基于启发式搜索分离向量的凸多面体碰撞检测

李学庆[1)] 孟祥旭[1)] 汪嘉业[1)] 王文平[2)] CHUNG Kelvin[2)] YIU Siu Ming[2)]

[1)](山东大学计算机科学与技术学院 济南 250100)

[2)](香港大学计算机科学与资讯系统系 香港)

摘 要 碰撞检测是计算机模拟物理过程的基础,在计算机图形学、CAD/CAM、虚拟现实和机器人等领域有着广泛的应用. 该文给出了一个新的用于凸多面体碰撞检测的算法——HP-jump. HP-jump 建立了一个有效的碰撞检测模型用于报告物体的碰撞, 同时提供了一个快速的启发式的策略用于搜索两个凸多面体的分离向量. 该算法是利用凸多面体的层次表示来搜索支撑顶点对,用平衡二叉树来记录球面凸多边形的顶点,同时还利用了时间、空间相关性,这些都加速了算法的执行. 该文的最后给出了 HP-jump 与 GJK,I_COLLIDE 算法的比较.

关键词 计算几何,计算机图形学,虚拟现实,碰撞检测

中图法分类号: TP18

Detecting Collision of Polytopes Using a Heuristic Search for Separating Vectors

LI Xue-Qing[1)] MENG Xiang-Xu[1)] WANG J Y[1)]

WANG Wen-Ping[2)] CHUNG Kelvin[2)] YIU Siu Ming[2)]

[1)](*College of Computer Science and Technology, Shandong University, Jinan* 250100)

[2)](*Department of Compute Science and Information Systems, The University of Hong Kong, Hong Kong*)

Abstract The collision detection problem is to determine whether two moving objects collide or not at any moment. It is fundamental to computer simulation of a physical environment, and has applications in computer graphics, CAD/CAM, virtual reality, and robotics. This paper proposes a new method, called 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. Suppose an ordered pair $(\boldsymbol{p},\boldsymbol{q})$ is a supporting vertex pair of two polytopes P and Q with respect to a vector $\boldsymbol{S}$, if it satisfies the separating condition $\boldsymbol{S}\cdot(\boldsymbol{q}-\boldsymbol{p})>0$, then P and Q do not collide and $\boldsymbol{S}$ is called a separating vector. The algorithm firstly sets a nonzero vector as the candidate separating vector $\boldsymbol{S}$ and computes the supporting vertex pair $(\boldsymbol{p},\boldsymbol{q})$. If the separating condition is satisfied, then P and Q are disjoint and the algorithm terminates. Otherwise the new candidate separating vector $\boldsymbol{S}$ is obtained using the heuristic formula $\boldsymbol{S}_{i+1}=\boldsymbol{S}_i-2(\boldsymbol{S}_i\cdot\boldsymbol{r}_i)\boldsymbol{r}_i$, where $\boldsymbol{r}_i=(\boldsymbol{q}_i-\boldsymbol{p}_i)/|(\boldsymbol{q}_i-\boldsymbol{p}_i)|$. This step is repeated until the algorithm finds the vector $\boldsymbol{S}$ which satisfies the separating condition or reports that P and Q collide by noting that the spherical convex polygon formed from the supporting vertex pairs is empty. To speed up the implementation of the algorithm, the hierarchical representation of a polyto-

收稿日期:2001-11-06;修改稿收到日期:2002-02-07. 本课题得到国家自然科学基金(69873028)和高等学校优秀青年教学科研奖励计划资助.**李学庆**,男,1964 年生,博士,副教授,主要研究领域为计算几何、计算机图形学和人机交互. E-mail: liyou@jn-public.sd.cninfo.net. **孟祥旭**,男,1962 年生,博士,教授,博士生导师,主要研究领域为计算机图形学、虚拟现实. **汪嘉业**,男,1937 年生,教授,博士生导师,主要研究领域为计算几何、计算机图形学. **王文平**,男,1963 年生,博士,副教授,主要研究领域为计算几何、计算机图形学. **CHUNG Kelvin**, 男,1970 年生, 硕士,主要研究领域为碰撞检测. **YIU Siu Ming**,男,1968 年生,博士,主要研究领域为算法.

pe is used to compute the supporting vertex pair, and a balanced binary tree is used to store the vertices of the spherical convex polygon, and between-frame coherence is exploited . Due to the particular nature of the search scheme used, HS-jump becomes more efficient when the convex polyhedra are more spherically shaped. Hence, in the case of applying HS-jump to convex polyhedra with a large number of vertices, HS-jump delivers the maximum efficiency if the objects are on average not very elongated.

Keywords computational geometry; computer graphics; virtual reality; collision detection

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, and 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 performance for both cases of separated and colliding polytopes. A pair of closest features 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 of Related Works

There are many techniques proposed in the literatures to solve the problem of collision detection: spatial decomposition[1], bounding volume[2], the space-time bounds[3], four-dimensional geometry[4], and image-based algorithms[5,6].

Using the techniques of computational geometry to study collision detection is also an important approach. One method is to utilize Voronoi diagrams[7,8] to keep track of the closest features between pairs of objects. The popular system, I-COLLIDE[7], 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. 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 Dobkin's paper[9] for the polytopes collision detection problem, where n is the number of vertices. Good theoretical and practical approaches based on the linear programming problem are also known[10,11]. Minkowski difference and convex optimization techniques are used in paper [12] to compute the distance between convex protopes by finding the closest points. In Gilbort's paper[13], Stephen Ca-

meron modified the algorithm of Gilbert, and showed that his algorithm has $O(1)$ time cost under the reasonable assumptions.

Most of these methods are efficient in the case of non-collision, but not very efficient in the case of collision. Furthermore the termination conditions of these methods are not established rigorously. In paper [14], a method, called Q-Collide, based on separating vectors is proposed for collision detection of polytopes. In this paper we improve the original method in paper [14] 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 developments.

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 P be a bounded polytope in E^3. $V(P)$ denote the set of vertices of P, and $(\boldsymbol{x}\cdot\boldsymbol{y})$ denote the inner product of 3D vectors $\boldsymbol{x}$ and $\boldsymbol{y}$. Given a vector $\boldsymbol{S}\neq 0$, a vertex $\boldsymbol{p}\in V(P)$ is called a supporting vertex of P with respect to $\boldsymbol{S}$ if $\boldsymbol{S}\cdot\boldsymbol{p}=\max\{\boldsymbol{S}\cdot\boldsymbol{x}\mid\boldsymbol{x}\in V(P)\}$; Such a supporting vertex $\boldsymbol{p}$ may not be unique for a given polytope P and vector $\boldsymbol{S}$.

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

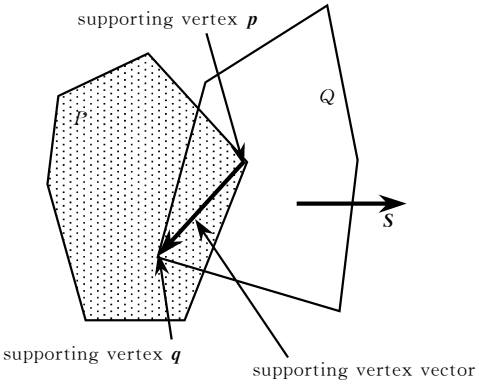


Fig. 1　The supporting vertex pair

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

The polytopes P and Q are said to be disjoint, or separated, or non-colliding if $P\bigcap Q=\varnothing$. For two disjoint polytopes P and Q, there exists a plane L such that P and Q are on the opposite sides of L and $P\bigcap L=\varnothing$ and $Q\bigcap L=\varnothing$. See Figure 2.

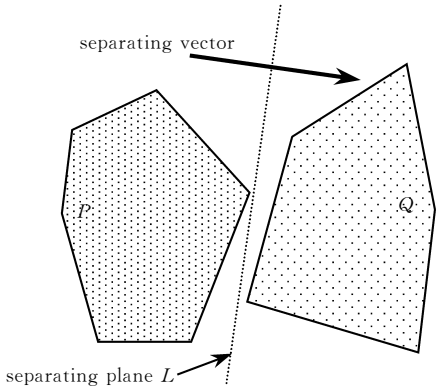


Fig. 2　P and Q are disjoint

The plane L is called a separating plane of P and Q. A separating vector of P and Q is defined to be a normal vector $\boldsymbol{S}$ of L that points from the side of P to the side of Q, i. e., $\boldsymbol{S}\cdot(\boldsymbol{y}-\boldsymbol{x})>0$ for any $\boldsymbol{x}\in P$ and any $\boldsymbol{y}\in Q$.

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

Proof.　Firstly, we suppose that $\boldsymbol{S}$ is a separating vector of polytopes P and Q. By the definition of $\boldsymbol{S}$, for any $\boldsymbol{x}\in P$ and $\boldsymbol{y}\in Q$, we have $\boldsymbol{S}\cdot(\boldsymbol{y}-\boldsymbol{x})>0$. Since $\boldsymbol{p}\in P$ and $\boldsymbol{q}\in Q$, we have $\boldsymbol{S}\cdot(\boldsymbol{q}-\boldsymbol{p})>0$.

Now we have $\boldsymbol{S}\cdot\boldsymbol{q}>\boldsymbol{S}\cdot\boldsymbol{p}$, since $\boldsymbol{p}$ and $\boldsymbol{q}$ are the supporting vertices of P and Q with respect to $\boldsymbol{S}$, by the definition of a supporting vertex, we have $\boldsymbol{S}\cdot\boldsymbol{p}\geq\boldsymbol{S}\cdot\boldsymbol{x}$, $\forall\,\boldsymbol{x}\in P$, $-\boldsymbol{S}\cdot\boldsymbol{q}\geq-\boldsymbol{S}\cdot\boldsymbol{y}$, $\forall\,\boldsymbol{y}\in Q$, then $\boldsymbol{S}\cdot\boldsymbol{y}\geq\boldsymbol{S}\cdot\boldsymbol{q}>\boldsymbol{S}\cdot\boldsymbol{p}>\boldsymbol{S}\cdot\boldsymbol{x}$, at last we have $\boldsymbol{S}\cdot(\boldsymbol{y}-\boldsymbol{x})>0$, $\forall\,\boldsymbol{x}\in P$ and $\forall\,\boldsymbol{y}\in Q$, i. e., $\boldsymbol{S}$ is a separating vector of the polytopes P and 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 $\boldsymbol{S}\cdot(\boldsymbol{q}-\boldsymbol{p})>0$. HS-jump is based on this idea and performs iterations to resolve the collision detection problem between two polytopes P and Q in a given frame. Within a finite number of steps, HS-jump can either find a separating vector, so declaring P and Q to be separated, or find P and 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 $\boldsymbol{S}_i$, and its corresponding pair of supporting vertices $\boldsymbol{p}_i \in V(P)$ and $\boldsymbol{q}_i \in V(Q)$ (The rules of generating candidate separating vectors will be introduced in Section 4 and Section 6). If $\boldsymbol{S}_i$ is a separating vector, i.e., $\boldsymbol{S}_i \cdot (\boldsymbol{q}_i - \boldsymbol{p}_i) > 0$, declare that P and Q do not collide in the present frame; otherwise, go to step (2).

(2) Use all the pairs of vertices $(\boldsymbol{p}_j, \boldsymbol{q}_j)$, $j = 0, 1, \cdots, i$, that have been produced so far, to check if P and 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.

4 Detecting separation

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

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 $\boldsymbol{S}_i$ has been found not to be a separating vector, then the first rule of generating the new candidate separating vector $\boldsymbol{S}_{i+1}$ in the next frame is by the formula

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

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 P_S and Q_S be two disjoint spheres in E^3. Let $\boldsymbol{S}_0$ be a nonzero vector which is not a separating vector of P_S and Q_S. The vector $\boldsymbol{S}_1$ derived from $\boldsymbol{S}_0$ through formula (1) is a separating vector of P_S and Q_S.

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

Since P_S and Q_S are separated, R and M are also separated. Therefore, the origin O is not contained in the sphere M. Let $\boldsymbol{m}_0$ be the supporting vertex vector of R and M with respect to $\boldsymbol{S}_0$. Then

$\boldsymbol{m}_0$ can be regarded as a point on the boundary of M. Let L be the line passing through the origin O and $\boldsymbol{m}_0$. Let $\boldsymbol{m}_0$ and $\boldsymbol{m}_1$ be the two intersections of L and M. Since $\boldsymbol{S}_0$ is not a separating vector of R and M, $\boldsymbol{m}_1$ is between $\boldsymbol{m}_0$ and O on line L. Figure 3 shows the sectional view of sphere M on the plane determined by the origin O, $\boldsymbol{m}_0$, and vector $\boldsymbol{S}_0$.

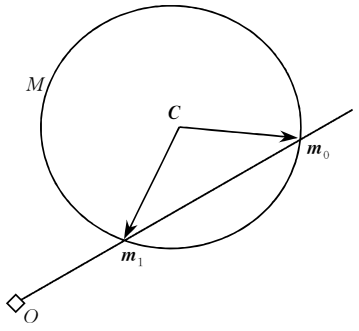


Fig. 3　The proof of the theorem 2

Let $\boldsymbol{C}$ denote the center of the sphere M. Without loss of generality, we may assume $\boldsymbol{S}_0=\boldsymbol{C}-\boldsymbol{m}_0$, since a common scaling factor to $\boldsymbol{S}_i$ and $\boldsymbol{S}_{i+1}$ in formula (1) is immaterial. Now the unit supporting vertex vector with respect to $\boldsymbol{S}_0$ is $\boldsymbol{r}_0=\boldsymbol{m}_0/|\boldsymbol{m}_0|$. Since $(\boldsymbol{m}_0+\boldsymbol{m}_1)/2=(\boldsymbol{C}\cdot\boldsymbol{r}_0)\boldsymbol{r}_0$, it follows that $\boldsymbol{m}_1=2(\boldsymbol{C}\cdot\boldsymbol{r}_0)\boldsymbol{r}_0-\boldsymbol{m}_0$. Thus, according to formula (1),

$$\begin{aligned}\boldsymbol{S}_1&=\boldsymbol{S}_0-2(\boldsymbol{S}_0\cdot\boldsymbol{r}_0)\boldsymbol{r}_0\\&=\boldsymbol{C}-\boldsymbol{m}_0-2[(\boldsymbol{C}-\boldsymbol{m}_0)\cdot\boldsymbol{r}_0]\boldsymbol{r}_0\\&=\boldsymbol{C}-[2(\boldsymbol{C}\cdot\boldsymbol{r}_0)\boldsymbol{r}_0-\boldsymbol{m}_0]\\&=\boldsymbol{C}-\boldsymbol{m}_1.\end{aligned}$$

Therefore, the supporting vertex vector with respect to the next candidate separating vector $\boldsymbol{S}_1=\boldsymbol{C}-\boldsymbol{m}_1$ is the normal vector of the tangent plane of M at $\boldsymbol{m}_1$, and this tangent plane is clearly a separating plane of R and M. Hence $\boldsymbol{S}_1$ is a separating vector of R and M, and, according to the argument at the beginning of the proof, $\boldsymbol{S}_1$ is also a separating vector of $P_{\boldsymbol{S}}$ and $Q_{\boldsymbol{S}}$. 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 $\boldsymbol{S}_i$ produced by formula (1) for two separated polytopes.

Theorem 3.　Let $P_{\boldsymbol{S}}$ and $Q_{\boldsymbol{S}}$ be two separated polytopes. Let $\hat{\boldsymbol{S}}$ be an arbitrary separating vector of $P_{\boldsymbol{S}}$ and $Q_{\boldsymbol{S}}$, then the angle between $\hat{\boldsymbol{S}}$ and the $\boldsymbol{S}_i$ produced by formula (1) decreases monotonically as i increase(see paper[14]).

Although Theorem 3 indicates that the vector $\boldsymbol{S}_i$ generated by formula (1) gets closer and closer to an arbitrary but fixed separating vector of two separated polytopes, there is no guarantee that the $\boldsymbol{S}_i$ produced by formula (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 $\boldsymbol{S}_{i+1}$ from $\boldsymbol{S}_i$.

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

Now the question is how to test efficiently if $\boldsymbol{0}\in CH_i$. We will show that it takes at most $O(\log i)$ time to test if $\boldsymbol{0}\in CH_i$ by checking if there exists an open hemisphere of $\boldsymbol{S}^2$ that contains the set of points $\boldsymbol{r}_j,j=0,1,2\cdots,i$, on $\boldsymbol{S}^2$, the unit sphere centered at the origin; an open hemisphere of $\boldsymbol{S}^2$ is

the intersection of S^2 and the open half space defined by a plane passing through the origin.

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

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 paper [15], and using a balanced binary tree, $\bigcap_{j=0}^{i+1}HS(r_j)$ can be computed from $\bigcap_{j=0}^{i}HS(r_j)$ in $O(\log i)$ time. Let $L(r)$ denote the plane with the normal vector r and passing through the origin, i.e., $\{x\mid x\cdot r=0\}$. When a new hemisphere $HS(r_{j+1})$ is added, the basic operation to form $\bigcap_{j=0}^{i+1}HS(r_j)$ is to find the intersection points of the plane $L(r_{i+1})$ with the boundary of $\bigcap_{j=0}^{i}HS(r_j)$. There are, in general, either none or two such intersection points. When $L(r_{i+1})$ and $\bigcap_{j=0}^{i}HS(r_j)$ intersect in two points, the sides of $\bigcap_{j=0}^{i}HS(r_j)$ need to be updated to obtain $\bigcap_{j=0}^{i+1}HS(r_j)$; when they do not have any intersection points, either $\bigcap_{j=0}^{i+1}HS(r_j)=\bigcap_{j=0}^{i}HS(r_j)$ or $\bigcap_{j=0}^{i+1}HS(r_j)=0$, with the latter case indicating that the two polytopes P and Q intersect. All these cases are illustrated in Figure 4. We end this section with one more property of the candidate separating vector S_i produced by formula (1).

Theorem 4. Let P_s and Q_s be two intersecting spheres in E^3, including the case of one sphere containing the other. Let S_0 be an arbitrary initial candidate separating vector. Let S_1 be the next candidate separating vector generated from S_0 using formula (1). Let r_0 and r_1 be the unit supporting vectors with respect to S_0 and S_1, respectively, then

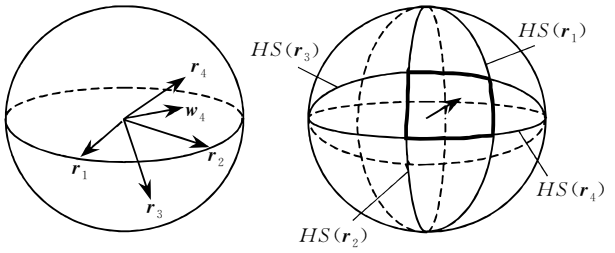


Fig. 4 Computing the spherical convex polygon

$r_0=-r_1$, and consequently, $HS(r_0)\bigcap HS(r_1)=\varnothing$.

Proof. We will use the similar argument to that used in the proof of Theorem 2, i.e., translating the problem regarding P_S and Q_S to the problem regarding the single point set $R=O$ and the Minkowski sum $M=Q_S\oplus(-P_S)$. However, the difference now is that, since P_S and 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 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.

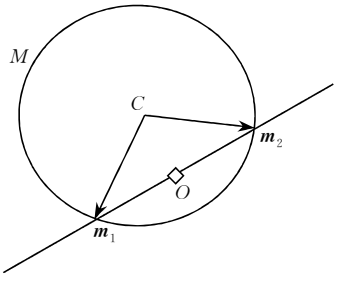


Fig. 5 The proof of the Theorem 4

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 P_S and 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 de-

tection 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 P and Q. A supporting vertex pair with a candidate separating vector $\boldsymbol{S}_i$ 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 $2\times|V(P)|\times|V(Q)|$ 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 $\boldsymbol{S}_{i+1}$ is obtained from $\boldsymbol{S}_i$ using formula (1). Let $(\boldsymbol{p}_i,\boldsymbol{q}_i)$ and $(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})$ be the supporting vertex pairs with respect to candidate separating vectors $\boldsymbol{S}_i$ and $\boldsymbol{S}_{i+1}$, respectively. Suppose $\boldsymbol{S}_i\cdot(\boldsymbol{q}_i-\boldsymbol{p}_i)<0$ and $(\boldsymbol{p}_i,\boldsymbol{q}_i)=(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})$. Then $\boldsymbol{S}_{i+1}\cdot(\boldsymbol{q}_{i+1}-\boldsymbol{p}_{i+1})>0$, i.e., P and Q do not collide, see paper [14]. Theorem 5 asserts that if the same supporting vertex pair appears in two consecutive steps, HS-jump will report that P and 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 be applied. Suppose that $\boldsymbol{S}_{i+1}$ in the $i+1$-th iteration is derived from formula (1) and the corresponding supporting vertex pair $(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})$ has appeared in the j-th iteration with $0\leq j\leq i$. Suppose $\boldsymbol{S}_{i+1}\cdot(\boldsymbol{q}_{i+1}-\boldsymbol{p}_{i+1})<0$. Then $\boldsymbol{S}_{i+1}$ will be discarded and the pair $(\boldsymbol{q}_{i+1},\boldsymbol{p}_{i+1})$ will not be used to update the spherical polygon $\bigcap_{j=0}^{i}HS(\boldsymbol{r}_j)$. Furthermore, instead of applying again formula (1), the candidate separating vector $\boldsymbol{S}_{i+1}$ will be regenerated by being set to a point $\boldsymbol{w}_i$ in $\bigcap_{j=0}^{i}HS(\boldsymbol{r}_j)$. Note that in the case $\bigcap_{j=0}^{i}HS(\boldsymbol{r}_j)$ is not empty, otherwise HS-jump would have reported collision between P and Q. The new supporting vertex pair $(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})$ resulting from the $\boldsymbol{S}_{i+1}$ using the above auxiliary rule has the favorite property that either (1) $(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})$ has not appeared in any preceding iteration j, $j<i$; or (2) $\boldsymbol{S}_{i+1}\cdot(\boldsymbol{p}_{i+1}-\boldsymbol{q}_{i+1})>0$, i.e., HS-jump can report that P and Q are separated. To see this, assuming that $(\boldsymbol{p}_{i+1},\boldsymbol{q}_{i+1})=(\boldsymbol{p}_j,\boldsymbol{q}_j)$ for some $j\leq i$, then

$$\boldsymbol{S}_{i+1}\cdot(\boldsymbol{q}_{i+1}-\boldsymbol{p}_{i+1})=\boldsymbol{w}_i\cdot(\boldsymbol{q}_j-\boldsymbol{p}_j)>0,$$

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

To summarize, the candidate separating vectors in HS-jump are generated with either of the two rules as follows. Suppose that $\boldsymbol{S}_i\cdot(\boldsymbol{q}_i-\boldsymbol{p}_i)<0$ in the i-th iteration. If the pair $(\boldsymbol{p}_i,\boldsymbol{q}_i)$ has not appeared in any preceding iteration, then $\boldsymbol{S}_{i+1}$ is computed using formula (1); otherwise, $\boldsymbol{S}_{i+1}$ is chosen to be any point in the spherical convex polygon $\bigcap_{j=0}^{i}HS(\boldsymbol{r}_j)$. 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, 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 $|V(P)|\times|V(Q)|$, the total number of iterations HS-jump performs for each frame is at most $2\times|V(P)|\times|V(Q)|$. In general, we found that when the $|V(P)|$ and $|V(Q)|$ are each not over 4000, the number of the iteration is not over 40. When all the vertex pairs $(\boldsymbol{p},\boldsymbol{q})$, where $\boldsymbol{p}\in V(P)$ and $\boldsymbol{q}\in V(Q)$, have been exhausted by HS-jump as supporting vertex pairs (and CH_i is still non-empty), HS-jump will declare that P and 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 P and Q be two bounded polytopes each with a finite number of vertices. Let $V(P)$ and $V(Q)$ denote the sets of vertices of P and Q, respectively. The following algorithm describes how HS-jump performs collision detection for one time frame with two (stationary) polytopes, P and 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 section 8.4.

HS-jump: Algorithm for Collision Detection of Polytopes

(1) Set a nonzero vector $\boldsymbol{S}_0$. Set $SCP_0=\boldsymbol{S}^2$. Compute the supporting vertex pair($\boldsymbol{p}_0$,$\boldsymbol{q}_0$) with respect to $\boldsymbol{S}_0$. Set iteration index $i:=0$.

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

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

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

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

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

(7) If ($\boldsymbol{p}_{i+1}$,$\boldsymbol{q}_{i+1}$)=($\boldsymbol{p}_i$,$\boldsymbol{q}_i$), report separation (Theorem 5); otherwise, set $\boldsymbol{S}_{i+1}$ to be a point in the spherical convex polygon $SCP_i\equiv\bigcap_{j=0}^{i}HS(\boldsymbol{r}_j)$ (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 $\boldsymbol{S}_i$, compute the supporting vertex pair ($\boldsymbol{p}_i$,$\boldsymbol{q}_i$)(step 1).

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

(3) Given a new added vector $\boldsymbol{r}$, compute the updated spherical polygon $SCP_{i+1}\equiv\bigcap_{j=0}^{i+1}HS(\boldsymbol{r}_j)$.

(4) Use 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[15] of polytope P with vertex set $V(P)$ is defined as a sequence of polytopes $hier(P)=\{P_0,P_1,\cdots,P_h\}$, $h=O(\log(|V(P)|)$, such that

(1) $P_0=P$ and P_h is a tetrahedron;

(2) $P_{i+1}\subset P_i$, for $0\leq i\leq h$; and

(3) $V(P_{i+1})\subset V(P_i)$, for $0\leq i\leq h$; and

the vertices in $V(P_i)-V(P_{i+1})$ form an independent set in $P_i(0\leq i\leq h)$.

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

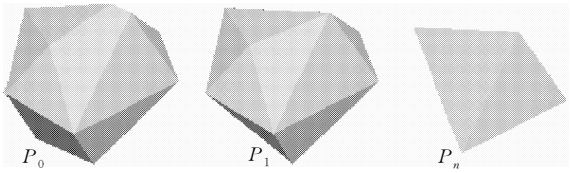


Fig. 6 A hierarchical representation of polygon

Let $\boldsymbol{p}_i$ be the supporting vertex of the polytope P_i. We first find the supporting vertex $\boldsymbol{p}_h$ of P_h by comparing its three or four vertices. Then we find the supporting vertex $\boldsymbol{p}_{h-1}$ of P_{h-1} by comparing the adjacent vertices of $\boldsymbol{p}_h$ in P_{h-1} (note that $\boldsymbol{p}_h$ is also a vertex of P_{h-1}). The correctness of this search is proved in paper [15]. Continuing this search through the hierarchy, finally we obtain the supporting vertex $\boldsymbol{p}_0$ of P_0, i.e., a supporting vertex $\boldsymbol{p}$ of P. Similar to this, we may find the supporting vertex $\boldsymbol{q}$ of Q, using $O(\log(V(P)+\log(V(Q))$ time.

8.2 Checking the re-appearance of a supporting vertex pair

We assign to each supporting vertex pair ($\boldsymbol{p}_i$, $\boldsymbol{q}_i$) a unique index $n=i\times N+j$, here N is the number of vertices of the polytope 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 Updating spherical polygon $\boldsymbol{S}_i$

We use a balanced binary tree to store the vertices of the spherical convex polygon 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 SCP_i; thus $k<i$. For each newly generated candidate separating vector $\boldsymbol{r}_{i+1}$, if the open-hemisphere $HS(\boldsymbol{r}_{i+1})$ does not intersect SCP_i, we can declare that the polytopes collide since SCP_{i+1} would be empty; otherwise, the two intersection points of SCP_i and the boundary of $HS(\boldsymbol{r}_{i+1})$, which is a great circle, can be found in $O(\log k)$ time, and updating SCP_i into 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 $\boldsymbol{S}$ and the associated supporting vertex pair $(\boldsymbol{p}, \boldsymbol{q})$. At the next frame, we may use $\boldsymbol{S}$ as the initial candidate separating vector when we search for a separating vector of the polytope at the new positions. To get the supporting vertex pair, we can start from $\boldsymbol{p}$ and $\boldsymbol{q}$ separately, and get the supporting vertex of P and Q by local and hierarchical searching. We first search the supporting vertex by local search on P_0 (P). If we cannot get the supporting vertex, local searching is performed on P_1 at the corresponding vertex; if the supporting vertex on P_1 still cannot be found P_2 will be searched, etc. At last we may enter P_n, where the supporting vertex can be obtained quickly because there are very few vertices to be searched. Once a supporting vertex is found on P_k, where $k \leq n$, we can obtain the supporting vertex of P by hierarchical searching. The supporting vertex of Q can be found similarly and the new supporting vertex pair is formed. Since the position and orientation of the moving objects under consideration change little between adjacent time frames, the new supporting vertex pairs are obtained by local search on few hierarchical polytopes. So using between-frame coherence, we can speed up the implement of the algorithm. If the polytopes P and Q collide at the current time frame, since we have recorded all processed $\boldsymbol{S}_i$ and $(\boldsymbol{p}_i, \boldsymbol{q}_i)$, the new supporting vertex pair $(\boldsymbol{p}_i, \boldsymbol{q}_i)$ with respect to $\boldsymbol{S}_i$ can be found quickly. With the same reasoning, the spherical cap determined by these $(\boldsymbol{p}_i, \boldsymbol{q}_i)$ is almost empty, and HS-jump reports that P and Q is collision.

9 Experiments

In this section we present some experimental results of using HS-jump to detect collision of two ellipsoids. The program is implemented in C++ and run on a PC with Pentium II 233MHz CPU. We compare our algorithm with two commonly used collision detection algorithm: GJK[13] and I-COLLIDE[7]. The codes of these algorithms are ported from the UNIX versions available from their authors' websites.

Performance evaluation and comparison of collision detection algorithms are somewhat 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 polyhedron is approximated by an ellipsoid, 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 first study the algorithms using two static objects. We use 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 if $d<0$, it is the distance that one object must be moved in one direction so that the two objects become disjoint.

For any fixed distance, one object is fixed and the other assumes 500 randomly generated orientations; thus 500 pairs of objects are generated for each fixed distance. Each collision detection algorithm is then run 500 times for each pair of objects with different orientations to get the average time of the collision detection method 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 size of the ellipsoid along three principle axes are $a=200$, $b=180$, $c=180$. The curves in Figure 7 show the time of the three algorithms with respect to distance. The middle curve (marked 1) is for HS-jump, the tint one (marked 2) for GJK, and the top one (marked 3) for I-COLLIDE.

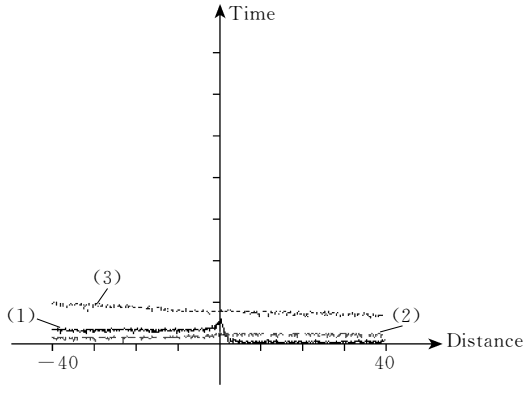


Fig. 7　$N=200$, $a:b=10:9$

Figure 8 and 9 show the comparison of three

methods using ellipsoids of the same size but with a varying number of vertices. 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 $a=200$, $b=180$, $c=180$.

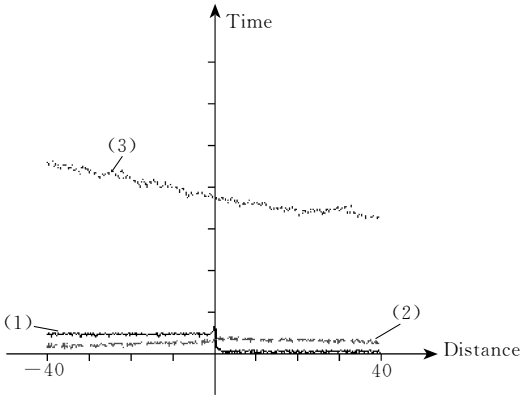


Fig. 8 $N=1000$, $a:b=10:9$

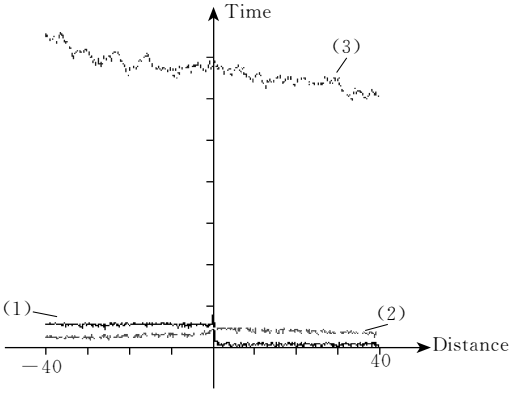


Fig. 9 $N=2000$, $a:b=10:9$

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

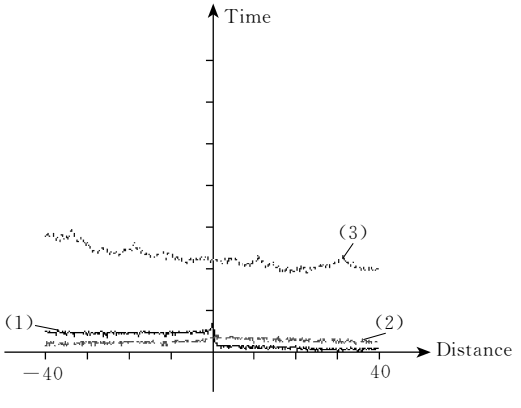


Fig. 10 $N=1000$, $a:b=10:5$

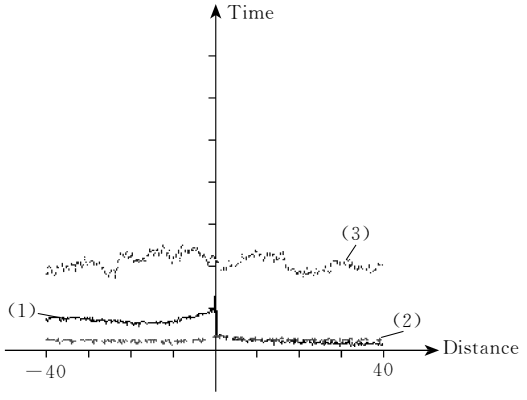


Fig. 11 $N=1000$, $a:b=10:1$

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 running times of the two algorithms are almost the 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 determine the shortest distance of two disjoint objects, but HS-jump only reports that two objects are disjoint. However, we can also compute the shortest distance quickly with the help of the information obtained by HS-jump. Although we have not experienced 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 increases quickly when the number of vertices of an object increases, but the time of HS-jump increases relatively slowly. 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 optimal case of two spherical objects.

Howerer the shape of an object has little effect on the GJK method.

10 Conclusions

We have presented an efficient exact collision detection algorithm for 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 spherical 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 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 Moore M, Wilhelms J. Collision detection and response for computer animation. In: Proceedings of Computer Graphics (SIGGRAPH'88), Atlanta, Georgia, 1988. 289～298

2 Gottschalk S, Lin M C, Manocha D. OBBTree: A hierarchical structure for rapid interference detection. In: Proceedings of Computer Graphics (SIGGRAPH'96), New Orleans, Louisiana, 1996. 171～180

3 Klosowski J, Held M, Mitchell J S B, Sowizral H, Zikan K. Efficient collision detection using bounding volume hierarchies of k-dops. In: Proceedings of Siggraph'96, 1996. 151

4 Cameron S. Collision detection by four-dimensional intersection testing. IEEE Transactions on Robotics and Automation, 1990,6(3):291～302

5 George Baciu, Wong S-K. Rendering in object interference detection on conventional graphics workstations. In: Proceedings of Pacific Graphics, Seoul, Korea, 1997. 51～58

6 Myszkowski K, Myszkowski O G, Okunev T L Kunuii. Fast collision detection between complex solids using rasterizing graphics hardware. The Visual Computer, 1995,11(9): 497～511

7 Cohen J D, Cohen M C, Lin D, Manocha M K Ponamgi. I-COLLIDE: An interactive detection system for large-scale environments. In: Proceedings of ACM Interactive 3D Graphics Conference, Monterey, California, 1995. 189～196

8 Lin M, Manocha D. Fast interference detection between geometric models. The Visual Computer, 1995,11(10): 542－561

9 Dobkin D P, Kirkpatrick D G. Determining the separation of preprocessed polyhedra——A unified approach. In: Proceedings of the 17th International Colloquium, Automata Lang. Program, 1990. 400～413

10 Megiddo N. Linear-time algorithms for linear programming in R^3 and related problems. SIAM Journal of Computing, 1983, 12(4): 759－766

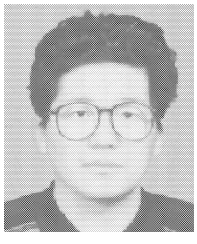
11 Seidel R. Linear programming and convex hulls made easy. In: Proceedings of the 6th Annel ACM Conference on Computational Geometry, Berkeley, California, 1990. 211－215

12 Gilbert E G, Johnson D W, Keerthi S S. A fast procedure for computing the distance between objects in three-dimensions space. IEEE Transactions on Robotics and Automation, 1988, RA-4:193－203

13 Stephen Cameron. A comparison of two fast Algorithms for computing the distance between convex polyhedra. IEEE Transactions on Robotics and Automation, 1997,13(6): 915－920

14 Kelvin Chung, Wenping Wang. Quick collision detection of polytopes in virtual environments. In: Proceedings of ACM Symposium on Virtual Reality Software and Technology, Hongkong, 1996. 125－131

15 Joseph o'Rourke. Computational Geometry in C. Cambriged: Cambriged University Press, 1993. 253－265



LI Xue-Qing, born in 1964, Ph. D., associate professor. His research interests include computational geometry compute graphics, computer-human interaction.

MENG Xiang-Xu, born in 1962, Ph. D., professor. His research interests include compute graphics, virtual reality.

WANG C Y, male, born in 1937, professor. His research interests include computational geometry, compute graphics.

WANG Wen-Ping, born in 1963. Ph. D., associate professor. His research interests include computational geometry compute graphics.

CHUNG Kelvin, born in 1970, master. His research interests include collision detection.

YIU Siu Ming, born in 1968, Ph. D.. His research interests include algorithm.