - \mathbf{x} \mid \mathbf{x} \in \mathcal{P}, \mathbf{y} \in \mathcal{Q}\}$, i.e., the

zero vector is contained in the Minkowski sum of $\mathcal{Q}$ and $-\mathcal{P}$. On the other hand, the supporting vertex pair $(\mathbf{p}_i, \mathbf{q}_i)$ gives the vector $\mathbf{q}_i - \mathbf{p}_i \in \mathcal{Q} \oplus (-\mathcal{P})$. Denote $\mathbf{r}_j = (\mathbf{q}_j - \mathbf{p}_j)/|\mathbf{q}_j - \mathbf{p}_j|$, assuming $\mathbf{q}_j - \mathbf{p}_j \neq 0$. So, in each iteration of HS-jump, we test if the zero vector is contained by the convex hull $\mathcal{CH}_i$ of all the vectors $\mathbf{r}_j$, $j = 0, 1, \ldots, i$, i.e. all the unit supporting vertex vectors that have been produced so far. Since $\mathcal{CH}_i$ is a subset of $\mathcal{Q} \oplus (-\mathcal{P})$, if it is found that $0 \in \mathcal{CH}_i$, HS-jump can declare that $\mathcal{P}$ and $\mathcal{Q}$ collide; otherwise, HS-jump will continue to search for a separating vector.

Now the question is how to test efficiently if $0 \in \mathcal{CH}_i$. We will show that it takes at most $O(\log i)$ time to test if $0 \in \mathcal{CH}_i$ by checking if there exists an *open hemisphere* of S^2 that contains the set of points $\mathbf{r}_j$, $j = 0, 1, 2, \ldots, i$ on S^2, the unit sphere centered at the origin; an *open hemisphere* of S^2 is the intersection of S^2 and the open half space defined by a plane passing through the origin.

Let the $\mathbf{r}_j$, $j = 0, 1, \ldots, i$, be identified with points on the unit sphere S^2. Clearly, $0 \in \mathcal{CH}_i$ if and only if there does not exist an open hemisphere of S^2 that contains all the $\mathbf{r}_j$. The open hemisphere determined by a vector $\mathbf{r}$ be defined by $\mathcal{HS}(\mathbf{r}) = \{\mathbf{x} | \mathbf{x} \in S^2, \mathbf{x} \cdot \mathbf{r} > 0\}$. Then all the $\mathbf{r}_j$, $j = 0, 1, \ldots, i$, can be contained in an open hemisphere of S^2 if and only if $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j) \neq \emptyset$. Evidently, $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ forms a spherical convex polygon if $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j) \neq \emptyset$. Hence, with a new $\mathbf{r}_{i+1}$ being generated at iteration $i+1$, we just need update the spherical convex polygon $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ to obtain $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j)$, since $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j) = \bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j) \cap \mathcal{HS}(\mathbf{r}_{i+1})$.

There is a planar analogue of the above problem, that is, the problem of computing dynamically (or on-line) the intersection of a set of half planes. Based on the same idea as used in [17], and using a balanced binary tree, $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j)$ can be computed from $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ in $O(\log i)$ time. Let $\mathcal{L}(\mathbf{r})$ denote the plane with the normal vector $\mathbf{r}$ and passing through the origin, i.e., $\{\mathbf{x} \mid \mathbf{x} \cdot \mathbf{r} = 0\}$. When a new hemisphere $\mathcal{HS}(\mathbf{r}_{i+1})$ is added, the basic operation to form $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j)$ is to find the intersection points of the plane $\mathcal{L}(\mathbf{r}_{i+1})$ with the boundary of $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$; there are, in general, either none or two such intersection points. When $\mathcal{L}(\mathbf{r}_{i+1})$ and $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ intersect in two points, the sides of $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ need to be updated to obtain $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j)$; when they do not have any intersection points, either $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j) = \bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ or $\bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j) = \emptyset$, with the latter case indicating that the two polytopes $\mathcal{P}$ and $\mathcal{Q}$ intersect. All these cases are illustrated in Figure 4. We

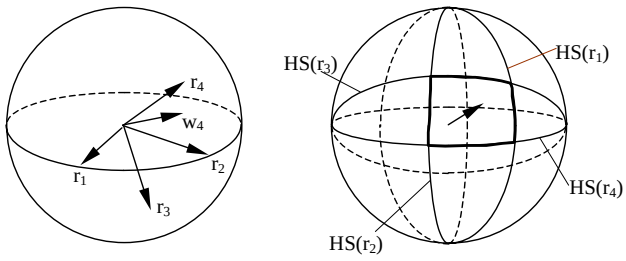

Figure 4: Compute the spherial convex polygon

end this section with one more property of the candidate separating vector $\mathbf{S}_i$ produced by (1).

Theorem 4: *Let $\mathcal{P}_s$ and $\mathcal{Q}_s$ be two intersecting spheres in E^3, including the case of one sphere containing the other. Let $\mathbf{S}_0$ be an arbitrary initial candidate separating vector. Let $\mathbf{S}_1$ be the next candidate separating vector generated from $\mathbf{S}_0$ using (1). Let $\mathbf{r}_0$ and $\mathbf{r}_1$ be the unit supporting vectors with respect to $\mathbf{S}_0$ and $\mathbf{S}_1$, respectively. Then $\mathbf{r}_0 = -\mathbf{r}_1$; consequently, $\mathcal{HS}(\mathbf{r}_0) \cap \mathcal{HS}(\mathbf{r}_1) = \emptyset$.*

Proof: We will use the similar argument to that used in the proof of Theorem 2, i.e. translating the problem regarding $\mathcal{P}_s$ and $\mathcal{Q}_s$ to the problem regarding the single point set $R = O$ and the Minkowski sum $M = \mathcal{Q}_s \oplus (-\mathcal{P}_s)$. However , the difference now is that, since $\mathcal{P}_s$ and $\mathcal{Q}_s$ are not separated, the origin O is contained within sphere M, i.e. O is between m_0 and m_1 on line $\mathcal{L}$, using the same notation introduced in the proof of Theorem 2. See Figure 5 for a 2D sectional illustration on the plane determined by O, m_0, and vector S_0.

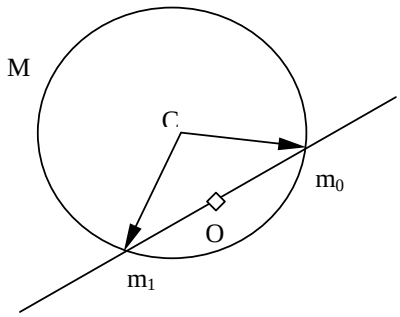

Figure 5: the proof of the theorem 4

Again, it is easy to show that $S_1 = C - m_1$ and that the supporting vertex vector of R and M with respect to S_1 is m_1. Hence, $r_0 = m_0/|m_0|$ and $r_1 = m_1/|m_1|$. Since the origin O is collinear with and between m_0 and m_1, $r_0 = -r_1$. This completes the proof.

Theorem 4 states that if, theoretically, HS-jump is applied to two intersecting spheres, then HS-jump can detect within two steps that $\mathcal{P}_s$ and $\mathcal{Q}_s$ intersect. Hence, we expect that HS-jump is also capable of detecting efficiently the collision between two intersecting polytopes if the polytopes are nearly sphere-shaped (referring to Theorem 2). This property furnishes another justification for using formula (1) for computing new candidate separating vectors in HS-jump.

6 Termination conditions

It is important to ensure that any collision detection algorithm terminates and reports the correct collision status within a finite number of steps. In this section we will discuss the termination conditions of HS-jump.

Suppose that HS-jump is applied to detecting collision of two polytopes $\mathcal{P}$ and $\mathcal{Q}$. A supporting vertex pair with a candidate separating vector $\mathbf{S}_i$ is generated in each iteration by HS-jump. One way to prevent HS-jump from falling into an infinite loop is to ensure that the number of times any supporting vertex pair occurs is bounded by a constant depending on the size of the input. With this in mind, in the following we will introduce an auxiliary rule of choosing a new candidate separating vector to guarantee that HS-jump terminates within $2 \times |\mathcal{V}(\mathcal{P})| \times (|\mathcal{V}(\mathcal{Q})|$ iterations.

A supporting vertex pair may appear in two consecutive iterations or reappear after more than one iteration. In the former case we have the following result.

Theorem 5: *Suppose that $\mathbf{S}_{i+1}$ is obtained from $\mathbf{S}_i$ using (1). Let $(\mathbf{p}_i, \mathbf{q}_i)$ and $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ be the supporting vertex pairs with respect to candidate separating vectors $\mathbf{S}_i$ and $\mathbf{S}_{i+1}$, respectively. Suppose $\mathbf{S}_i \cdot (\mathbf{q}_i - \mathbf{p}_i) < 0$ and $(\mathbf{p}_i, \mathbf{q}_i) = (\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$. Then $\mathbf{S}_{i+1} \cdot (\mathbf{q}_{i+1} - \mathbf{p}_{i+1}) > 0$, i.e., $\mathcal{P}$ and $\mathcal{Q}$ do not collide.*

Proof: Since $S_i \cdot (q_i - p_i) < 0$, by formula (1) we have

$$\begin{aligned} S_{i+1} \cdot (q_{i+1} - p_{i+1}) &= S_{i+1} \cdot (q_i - p_i) \\ &= (S_i - 2(S_i \cdot r_i)r_i) \cdot (q_i - p_i) \\ &= S_i \cdot (q_i - p_i) - 2(S_i \cdot r_i)r_i \cdot (q_i - p_i) \\ &= S_i \cdot (q_i - p_i) - 2S_i \cdot (q_i - p_i) \\ &= -S_i \cdot (q_i - p_i) > 0 \end{aligned}$$

so we have $S_{i+1} \cdot (q_{i+1} - p_{i+1}) > 0$, i.e. $\mathcal{P}$ and $\mathcal{Q}$ do not collide. This completes the proof.

Theorem 5 asserts that if the same supporting vertex pair appears in two consecutive steps then HS-jump reports that $\mathcal{P}$ and $\mathcal{Q}$ are separated, However, in the case where a supporting vertex pair reappear after more than one iteration, the following auxiliary (second) rule for choosing a new candidate separating vector will apply. Suppose that $\mathbf{S}_{i+1}$ in the $(i+1)$-th iteration is derived from (1) and the corresponding supporting vertex pair $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ has appeared in the j-th iteration with $0 \leq j < i$. Suppose $\mathbf{S}_{i+1} \cdot (\mathbf{q}_{i+1} - \mathbf{p}_{i+1}) < 0$. Then S_{i+1} will be discarded and the pair $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ will not be used to update the spherical polygon $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$. Furthermore, instead of applying again formula (1), the candidate separating vector $\mathbf{S}_{i+1}$ will be regenerated by being set to be a point $\mathbf{w}_i$ in $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$. Note that in the case $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ is not empty, for otherwise HS-jump would have reported collision between $\mathcal{P}$ and $\mathcal{Q}$. The new supporting vertex pair $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ resulting from the $\mathbf{S}_{i+1}$ with this above auxiliary rule has the favorite property that either 1) $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ has not appeared in any preceding iteration j, $j < i$; or 2) $\mathbf{S}_{i+1} \cdot (\mathbf{q}_{i+1} - \mathbf{p}_{i+1}) > 0$, i.e., HS-jump can report that $\mathcal{P}$ and $\mathcal{Q}$ are separated. To see this, assuming that $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1}) = (\mathbf{p}_j, \mathbf{q}_j)$ for some $j \leq i$, then

$$\mathbf{S}_{i+2} \cdot (\mathbf{q}_{i+2} - \mathbf{p}_{i+2}) = \mathbf{w}_i \cdot (\mathbf{q}_j - \mathbf{p}_j) > 0,$$

since $\mathbf{w}_i \in \bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$, where $\mathbf{r}_i = (\mathbf{q}_j - \mathbf{p}_j)/|\mathbf{q}_j - \mathbf{p}_j|$.

To summarize, the candidate separating vectors in HS-jump are generated with either of the two ruless as follows. Suppose that $\mathbf{S}_i \cdot (\mathbf{q}_i - \mathbf{p}_i) < 0$ in the i-th iteration. If the pair $(\mathbf{p}_i, \mathbf{q}_i)$ has not appeared in any preceding iteration, then $\mathbf{S}_{i+1}$ is computed using formula (1); otherwise, $\mathbf{S}_{i+1}$ is chosen to be any point in the spherical convex polygon $\bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$. According to the preceding discussion, with this way of choosing the candidate separating vectors, whenever a supporting vertex pair is found to have appeared, if it is the immediately preceding one, then separation is reported (referring to Theorem 5), otherwise, the supporting vertex pair to be generated in the next iteration is guaranteed to be a new one (by the auxiliary rule). Thus, in the worst case, at least one new supporting vertex pair is generated for every two consecutive iterations. Since the total number of vertex pairs is bounded by $|\mathcal{V}(\mathcal{P})| \times |\mathcal{V}(\mathcal{Q})|$, the total number of iterations HS-jump performs for each frame is at most $2 \times |\mathcal{V}(\mathcal{P})| \times |\mathcal{V}(\mathcal{Q})|$. In general, when the $|\mathcal{V}(\mathcal{P})|$ and $|\mathcal{V}(\mathcal{Q})|$ are not over 4000, the number of the iteration is not over 40. When all the vertex pairs $(\mathbf{p}, \mathbf{q})$, where $\mathbf{p} \in \mathcal{V}(\mathcal{P})$ and $\mathbf{q} \in \mathcal{V}(\mathcal{Q})$, have been exhausted by HS-jump as supporting vertex pairs (and $\mathcal{CH}_i$ is still non-empty), HS-jump will declare that $\mathcal{P}$ and $\mathcal{Q}$ are separated.

7 Complete algorithm

In the preceding sections we have explained the mechanisms to detecting separation and collision by HS-jump, as well as the means to ensure HS-jump to terminate with a correct report of collision status in a finite number of steps. Now we are in a position to present the complete algorithm.

Assumptions and notations: Let $\mathcal{P}$ and $\mathcal{Q}$ be two bounded polytopes each with a finite number of vertices. Let $\mathcal{V}(\mathcal{P})$ and $\mathcal{V}(\mathcal{Q})$ denote the sets of vertices of $\mathcal{P}$ and $\mathcal{Q}$, respectively. The following describe how HS-jump performs collision detection for one time frame with two (stationary) polytopes, $\mathcal{P}$ and $\mathcal{Q}$. When applying HS-jump to moving polytopes in a sequence of time frames, between-frame coherence can be exploited to speed up computation; this will be elaborated in the section 8.4.

HS-jump: Algorithm for Collision Detection of Polytopes

(1) Set a nonzero vector $\mathbf{S}_0$. Set $\mathcal{SCP}_0 = S^2$. Compute the supporting vertex pair $(\mathbf{p}_0, \mathbf{q}_0)$ with respect to $\mathbf{S}_0$. Set iteration index $i := 0$.

(2) If $\mathbf{S}_i \cdot (\mathbf{q}_i, \mathbf{p}_i) > 0$, report that $\mathcal{P}$ and $\mathcal{Q}$ are separated, and stop for the current frame (see Section 4); otherwise, continue to step (3).

(3) Add the hemisphere $\mathcal{HS}(\mathbf{r}_i)$ to compute the updated spherical polygon $\mathcal{SCP}_i \equiv \bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ (see Section 5). Here $\mathbf{r}_i = (\mathbf{q}_j - \mathbf{p}_j)/|\mathbf{q}_j - \mathbf{p}_j|$. If $\mathcal{SCP}_i = \emptyset$, report that $\mathcal{P}$ and $\mathcal{Q}$ collide, and stop for the current frame (see section 5); otherwise, continue to step (4).

(4) Compute $\mathbf{S}_{i+1}$ by (1) (see Section 4).

(5) Computing the supporting vertex pair $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ with respect to $\mathbf{S}_{i+1}$.

(6) If $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1})$ has not appeared in a preceding iteration, set $i := i+1$ and go to (2); otherwise go to (7)

(7) If $(\mathbf{p}_{i+1}, \mathbf{q}_{i+1}) = (\mathbf{p}_i, \mathbf{q}_i)$, report separation (Theorem 5); otherwise, set $\mathbf{S}_{i+1}$ to be a point in the spherical convex polygon $\mathcal{SCP}_i \equiv \bigcap_{j=0}^{i} \mathcal{HS}(\mathbf{r}_j)$ (the auxiliary rule in Section 6). Go to step (5).

8 Efficient implementation

In this section we will discuss the efficient implementation of HS-jump. The emphasis is put on the following key tasks:

(1) Given a candidate separating vector $\mathbf{S}_i$, compute the supporting vertex pair $(\mathbf{p}_i, \mathbf{q}_i)$ (step 1).

(2) Checking if the supporting vertex pair $(\mathbf{p}_i, \mathbf{q}_i)$ has appeared in a preceding iteration (step 3).

(3) Given a new added vector $\mathbf{r}$, compute the updated spherical polygon $\mathcal{SCP}_{i+1} \equiv \bigcap_{j=0}^{i+1} \mathcal{HS}(\mathbf{r}_j)$ from $\mathcal{SCP}_i$.

(4) Use of between-frame coherence for speedup.

8.1 Searching for supporting vertex pairs

We use the hierarchical representation of polytope to compute the supporting vertex pair. A hierarchical representation [7] of polytope $\mathcal{P}$ with vertex set $\mathcal{V}(\mathcal{P})$ is defined as a sequence of polytopes $hier(\mathcal{P}) = \{\mathcal{P}_0, \ldots, \mathcal{P}_h\}$, $h = O(log(|\mathcal{V}(\mathcal{P})|)$, such that

1. $\mathcal{P}_0 = \mathcal{P}$ and $\mathcal{P}_h$ is a tetrahedron;
2. $\mathcal{P}_{i+1} \subset \mathcal{P}_i$, for $0 \leq i \leq h$; and
3. $\mathcal{V}(\mathcal{P}_{i+1}) \subset \mathcal{V}(\mathcal{P}_i)$, for $\leq i \leq h$; and
   the vertices in $\mathcal{V}(\mathcal{P}_i) - \mathcal{V}(\mathcal{P}_{i+1})$ form an independent set in $\mathcal{P}_i$ for $0 \leq i \leq h$.

Figure 6 shows the hierarchical representation of a polytope. The space for storing all hierarchical polytope is $O(|\mathcal{V}(\mathcal{P})|)$ and the time of preprocess of creating the hierachical polytope is $O(|\mathcal{V}(\mathcal{P})|)$.

Let p_i be the supporting vertex of the polytope $\mathcal{P}_i$. We first find the supporting vertex p_h of $\mathcal{P}_h$ by comparing its three or four vertices. Then we find the supporting vertex p_{h-1} of $\mathcal{P}_{h-1}$ by comparing

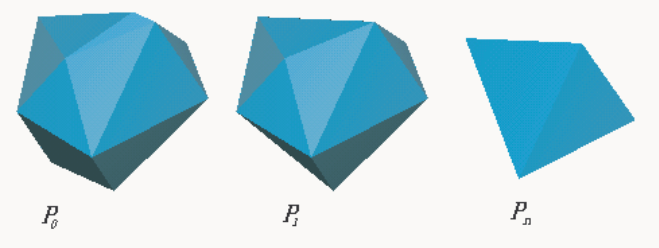

Figure 6: A Hierarchical representation of polygon

the adjacent vertices of p_h in $\mathcal{P}_{h-1}$ (note that p_h is also a vertex of $\mathcal{P}_{h-1}$). The correctness of this search is proved in [17] Continuing this search through the hierarchy, at last we find the supporting vertex p_0 of $\mathcal{P}_0$, i.e., a supporting vertex p of $\mathcal{P}$. Similar to this, we may find the supporting vertex q of $\mathcal{Q}$, using $O(\log(\mathcal{V}(\mathcal{P}) + \log(\mathcal{V}(\mathcal{Q}))$ time.

8.2 Checking the re-appearance of a supporting vertex pair

We assign to each supporting vertex pair (p_i, q_j) a unique index $n = i \times N + j$, here N is the number of vertices of the polytope $\mathcal{P}$. The indices of all supporting vertex pairs produced are stored in a balanced binary tree using each index as a key value. So we can query whether a supporting vertex pair has appeared before by search the binary tree using $O(\log k)$ time, where k is the number of the supporting vertex pairs that have been produced.

8.3 Updated spherical polygon $\mathcal{SCP}_i$

We use a balanced binary tree to store the vertices of the spherical convex polygon $\mathcal{SCP}_i$. Clearly, a primitive operation (i.e., querying, modifying, insertion and deletion of a node) in a balanced tree takes $O(\log k)$ time, here k is the number of the vertices of $\mathcal{SCP}_i$; thus $k \leq i$. For each newly generated candidate separating vector $\mathbf{r}_{i+1}$, if the open-hemisphere $\mathcal{HS}(\mathbf{r}_{i+1})$ does not intersect $\mathcal{SCP}_i$, we would be done by declaring that the polytopes collide since $\mathcal{SCP}_{i+1}$ would be empty; otherwise, the two intersection points of $\mathcal{SCP}_i$ and the boundary of $\mathcal{HS}(\mathbf{r}_{i+1})$, which is a great circle, can be found in $O(\log k)$ time, and updating $\mathcal{SCP}_i$ into $\mathcal{SCP}_{i+1}$ therefore also takes $O(\log k)$ time. Note that the rearrangement of the balanced tree might take more than $O(\log k)$ time in a particular step, but the average time (or amortized time) spent on this task in each iteration is also $O(\log k)$.

8.4 Use of between-frame coherence for speedup

At the current frame, if two polytopes are disjoint, we have a separating vector S and the associated supporting vertex pair (p, q). At the next frame, we may use S as the initial candidate separating vector when we search for a separating vector of the polytope at the new positions. To get the suppoorting vertex pair, we can start from p and q separately, and get the supporting vertex of $\mathcal{P}$ and $\mathcal{Q}$ by local and hierarchical searching. We first search supporting vertex by local search on $\mathcal{P}_0(\mathcal{P})$, If we can not get the supporting vertex, we will enter the local searching on the $\mathcal{P}_1$ by the corresponding vertex, if we still cannot get the supporting vertex on $\mathcal{P}_1$, we will enter $\mathcal{P}_2$,etc. At last we may enter $\mathcal{P}_n$, here we can get the supporting vertex quickly because there are very few vertices to be searched. Once we get the supporting vertex on $\mathcal{P}_k$, where $k \leq n$, we can get the supporting vertex by our hierarchical searching. Similarly, we can also get the supporting vertex of $\mathcal{Q}$, and form the new supporting vertex pair. Since the position and orientation of moving objects under consideration change little between adjacent time frame, we will get the new supporting vertex pair by local search on few hierachical polytopes. So using *between-frame coherence*, we can speed up the implement of our algrithm. If the polytopes $\mathcal{P}$ and $\mathcal{Q}$ collide at the current time frame, since we have recorded all S_i and (p_i, q_i) processed, using these information, we can get new supporting vertex pair (p_i, q_i) with respect to S_i quickly. With the same reasoning, the spherical cap determined by these (p_i, q_i) is almost empty, and HS-jump reports that $\mathcal{P}$ and $\mathcal{Q}$ collide.

9 Experiments

In this section we present some experimetal results of using HS-jump to detect collision of two ellipsoids. The programe was implemented in C++ and ran on a PC with Pentium II 233MHz CPU. We compared our algorithm with two commonly used collision detection algorithm: GJK [4] and I-COLLIDE [6]. The code of these aogrithms are adapted from the UNIX versions available from their authors' websites.

Performance evaluation and comparison of collision detection algorithms are not complicated because the performance depends on the shape, size, relative distance and orientation of an object, as well as whether an object is moving in which case different ways of exploiting between-frame coherence can make a difference. In addition, the efficiency of a particular algorithm depends on the level of code optimization. For all these reasons there is so far no standard benchmark for evaluating a collision detection algorithm. Hence, in our experiments, we assume that the shape of a convex is approximated by an ellipsoids, i.e., the polyhedron is obtained as a convex hull of some sampled points of an ellipsoid. However, we allow the input objects to have different relative positions and orientations.

We used two objects of the same ellipsoidal shape, same size, and same number of vertices. The distance d between the two objects ranges from -40 to 40, with increment of 0.2, where d is

the distance that the one object must be moved in one direction so that the two objects become disjont. For any fixed distance, one object is fixed and the other assumes 500 randomly generated orietnations; thus 500 pairs of objects are generated for each fixed distance. We then ran a collision detection algorithm 500 times for each pair of objects with different orietnations to get the average time of the collision detection methods for a fixed distance.

Figure 7 shows the comparison of three methods: HS-jump, GJK and I-COLLIDE. In this case, the number of vertices of an object is 200, and the three principle axes of the ellipsoid from which the object is derived are $a = 200$, $b = 180$, and $c = 180$. The curves in Figure 7 show the time of the three algorithms with respect to distance. The black curve is for HS-jump, the tint one for GJK, and the top one for I-COLLIDE.

Figure 8 and 9 show the comparison of three methods with a varing number of vertices and the same shape. The number of vertices of an object in Figure 8 is 1000, and the number of vertices of an object in Figure 9 is 2000, and the three principle axes of the ellipsoid from which the objects in Figures 8 and 9 are derived are $a = 200$, $b = 180$, and $c = 180$.

Figure 10 and 11 show the comparison of three methods with the same number of the vertices and different shapes. The number of vertices of two objects are 2000. In Figure 10, the three principle axes of the ellipsoid from which the object is derived are $a = 200$, $b = 100$, and $c = 100$. In Figure 11 the three principle axes of the ellipsoid from which the object is derived are $a = 200$, $b = 20$, and $c = 20$.

From Figure 7 to Figure 11, we can see that I-COLLIDE is slower than the other two in both cases of disjoint and collision. Moreover, I-COLLIDE runs much more slowly in the case of collision than the case of non-collision. The reason is probably that I-COLLIDE needs to solve a linear programming problem when two polytopes collide, thus taking longer time. We see that 1) HS-jump is faster than GJK when two polytopes are disjoint; 2) the times of two algorithm are almost same when the two polytopes have little intersection; 3)GJK gets faster when the objects intersect more deeply. On the other hand, GJK and I-COLLIDE can get the shortest distacne of two disjoint objects, but HS-jump only reports that two objects are disjoint. However, we can aslo compute the shortest distance quickly with the help of the information obtained by HS-jump. Although we have not experience any problem with the proper termination of GJK, we note that the termination condition of the GJK method has not been established, while we have proven the termination condition of HS-jump.

We see from figure 7 to figure 9 that the collision detection time of GJK increase quickly when the number of vertices of an object increases, but the time of HS-jump increse relatively slowly. From figure 8 to figure 11, There is a limitation revealed in figure 8 through figure 11, that is, HS-jump takes longer time to run when the objects get thinner and longer; this is explained by that the heuristic search strategy is based on the opitimal case of two spherical objects. But the shape of an object has little effect on the GJK method.

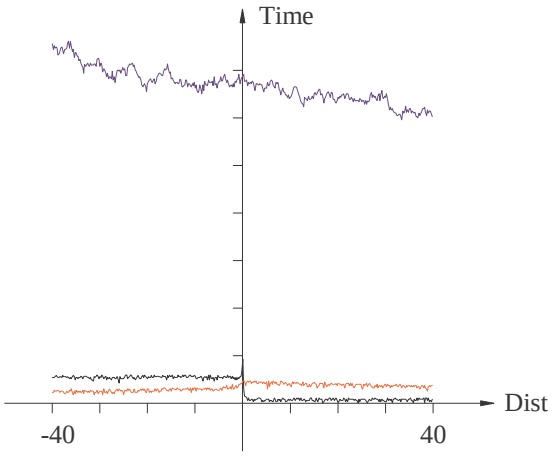

Figure 7: $N = 200, a : b = 10 : 9$

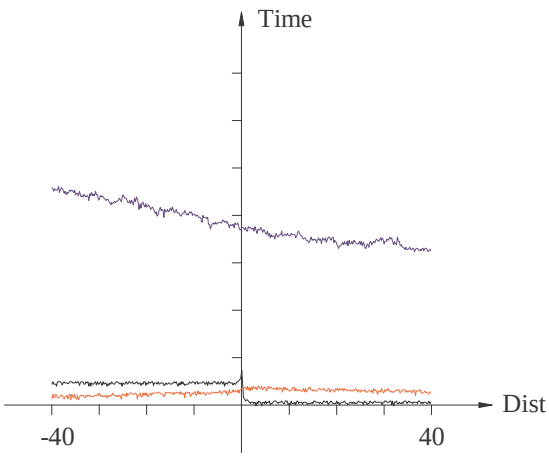

Figure 8: $N = 1000, a : b = 10 : 9$

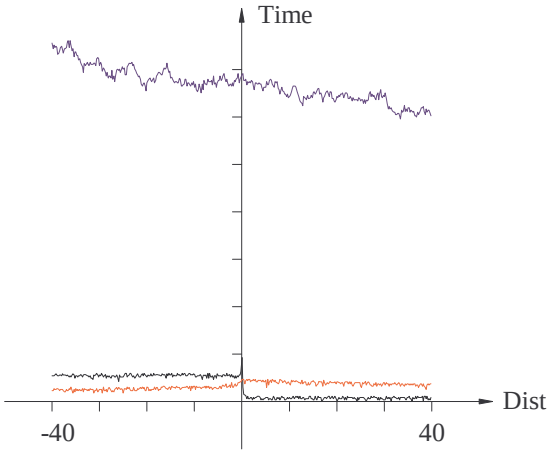

Figure 9: $N = 2000, a : b = 10 : 9$

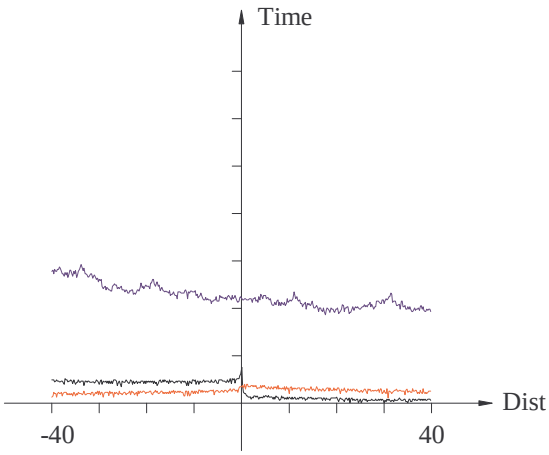

Figure 10: $N = 2000, a : b = 10 : 5$

10 Conclusion

We have presented an efficient exact collision detection algorithm polytopes (convex polyhedra). The algorithm is based on a simple heuristic technique to quickly locate a separating plane between polytopes if they do not collide, or otherwise, to report collision quickly by updating a spherial convex polygon. Experiments show that our algorithm is fast and simple to implement, especially when the two polytopes are disjoint or nearly spherical. The algorithm gains further speedup by

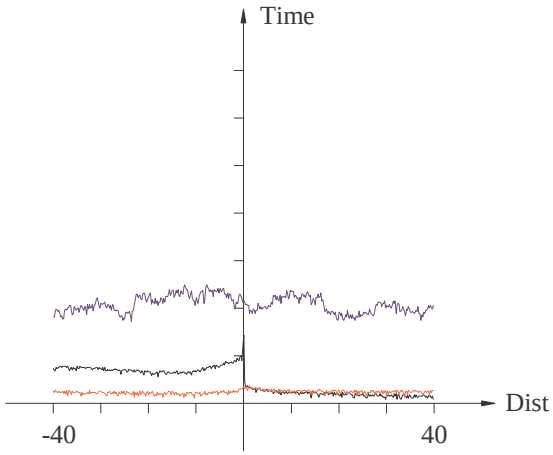

Figure 11: $N = 2000, a : b = 10 : 1$

taking advantage of geometric and temporal coherence in a dynamic environment. Furthermore, unlike most other existing collision detection methods for polytopes, the termination condition of HS-jump has been rigorously established. One drawback of HS-jump is that it takes longer time to report collision for objects that are thin and long. We expect to expend HS-jump to deal with convex bodies bounded by curved surfaces.

References

[1] G. Barequet, B. Chazelle, L. Guibas, J. Mitchell, and A. Tal, Boxtree: A hierarchical representation of surfaces, in *Proceeding of Eurographics'96*, Vol. 15, pp.387-484, 1996.

[2] Julien Basch, Jeff Erickson, Leonidas J. Guibas, John Hershberger, and Li Zhang, Kinetic Collision Detection Between Two Simple Polygons, in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[3] S. Cameron, Collision Detection by Four-Dimensional Intersection Testing, *IEEE Transactions on Robotics and Automation*, 6(3):291-302,1990.

[4] Stephen Cameron, A Comparison of Two Fast Algorithms for Computing the Distance Between Convex Polyhedra, in *IEEE Transactions on Robotics and Automation*, 13(6):915-920, 1997.

[5] Kelvin Chung and Wenping Wang, Quick Collision Detection of Polytopes in Virtual Environments, in VRSR'96 , p125-131,1996

[6] J. D. Cohen, M. C. Lin, D. Manocha, and M. K. Ponamgi, I-COLLIDE: An Interactive Detection System for Large-Scale Environments, In *Pro. ACM Interactive 3D Graphics Conf*, pp. 189-196, 1995.

[7] D.P.Dobkin and D.G.Kirkpatrick, A linear algorithm for determining the separation of vonvex polyhedra, *Journal of Algorithms*Vol. 6, pp. 381-392, 1985

[8] D. P. Dobkin and D. G. Kirkpatrick, Determining the Separation of Preprocessed Polyhedra-A Unified Approach, In *Proceedings 17th Internat. Colloq. Automata Lang. Program*, LNCS 433, pp. 400-413, 1990.

[9] Jeff Erickson, Leonidas J. Guibas, Jorge Stolfi, and Li Zhang, Separation-Sensitive Collision Detection for Convex Objects, in *Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[10] A. Garcia-Alonso, N. Serrano, and J. Flaquer, Solving the Collision Detection Problem, *IEEE Comput. Graph.*Appl., 14:36-43, May 1994.

[11] George Baciu and S-K W. Wong, Rendering in Object Interference Detection on Conventional Graphics Workstations, in *Proceedings of the Pacific Graphics*, pp.51-58, 1997.

[12] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A Fast Procedure for Computing the Distance Between Objects in Three-Dimensions Space, *IEEE Trans., Robotics and Automation.*Vol RA-4: 193-203, 1988.

[13] S. Gottschalk, M. C. Lin, and D. Manocha, OBBTree: A Hierarchical Structure for Rapid Interference Detection, *Comput. Graphics (SIGGRAPH '96 Proc.)*, pp. 171-180, Aug 1996.

[14] M. Held, J. T. Klosowski, and J. S. B. Mitchell, Evaluation of Collision Detection Methods for Virtual Reality Fly-Throughs, *In Proc. 7th Canad. Conf. Comput. Geom.*, pp. 205-210, 1995.

[15] P. M. Hubbard, Collision Detection for Interactive Graphics Applications, *IEEE Trans. Visual. comput. Graph.*1(3):218-230, Sept 1995.

[16] P. M. Hubbard, Appoximating Polyhedra with Spheres for Time Critical Collision Detection, *ACM Trans. Graph.*,15(3):179-210, July 1996.

[17] Joseph o'Rourke,*Computational Geometry in C*, Cambriged University Press 1993, pp. 253-265.

[18] J. Klosowski, M. Held, J. S. B. Mitchell, H. Sowizral, and K. Zikan, Efficient Collision Detection Using Bounding Volume Hierarchies of k-dops, In *Siggraph'96 Virtual Proceedings*, pp.151, 1996.

[19] M. Lin and D. Manocha, Fast Interference Detection Between Geometric Models, *Visual Comput.*, 11(10):542-561, 1995.

[20] N. Megiddo, Linear-time Algorithms for Linear Programming in R^3 and Related Problems, *SIAM J. Computing*, 12:759-766, 1983.

[21] Brian Mirtich, Rigid Body Contact: Collision Detection to Force Computation, in *IEEE International Conference on Robotics and Automation*, May, 1998.

[22] M. Moore and J. Wilhelms, Collision Detection and Response for Computer Animation, In *Comput., Graphics (SIGGRAPH88 Proc.)*, Vol. 22, pp. 289-298, Aug 1988.

[23] K. Myszkowski, O. G. Okunev, and T. L. Kunuii, Fast Collision Detection between Complex Solids Using Rasterizing Graphics Hardware, *The Visual Computer*, 11:497-511, 1995.

[24] B. Naylor, J. A. Amatodes, and W. Thibault, Merging BSP Tree Yields Polyhedral Set Operations, In *Comput. Graphics (SIGGRAPH '90 Proc.)*, Vol. 24, pp.115-124, Aug 1990.

[25] I. Palmer and R. Grimsdale, Collision Detection for Animation Using Sphere-Trees, *Compute. Graph. Forum*, 14(2): 105-116, June 1995.

[26] M. Ponamgi, D. Manocha, and M. Lin, Incremental Algorithms for Collision Detection Between General Solid Models, in *Proc. ACM Siggraph Sympos. Solid Modeling*, pp.293-304, 1995.

[27] R. Seidel, Linear Programming and Convex Hulls Made Easy, In *proceeding 6th Ann. ACM Conference On Computational Geometry*, pp. 211-215, Berkeley, California, 1990.

[28] Colin Turnbull, Stephen Cameron, Computing Distances between NURBS-defined Convex Objects, in *ICRA'98, Leuven*, May 1998.

[29] G. Vanecek, Jr., Brep-index: A Multidimensional Space Partitioning Tree, *Internat. J. Comput. Geom.*Appl., 1(3):243-261, 1991.