

# Intersection of Convex Objects in Two and Three Dimensions

B. CHAZELLE

*Yale University, New Haven, Connecticut*

AND

D. P. DOBKIN

*Princeton University, Princeton, New Jersey*

**Abstract.** One of the basic geometric operations involves determining whether a pair of convex objects intersect. This problem is well understood in a model of computation in which the objects are given as input and their intersection is returned as output. For many applications, however, it may be assumed that the objects already exist within the computer and that the only output desired is a single piece of data giving a common point if the objects intersect or reporting no intersection if they are disjoint. For this problem, none of the previous lower bounds are valid and algorithms are proposed requiring sublinear time for their solution in two and three dimensions.

Categories and Subject Descriptors: E.1 [Data]: Data Structures; F.2.2 [Analysis of Algorithms]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Theory, Verification

Additional Key Words and Phrases: Convex sets, Fibonacci search, Intersection

## 1. Introduction

This paper describes fast algorithms for testing the predicate,

*Do convex objects  $P$  and  $Q$  intersect?*

where an object is taken to be a line or a polygon in two dimensions or a plane or a polyhedron in three dimensions. The related problem

*Given convex objects  $P$  and  $Q$ , compute their intersection*

has been well studied, resulting in linear lower bounds and linear or quasi-linear upper bounds [2, 4, 11, 15–17]. Lower bounds for this problem use arguments

This research was supported in part by the National Science Foundation under grants MCS 79-03428, MCS 81-14207, MCS 83-03925, and MCS 83-03926, and by the Defense Advanced Project Agency under contract F33615-78-C-1551, monitored by the Air Force Office of Scientific Research. The research was facilitated by the use of Theory Net, NSF grant MCS 78-01689. Portions of this research appeared in Chazelle's Ph.D. dissertation for Yale University, entitled "Computational Geometry and Convexity," and appeared in the *Proceedings of the 12th Annual ACM Symposium on the Theory of Computing* (Los Angeles, Calif., May). ACM, New York, 1980, pp. 146–153.

Authors' current address: Department of Computer Science, Princeton University, Princeton, NJ 08544.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0004-5411/87/0100-0001 \$00.75

INTERSECTED	LINE	POLYGON	PLANE	POLYHEDRON
LINE	constant	$\log n$	constant	$\log^2 n$
POLYGON		$\log n$	$\log n$	$\log^2 n$
PLANE			constant	$\log^2 n$
POLYHEDRON				$\log^3 n$

FIG. 1. The time bounds of our algorithm.

claiming that linear time is required to read all inputs or report the output. For the problem that we pose, such arguments do not apply. We only require a witness to the intersection or nonintersection of  $P$  and  $Q$ , and we further assume that the objects we wish to intersect are available (i.e., in random-access memory [1]) so that we cannot rely on input time to yield a linear lower bound.

Figure 1 summarizes our results, most of which are fully original. The time bounds are achieved by using the standard array representation for two-dimensional objects and a special representation of polyhedra that requires  $O(n^2)$  operations to reach from the standard representation (where  $n$  denotes the total number of vertices). An  $O(n \log n)$  preprocessing of the standard representation is actually sufficient, but the running times given here must then be multiplied by a  $\log n$  factor [7]. Note that convex polyhedra have the structure of planar graphs, so the number of vertices, edges, and faces are linearly related, and any of these measures can be used to represent the input size. Although the times given in Figure 1 are asymptotic, the constants involved are sufficiently small to make the algorithms viable in practice. Furthermore, many of the applications for which such algorithms might be used require a knowledge of only portions of the intersection or of the existence of an intersection, rather than a complete description of any intersection (A. Forrest, private communication). For example, in computer graphics, when we wish to clip or window a scene [12], algorithms of the form given here would be sufficient for identifying the polygons that would require further processing. Also, many applications for which such algorithms might be used in design rule checking for VLSI [4], computer geography [8], (D. Tomlin, private communication), computer-aided design, and computer animation require a gross procedure that detects the possibility of an intersection, from which refined procedures can handle the small number of cases in which an intersection has been reported and must be computed.

All these algorithms rely on a small number of unifying concepts. Convexity combined with random-access capabilities allows for binary and Fibonacci search, and it is with an explanation of these basic principles that we start our analysis. Section 2 is devoted to the two-dimensional case, while Section 3 investigates the problem cast in three dimensions.

## 2. Computing Planar Intersections

2.1 NOTATION. Polygons are represented by arrays with their vertices given in clockwise order. Polygon  $P$  will have vertices  $p_1, \dots, p_p$  and polygon  $Q$  vertices  $q_1, \dots, q_q$ . We assume that no three vertices of a polygon are collinear. All indices of  $P$  (respectively,  $Q$ ) are taken modulo  $p$  (respectively,  $q$ ) in the obvious fashion. A line is specified by any two of its points and a segment by its two endpoints.  $AB$  always refers to the segment from  $A$  to  $B$ , and “line( $AB$ )” represents the infinite line containing  $AB$ . We define  $d(x, L)$  as the orthogonal distance from the point  $x$

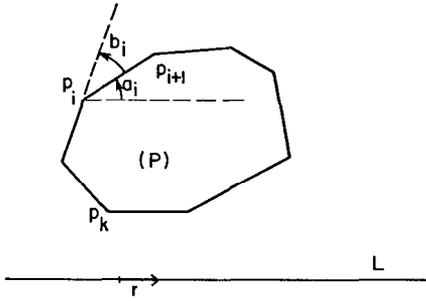


FIG. 2. The distance from a convex polygon to a line is bimodal.

to the line  $L$  and  $h(x, L, v)$  as the oriented distance from  $x$  to  $L$  with respect to point  $v$ . This latter quantity is defined as  $-d(x, L)$  if  $x$  and  $v$  lie on opposite sides of  $L$  and as  $d(x, L)$  if they lie on the same side. Both  $d$  and  $h$  can be computed in constant time.  $F_i$  represents the  $i$ th Fibonacci number with  $F_0 = F_1 = 1$  and  $F_N = F_{N-1} + F_{N-2}$ , for  $N > 1$ .

**2.2 FIBONACCI SEARCH ON BIMODAL FUNCTIONS.** A real function  $f$  defined on the integers  $1, 2, \dots, n$  is said to be *unimodal* if there exists an integer  $m$  ( $1 \leq m \leq n$ ) such that  $f$  is strictly increasing (respectively, decreasing) on  $[1, m]$  and decreasing (respectively, increasing) on  $[m + 1, n]$ , with  $f(m) \geq f(m + 1)$  (respectively,  $f(m) \leq f(m + 1)$ ). Kiefer [9] showed that Fibonacci search was an optimal method of finding  $m$ , the turning point of a unimodal function, requiring  $1.44 \dots \log n$  probes. We extend his algorithm to find the turning points of a bimodal function. For our purposes, it suffices to define a bimodal function as one for which there is an  $r$  in  $[1, n]$  such that  $f(r), f(r + 1), \dots, f(n), f(1), \dots, f(r - 1)$  is unimodal. Our interest in bimodal functions stems from the following:

**LEMMA 1.** Let  $P$  be a convex polygon with  $p$  vertices  $p_1, \dots, p_p$  in clockwise order. For any line  $L$  and any point  $v$  not in  $L$ , the function defined for  $i = 1, \dots, p$  by  $f(i) = h(p_i, L, v)$  is bimodal.

**PROOF.** Let  $p_k$  be the vertex of  $P$  that minimizes  $f(i)$  for  $i = 1, \dots, p$ . In case of a tie, we choose  $k$  so that the only other integer that achieves the same value of  $f$  is  $k - 1$ . We can do this because  $P$  is convex. We show that the sequence  $f(k), f(k + 1), \dots, f(k - 1)$  is unimodal, which suffices to prove the lemma. Let us choose a directing vector  $r$  of the line  $L$  such that the angle  $(r, p_k p_{k+1})$  is less than  $180$ . All angles are measured between  $0$  and  $360$  degrees in a counterclockwise motion. We define the oriented angles  $a_i = (r, p_i p_{i+1})$  and  $b_i = (p_i p_{i+1}, p_{i-1}, p_i)$  for  $i = 1, \dots, p$  as in Figure 2. By construction, the following relations hold for all  $i$ :

$$f(i + 1) = f(i) + |p_i p_{i+1}| \sin a_i;$$

$$a_{i+1} = a_i - b_{i+1} [\text{mod } 360].$$

Since  $P$  is convex, all  $b_i$  are less than  $180$  degrees; therefore the sequence  $\sin(a_k), \sin(a_{k+1}), \dots, \sin(a_{k-1})$  will be positive then negative, thus showing that  $f(k), f(k + 1), \dots, f(k - 1)$  is unimodal.  $\square$

Since a unimodal sequence has exactly one maximum and one minimum, and each of them is achieved in at most two points which must be consecutive modulo  $n$ , bimodal functions have the same property. However, finding the extrema of a bimodal function may not be so easy, since we do not know the starting point of its unimodal sequence in advance. To circumvent this difficulty, given a bimodal

function  $f$ , we construct a unimodal function  $g$  as follows: First, let  $T$  be the line through the points  $(1, f(1))$  and  $(n, f(n))$ , that is,

$$T(x) = \frac{x-1}{n-1} (f(n) - f(1)) + f(1).$$

If  $f(1) = f(n)$ , then  $f(1)$  is an extremum and  $f$  is unimodal, showing that the extrema can be found with the previous method. Otherwise, we assume that  $f(1) < f(n)$  (the case  $f(1) > f(n)$  being similar). If  $f(2) \geq f(1)$ , then  $f(1)$  is a minimum and the subsequence  $f(2), \dots, f(n)$  is unimodal, which solves our problem. Else, if  $f(2) < f(1)$ , the function  $g$  defined by

$$g(x) = \min(f(x), T(x))$$

can be evaluated in constant time and is unimodal. This follows, since for all  $x$ ,  $g(x) \leq f(n) \leq \max(f(t))$ , so  $g$  first decreases, then rises, and is therefore unimodal. It follows that the minimum of  $g$  (which is also the minimum of  $f$ ) can be found with a Fibonacci search. If  $x$  is the point at which  $g$  achieves its minimum, the sequence  $f(x+1), f(x+2), \dots, f(n)$  is unimodal, and the maximum of  $f$  can also be determined through a second Fibonacci search, yielding

LEMMA 2. *The extrema of a bimodal function  $f(1), \dots, f(n)$  can be computed in  $O(\log n)$  time, which involves at most  $2.88 \dots \log_2 n + O(1)$  function evaluations.*

2.3 INTERSECTION OF A LINE WITH A CONVEX POLYGON (IGL). Combining previous facts yields an algorithm for determining the intersection (null, 1 point, a segment, or an edge of  $P$ ) of an infinite line  $L$  and a convex polygon  $P$ .

THEOREM 3. *The intersection of an infinite line with a convex polygon with  $p$  vertices can be computed in  $O(\log p)$  time.*

PROOF. We can always assume that  $p_1$  does not lie on  $L$ . Then it follows from Lemma 1 that the function  $f(p_i) = h(p_i, L, p_1)$  is bimodal; therefore, the algorithm of Lemma 2 allows us to find a vertex  $w$  of  $P$  that minimizes  $f$ . We know that  $P$  and  $L$  intersect if and only if  $f(w)$  is negative or zero. In the latter case,  $w$  or an edge including  $w$  is the unique intersection of  $P$  and  $L$ . In the former case, the signs of  $f(p_1), f(p_2), \dots, f(w)$  and  $f(w), f(w+1), \dots, f(p_p)$  can be searched by binary search to determine  $i$  and  $j$  such that  $f(p_i) \geq 0 > f(p_{i+1}), f(p_j) \leq 0 < f(p_{j+1})$  from which the two points of intersection are determined.  $\square$

Our algorithm involves approximately  $2.8808 \log_2 p + O(1)$  computations of  $f$ . The extension to the case in which  $L$  is a line segment does not increase the time bound. Since  $O(\log p)$  has been shown to be a lower bound on the time complexity for testing the inclusion of a point in a convex polygon, which is constant time reducible to our problem, the algorithm we have described is optimal in the minimax sense [16]. In what follows, we refer to this algorithm as IGL. Though our algorithms are more complex than IGL, they are based on principles similar to those used to derive this algorithm.

2.4 INTERSECTION OF TWO CONVEX POLYGONS (IGG). The algorithm for computing the intersection of a line and a polygon suggests methods that might be used to speed up algorithms for intersecting two polygons  $P$  and  $Q$ . If we could determine the sides of  $P$  closest to  $Q$  (and vice versa), we would be able to reduce the problem to a small number of tests of segment intersections. Our method reduces the number of remaining edges of one of the polygons by a factor of 2 at each iteration. The algorithm we present (referred to as IGG) returns NO if  $P$  and

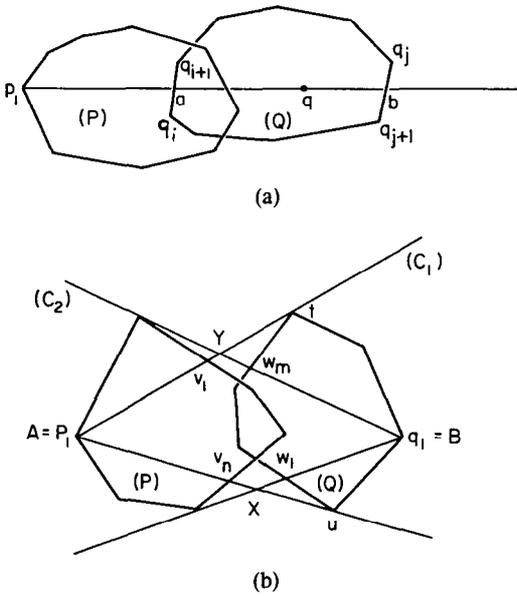


FIG. 3.  $AYBX$  formed as a bounding quadrilateral in which  $P \cap Q$  lies if it exists.

$Q$  do not intersect and (YES,  $K$ ) if they do, where  $K$  is a point of their intersection. When a NO answer is returned, it is possible to generate in constant time a pair of parallel lines that separate the polygons.

We begin by limiting the intersection of  $P$  and  $Q$  to a quadrilateral or a pentagon. This requires  $O(\log pq)$  steps and also tests for simple intersections (e.g.,  $P$  contained in  $Q$ ). From the quadrilateral we form two chains of vertices  $L_v$  and  $L_w$  that intersect if and only if  $P$  and  $Q$  intersect. The iterative step of the algorithm is a division in which we eliminate half of either  $L_v$  or  $L_w$ . This step is reached as one of five possible cases determined from the structure of the remaining vertices.

It is important to keep in mind that by intersection of  $P$  and  $Q$  we mean the intersection of the regions  $P$  and  $Q$  and not of the polygonal boundaries. We shall prove further that the latter problem requires linear time, whereas our problem can be solved in  $O(\log(p + q))$  time. We first give a description of the algorithm and prove its correctness, and then we establish its running time.

**Algorithm IGG (intersecting two polygons):**

(I) "Cover  $Q$  (respectively,  $P$ ) with two lines of support intersecting in  $P$  (respectively,  $Q$ )."

(a) Let  $q$  be a point interior to  $Q$  (say the center of mass of three vertices) such that  $q$  is not interior to  $P$  (if  $q$  is interior to  $P$ , we have found an intersection). Compute the intersection of  $Q$  with line  $(p_1q)$ . This line always intersects  $Q$  in two points  $a$  and  $b$ , which the algorithm IGL can find, as well as the edges of  $Q$  where  $a$  and  $b$  lie, say  $q_iq_{i+1}$  and  $q_jq_{j+1}$ , respectively (see Figure 3a).

(b) If  $p_1$  lies on the segment  $ab$ , it also lies in  $Q$  and the algorithm can return (YES,  $p_1$ ). Otherwise, we do a Fibonacci search on the sequence of oriented angles  $(p_1q, p_1q_k)$ , for all  $q_k$  between  $q_{i+1}$  and  $q_j$  in clockwise order, in order to find the maximum angle. Call  $t$  the corresponding vertex of  $Q$ . Such a Fibonacci search is legitimate since the sequence is unimodal. If it were not, we could find an ordered list of three consecutive vertices of  $Q$  with the angle relative to the middle vertex smaller than both of the others. Then the line joining  $p_1$  to this vertex would cut  $Q$  in more than two points, contradicting the convexity of  $Q$ . Similarly, by

considering the sequence  $q_{j+1}, \dots, q_i$ , we find the vertex  $u$  which minimizes the angle  $(p_1q, p_1q_k)$ . Call  $C_1$  the pencil  $(p_1u, p_1t)$  so defined (see Figure 3b).

(c) Apply the previous procedure (Steps a, b) with  $q_1$  relative to  $P$ . If the algorithm does not return, it will determine another pencil,  $C_2$ , centered in  $q_1$  and covering  $P$ . Since  $C_1$  (respectively,  $C_2$ ) contains  $q_1$  (respectively,  $p_1$ ), the intersection of  $C_1$  and  $C_2$  is a convex quadrilateral,  $p_1Yq_1X$ , as shown in Figure 3b. Note that  $X$  or  $Y$  may not be defined, in which case we can replace the missing intersection by a segment joining the two pencils, thus obtaining a pentagon.

(II) Note that the portion of the boundary of  $P$  that lies in  $C_1$  is a contiguous polygonal line from the intersection of  $p_1X$  and  $P$  to the intersection of  $p_1Y$  and  $P$ , and lies also in  $C_2$ . Determine its two endpoints (note that one or even both of these endpoints may be  $p_1$ ). Renumber the vertices of  $P$  so that  $L_v = \{v_1, \dots, v_n\}$  gives the vertices of this polygonal line in clockwise order (we have  $v_1$  on  $p_1Y$  and  $v_n$  on  $p_1X$ ). Throughout this section, any renumbering is implicit, that is, does not involve any scan through the vertices. It may simply consist of the setting of an arithmetic expression redefining the mapping. The same procedure is carried out with  $Q$  defining  $L_w = \{w_1, \dots, w_m\}$ . In what follows, we rename the former  $p_1$  and  $q_1$ ,  $A$  and  $B$ , respectively, as in Figure 3b. Note that although  $L_v$  intersects  $AY$  and  $AX$ , it may also intersect  $BX$  or  $BY$  (in at most one point, though).

(III) We have now reduced the original problem to checking the intersection of  $L_v$  and  $L_w$ .

Let  $x$  (respectively,  $y$ ) denote the polygonal line  $AXB$  (respectively,  $AYB$ ). To simplify the exposition, for two points  $F$  and  $G$ , we say that  $F < G$  if  $F$  and  $G$  are both on  $x$  or both on  $y$  and  $F$  is on the path from  $A$  to  $G$ .

At this stage, we call upon the function  $\text{INTERSECT}(L_v, L_w)$  defined recursively as follows:

$\text{INTERSECT}(L_v, L_w)$

Assume that  $n, m > 5$ , where  $n = |L_v|$  and  $m = |L_w|$ , using the procedure of the previous section if this is not the case:

$$i = \left\lfloor \frac{n}{2} \right\rfloor; j = \left\lfloor \frac{m}{2} \right\rfloor.$$

Let  $F$  and  $G$  (respectively,  $E, H$ ) denote the two intersections of  $\text{line}(v_i v_{i+1})$  (respectively,  $\text{line}(w_j w_{j+1})$ ) with the boundary  $AYBXA$ . The point  $F$  (respectively,  $H$ ) is chosen such that  $v_{i+1}$  (respectively,  $w_{j+1}$ ) lies on the segment  $v_i F$  (respectively,  $w_j H$ ) (see Figure 4a).

The algorithm distinguishes between cases depending on the relative positions of  $GF$  and  $EH$ . Each case reduces the size of  $L_v$  and/or  $L_w$ , after which a recursive call is made. (Cases are not mutually exclusive and each should be tested.)

*Case 1.* Either  $GF$  or  $EH$  lies on the same side of  $AB$  (Figure 4a).

```

if  $G$  and  $F$  lie on  $x$ 
  then  $L_v = \{v_1, \dots, v_{i+1}, v_n\}$ 
  else if  $G$  and  $F$  lie on  $y$ 
    then  $L_v = \{v_1, v_i, \dots, v_n\}$ 
if  $E$  and  $H$  lie on  $x$ 
  then  $L_w = \{w_1, w_j, \dots, w_m\}$ 
  else if  $E$  and  $H$  lie on  $y$ 
    then  $L_w = \{w_1, \dots, w_{j+1}, w_m\}$ 

```

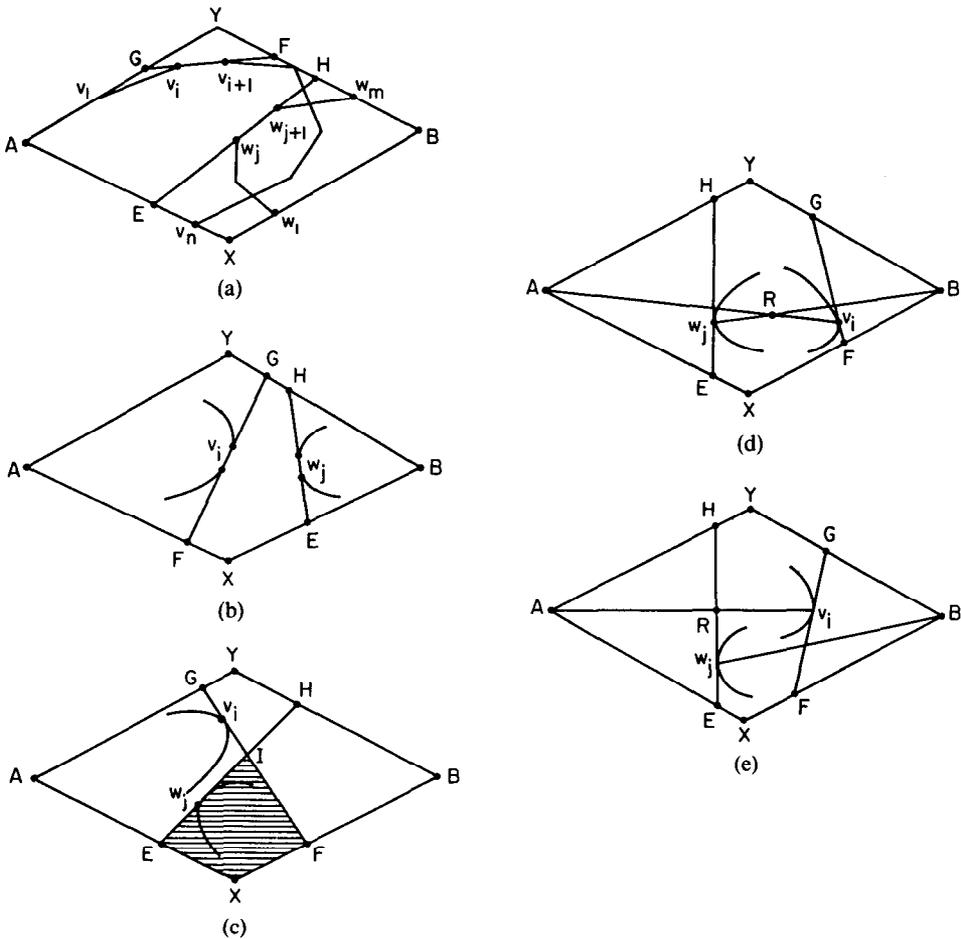


FIG. 4. The algorithm INTERSECT. (a) Case 1.  $GF$  ( $EH$ , respectively) lies on the same side of  $AB$ , in which case half of  $L_v$  ( $L_w$ , respectively) is eliminated. (b) Case 2.  $A, F, E, B$  and  $A, G, H, B$  occur in this order on  $x$  and  $y$ , respectively, in which case there is no intersection. (c) Case 3.  $GF$  and  $EH$  intersect. (d) Case 4.  $Av_i$  and  $Bw_j$  intersect, in which case  $P \cap Q$  contains their intersection point. (e) Case 5.  $Av_i$  lies strictly "above" (or "below")  $Bw_j$ .

Case 2. From now on,  $F$  and  $E$  (respectively,  $G$  and  $H$ ) lie on  $x$  (respectively,  $y$ ) (Figure 4b).

if  $F < E$  and  $G < H$  then return (NO)

Case 3. If the segments  $GF$  and  $EH$  intersect, let  $I$  be this intersection (Figure 4c).

if  $G < H$  and  $E < F$   
 then if  $v_i$  lies on  $GI$   
     then  $L_v = \{v_1, v_i, \dots, v_n\}$   
     if  $w_{j+1}$  lies on  $HI$   
         then  $L_w = \{w_1, \dots, w_{j+1}, w_m\}$   
 if  $H < G$  and  $F < E$   
 then if  $v_{i+1}$  lies on  $FI$   
     then  $L_v = \{v_1, \dots, v_{i+1}, v_n\}$   
     if  $w_j$  lies on  $EI$   
         then  $L_w = \{w_1, w_j, \dots, w_m\}$

*Case 4.*  $Av_i$  and  $Bw_j$  intersect (Figure 4d).

if  $Av_i$  and  $Bw_j$  intersect in  $R$   
 then return (YES,  $R$ )  
 else

*Case 5.* Let  $R$  be the intersection of  $Av_i$  and  $HE$  (Figure 4e).

if  $w_j$  lies on  $ER$   
 then  
 $L_v = \{v_1, v_i, \dots, v_n\}$   
 $L_w = \{w_1, w_j, \dots, w_m\}$   
 else  
 $L_v = \{v_1, \dots, v_{i+1}, v_n\}$   
 $L_w = \{w_1, \dots, w_{j+1}, w_m\}$

Recursive call with parameters of smaller size.

INTERSECT ( $L_v, L_w$ )

Next we show that INTERSECT runs correctly within the given time bound. For correctness, it suffices to show that INTERSECT( $L_v, L_w$ ) indeed tests for the intersection of  $L_v$  and  $L_w$  and possibly outputs a point common to  $P$  and  $Q$ .

*Case 1.* Suppose that  $G$  and  $F$  lie on  $y$  (the three other cases being similar) (see Figure 4a). By construction, line( $BY$ ) intersects  $P$  at exactly one point, which lies on the same side of  $B$  as  $Y$ . Now, since  $P$  lies totally on the same side of line( $GF$ ) as  $X$ , the intersection of  $P$  with line( $BY$ ) lies on the segment  $BF$ . Therefore, if  $L_v$  and  $L_w$  intersect, at least one intersection point lies on the polygonal line  $\{v_i, \dots, v_n\}$ . By making  $L_v$  equal to  $\{v_1, v_i, \dots, v_n\}$ , we reset the initial conditions required by the algorithm. Moreover, we note that since the region delimited by the new setting of  $L_v$  is included in  $P$ , any intersection point later output will surely be in  $P$ . This remark prevails in all the remaining cases.

Consider the two polygons delimited by  $(A, x, FG, y)$  and  $(B, x, EH, y)$  and call  $V$  their intersection (see Figure 4b). Since  $P$  and  $Q$  are convex, their intersection lies totally in  $V$ .

*Case 2.* Corresponds to  $V$  empty (see Figure 4b).

*Case 3.* The first if statement supposes that  $E$  and  $F$  belong to  $V$  and the other that  $H$  and  $G$  belong to  $V$ . Since both cases are similar, we treat only the first. Suppose that  $v_i$  does not lie in  $V$ . Then, since  $Gv_i$  lies outside of  $EHyBxE$ ,  $L_w$  cannot intersect this segment; therefore, if  $L_w$  intersects the polygonal line  $v_1, \dots, v_i$ , it also intersects  $v_1v_i$ . Thus the new setting of  $L_v$  is legitimate. The same is true of  $w_{j+1}$ . From now on, we know that both  $v_iv_{i+1}$  and  $w_jw_{j+1}$  lie on the boundary of  $V$ .

*Case 4.* Assumes that  $Av_i$  and  $Bw_j$  intersect (see Figure 4d). Since these two segments lie in  $P$  and  $Q$ , respectively, their intersection lies in the intersection of  $P$  and  $Q$ , which is then nonempty.

*Case 5.* First, we note that since  $E$  lies on  $x$  and  $v_i$  lies in  $V$ ,  $R$  is well defined. We also know that  $Av_i$  and  $Bw_j$  do not intersect. The algorithm supposes successively that  $Av_i$  lies "above" and "below"  $Bw_j$ . The two cases being similar, we treat only the first.  $L_w$  cannot intersect the polygonal line  $v_1, \dots, v_i$  without first crossing  $v_1v_i$ . Similarly,  $L_v$  cannot intersect  $w_1, \dots, w_j$  without first crossing  $w_1w_j$ . Conversely, if either  $L_w$  crosses  $v_1v_i$  or  $L_v$  crosses  $w_1w_j$ , the intersection belongs to both  $P$  and  $Q$ . Finally, since  $w_1, \dots, w_j$  (respectively,  $v_1, \dots, v_i$ ) cannot intersect  $v_1v_i$  (respectively,  $w_1w_j$ ), the new setting of  $L_v$  and  $L_w$  is legitimate.

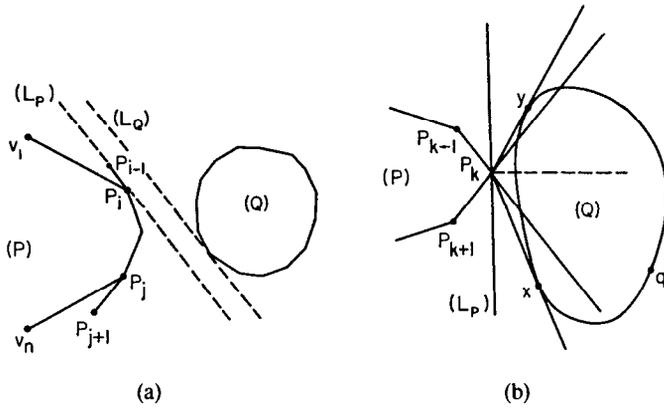


FIG. 5. Computing a pair of separating lines.

To prove the time bound, we observe that the algorithm runs in constant time between consecutive recursive calls. Every call reduces the size of one or both polygonal lines by roughly half, and when either becomes smaller than 6, the algorithm returns after  $O(\log(p + q))$  operations. Therefore, the main algorithm detects the intersection of  $P$  and  $Q$  in  $O(\log(p + q))$  time.

We can regard the intersection of a line with a polygon as a special case of this problem, and the results of the preceding section show that the algorithm described above is optimal in the minimax sense.

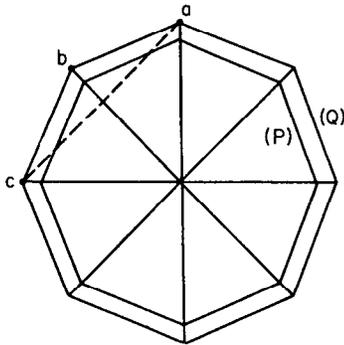
We have achieved our main goal. However, we now wish to refine the algorithm IGG so that it produces a pair of parallel separating lines  $(L_P, L_Q)$  when it fails to detect an intersection. We have preferred to present this procedure separately since there are applications in which this additional information is not needed. Instead of a complicated formal definition, Figure 5a best illustrates what we mean by a pair of separating lines.

Recall that the algorithm IGG fails to detect an intersection in two cases:

(1) It falls into case 2 of the INTERSECT procedure (see Figure 4b). Since  $P$  does not intersect  $Q$ , as is easily checked,  $AyBx$  must be a bounded quadrilateral, and so,  $EH$  joins  $BX, BY$ , or  $FG$  joins  $AX, AY$ . Assuming the former (without loss of generality), we find that line( $EH$ ) is a separating line  $L_Q$ . To compute  $L_P$ , we observe that it passes through the vertex of  $P$ , which minimizes the distance to  $L_Q$ . This distance is a bimodal function of the vertices of  $P$ ; therefore,  $L_P$  can be determined in  $O(\log p)$  time.

(2) Either  $L_v$  or  $L_w$  (say  $L_v$ ) is reduced to fewer than six vertices ( $n < 6$ ). We say that the intersection of line( $p_i p_{i+1}$ ) with  $Q$  is positive, if it is not empty, and lies entirely on the same side of  $p_i$  as  $p_{i+1}$ . If it is not empty and lies totally on the same side of  $p_{i+1}$  as  $p_i$ , it is called negative. It is clear that if  $P$  and  $Q$  do not intersect, any intersection of line( $p_i p_{i+1}$ ) with  $Q$  (called  $Q_i$ ) is positive, negative, or empty. The algorithm proceeds in stages, each consisting of the reduction of one or both polygonal lines  $L_v, L_w$ . Let  $v_1 = p_k$ ; at any stage, we show that, if  $v_2 = p_l$ , then, for each  $u$  between  $k$  and  $l - 1$ , the intersection  $Q_u$  is either empty or positive. Starting with the obvious observation that initially  $v_1$  and  $v_2$  are consecutive around  $P$  ( $l = k + 1$ ) and, therefore, that the fact is true at the first stage, we prove the assertion by induction on the number of stages. Clearly, the only stages of interest are those that reduce  $L_v$  from  $\{v_1, \dots, v_n\}$  to  $\{v_1, v_i, \dots, v_n\}$ . As before, let  $v_1$  be  $p_k$  and  $v_2$  be  $p_l$ , and let  $v_i$  be  $p_h$ . Using the induction hypothesis, it suffices to show that  $Q_u$  is empty or positive for each  $u$  between  $l$  and  $h - 1$ . Assume that one of them is negative. Then the intersection must occur in the triangle  $v_1 G v_i$ : A trivial

FIG. 6. Intersecting polygonal lines.



examination of case 3 shows this to be impossible. Cases 2 and 4 are ruled out by assumption. As to cases 1 and 5, the presence of  $Q_u$  in the triangle  $v_1 G v_i$  implies a nonempty intersection between  $L_v$  and  $L_w$ , which is excluded.

Let us now come back to the first stage where  $L_v$  has fewer than six vertices. Let  $p_i$  be the vertex  $v_2$  and  $p_j$  the vertex  $v_{n-1}$ . We have just shown that  $Q_{i-1}$  is empty or positive. Similarly, a symmetric reasoning would show that  $Q_j$  is empty or negative. It follows that, if some  $Q_k$  among  $Q_{i-1}, \dots, Q_j$  (note that there are at most four of them to consider) are empty,  $L_P$  can be set to  $Q_k$  (see Figure 5a). Otherwise, there exists a pair  $(Q_{k-1}, Q_k)$  with  $Q_{k-1}$  positive and  $Q_k$  negative (see Figure 5b for the case  $k = l - 1$ ). Observing that the angles  $(p_k q_1, p_k q_l)$  are bimodal for  $l = 1, \dots, q$  (here we measure angles counterclockwise with values between  $-180$  and  $+180$  degrees), we can find the vertex  $x$  (respectively,  $y$ ) of  $Q$  that minimizes (respectively, maximizes) that angle in  $O(\log q)$  time. A simple argument shows that  $L_P$  may be set to the line passing through  $p_k$  and perpendicular to the bisector of  $(p_k x, p_k y)$ . The line  $L_Q$  is then obtained by minimizing the distance to  $P$  as we did earlier (1). We can conclude:

**THEOREM 4.** *An intersection between two convex polygons with  $p$  and  $q$  vertices, respectively, can be detected in  $O(\log(p + q))$  time. When the polygons intersect, a common point is returned, and a pair of parallel separating lines otherwise.*

Although the previous algorithm can decide whether  $P$  and  $Q$  intersect, it is unable to tell whether one polygon lies strictly inside the other, that is, whether or not the boundaries of  $P$  and  $Q$  intersect. This is because the more general problem of deciding whether two convex polygonal lines intersect requires linear time to be solved. To see this, consider two polygons  $P$  and  $Q$  given in the complex plane with vertices of  $P$  being the roots of  $z^n - 1 = 0$  and the vertices of  $Q$  the roots of

$$z^n - \left( \frac{1}{2} + \frac{1}{2 \cos(2\pi/n)} \right)^n = 0.$$

It can be easily verified that for any consecutive vertices  $a, b, c$  on the boundary of  $Q$ , neither the edge  $ab$  nor  $bc$  intersects the boundary of  $P$ , whereas the segment  $ac$  does (see Figure 6). So, any vertex of  $Q$  can be moved along a radius to create an intersection without altering any of the  $n - 1$  remaining vertices. Therefore, any algorithm checking the intersection of the boundaries of  $P$  and  $Q$  has to look at all the vertices of  $Q$  yielding the claimed lower bound. This is assuming, as usual, that the points are given in an array, with no additional information except the size of the polygons.

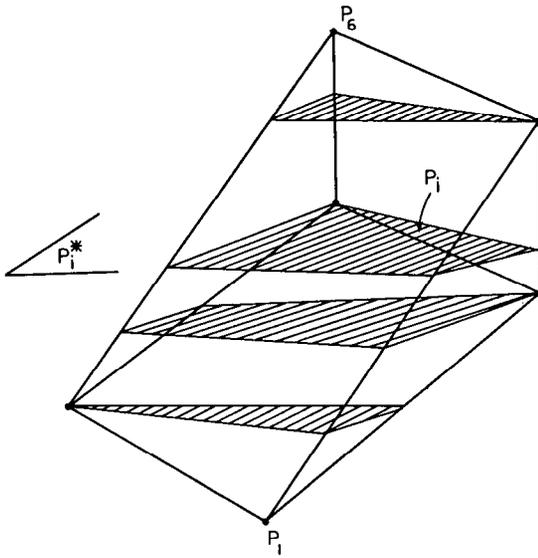


FIG. 7. Preprocessing three-dimensional objects.

**THEOREM 5.** *Testing the intersection of two convex polygonal lines requires linear time.*

Thus no general extensions of the algorithm in the plane are possible. However, there are many cases to which the algorithm can be applied to give a description of the region of intersection sufficient for its reconstruction.

### 3. Detecting Three-Dimensional Intersections

**3.1 INTRODUCTION.** Although detecting intersections becomes substantially more difficult in three dimensions, the algorithms that we are about to describe are based on principles similar to those used in the previous sections. We still use Fibonacci searches to find extrema of bimodal functions and answer questions of the form: Does object *A* lie entirely on one side of a given hyperplane? Similarly, binary searches are used to reduce the size of a problem by a constant factor.

Since all these techniques assume some random-access capabilities, we must give our three-dimensional objects a special representation to provide these features. From the observation that the surface of a convex polyhedron has the structure of a planar graph, a standard method has been to represent convex polyhedra by a description of the planar graph, along with the geometric location of the vertices [11]. Unfortunately, this representation does not meet all of our requirements, and some preprocessing is needed. We represent each polyhedron as a set of parallel convex polygons. These polygons, called *preprocessing polygons*, consist of a cross-section of the polyhedron for each vertex. Each cross-section is the intersection of the polyhedron with a plane parallel to the *xy*-plane passing through the vertex (see Figure 7). This reduces a polyhedron *P* of *p* vertices to a set of *p* (or fewer) convex polygons  $P_1, \dots, P_p$  and  $p - 1$  convex drums (we call a drum a convex polyhedron with all the vertices lying on two parallel faces). Since each drum can be tested for intersection with a convex polygon in logarithmic time, and projections and intersections of those drums with a plane give convex objects, only  $O(\log p)$  preprocessing polygons need be considered for all of our purposes, which yields the

desired results. Before describing the preprocessing more precisely, we briefly outline the various algorithms that we shall present.

(1) IHP—Intersection of a Polyhedron  $P$  with a Plane  $T$ . The projections of  $P$  and  $T$  on a plane perpendicular to  $T$  and the preprocessing polygons form, respectively, a convex polygon and a line that intersect if and only if  $P$  and  $T$  intersect. We call IGL to test the intersection. This requires  $O(\log p)$  steps, each step involving  $O(\log p)$  operations, since the access to any vertex of the projected polygon involves maximizing a linear combination of the  $x$ - or  $y$ -coordinates of a preprocessing polygon, that is, maximizing a bimodal function.

(2) IHG—Intersection of a Polyhedron  $P$  with a Polygon  $R$ . If IHP fails to detect an intersection between  $P$  and the plane  $T$  supporting  $R$ , we are finished. Otherwise,  $T$  intersects a set of consecutive preprocessing polygons which we can compute implicitly in  $O(\log^2 p)$  time by a binary search whose basic step involves intersecting a polygon with a line. Letting  $Q$  be the intersection of  $P$  and  $T$ , we first test the intersection of  $R$  with the subpolygon of  $Q$  formed by the preprocessing polygons determined earlier. If we fail, IGH will return a separating line adjacent to  $Q$ . We can show that this line is adjacent to two consecutive drums of  $P$ , which must intersect  $R$  if  $P$  does. We can test each drum for intersection in turn; thus the whole algorithm runs in time  $O(\log^2 N)$ , if  $N$  is the total number of vertices involved in  $P$  and  $R$ .

(3) IHH—Intersection of Two Polyhedra  $P$  and  $Q$ . By intersecting  $P$  and  $Q$  with a plane, a series of binary searches will reduce  $P$  successively to a drum, a “slice,” and a pentahedron. Each step of the binary searches involves  $O(\log^2 N)$  operations, thus leading to an  $O(\log^3 N)$ -time algorithm, with  $N$  the total number of vertices in  $P$  and  $Q$ .

**3.2 REPRESENTATION OF THREE-DIMENSIONAL OBJECTS.** All of our polyhedra are assumed to be in a standard representation as  $p$  (or fewer) polygonal cross-sections. These cross-sections are created by setting a planar direction  $K$  and intersecting the polyhedron with a plane in that direction passing through each of its vertices (see Figure 7). The naive representation of this structure would require  $O(p^2)$  preprocessing time and  $O(p^2)$  storage space in the worst case. In [7] a representation using  $O(p \log p)$  time and space is given. Accessing this data structure, however, adds a factor of  $O(\log p)$  to the running times of our algorithms. We do not consider the details of this algorithm here and assume that questions of the form:

What is the  $i$ th vertex of the  $j$ th cross-section?

may be answered in constant time. In a model where the accesses actually require  $O(f(p))$  operations, our upper bounds of  $O(g(p))$  are actually  $O(f(p)g(p))$ .

For the polyhedron  $P$ , we denote its cross-sections as  $P_1, P_2, \dots, P_p$  and let  $P_{i,j}$  represent the part of the polyhedron between  $P_i$  and  $P_j$  (inclusive). A key feature of this representation is that we make no assumption about the preprocessing direction. Therefore, the representation (in terms of  $P_1, P_2, \dots, P_p$ ) is invariant under scaling, rotation, or translation. Furthermore, when we consider the intersection of two preprocessed polyhedra, we need not assume that their polygonal cross-sections lie in parallel planes. Since it is almost the case that each vertex of  $P_i$  is adjacent to a unique vertex of  $P_{i+1}$ , we nearly have a one-to-one correspondence between  $P_i$  and  $P_{i+1}$ , and these two polygons almost fully describe the drum  $P_{i,i+1}$ . Unfortunately, the vertices of  $P_i$  that are also vertices of  $P$  may be adjacent

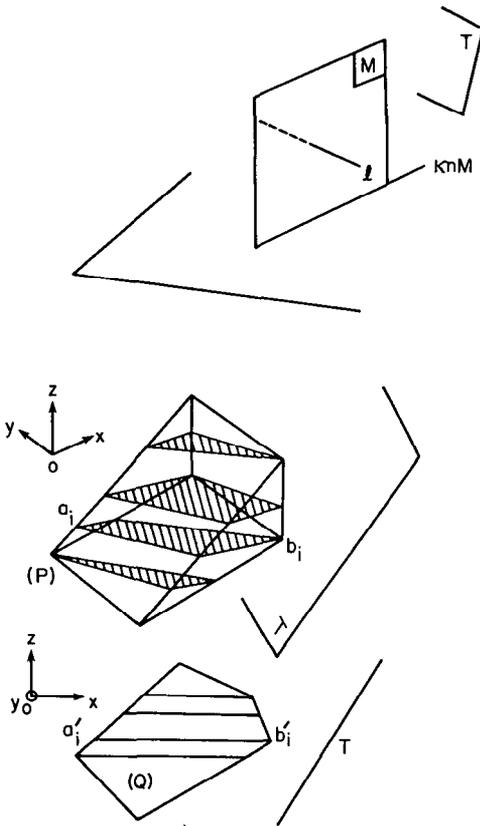


FIG. 8. The algorithm IHP.

to several vertices of  $P_{i+1}$ , and to remedy this discrepancy, we add dummy vertices and dummy edges of length 0. More precisely, let  $x_{i,j}$  be the  $j$ th vertex of  $P_i$  and let  $e_1, \dots, e_k$  be the lateral (i.e., joining  $P_i$  and  $P_{i+1}$ ) edges of  $P_{i+1}$  emanating from  $x_{i,j}$ , given in clockwise order around  $P_{i+1}$  (whichever order on  $P_{i+1}$  can be called clockwise, as long as this is done consistently with all the cross-sections). In general  $k = 1$ . If, however,  $k > 1$ , we conceptually duplicate  $x_{i,j}$  into  $k$  vertices  $y_1, \dots, y_k$ , all of which have the same geometric location as  $x_{i,j}$ . Each  $y_u$ , however, is made incident to exactly one edge  $e_u$ .

Iterating on this process for all vertices of  $P_i$  and all preprocessing polygons, we rename the vertices thus obtained for each  $P_{i,i+1}, x_{i,1}^+, x_{i,2}^+, \dots$  in clockwise order. Similarly, we consider all the lateral edges of  $P_{i,i-1}$  emanating from  $x_{i,j}$  and duplicate  $x_{i,j}$  accordingly. We thus define a refinement of  $P_i$  with respect to the drum  $P_{i-1,i}$ , renaming all the vertices of  $P_i, x_{i,1}^-, x_{i,2}^-, \dots$ . Note that there is a one-to-one correspondence between  $\{x_{i,1}^+, x_{i,2}^+, \dots\}$  and  $\{x_{i+1,1}^-, x_{i+1,2}^-, \dots\}$ , and all the preprocessing can be done in  $O(p^2)$  time.

**3.3 INTERSECTION OF A PLANE WITH A POLYHEDRON (IHP).** Let  $P$  be a convex polyhedron with  $p$  vertices  $p_1, p_2, \dots$  and let  $T$  denote the plane under consideration. Let  $K$  be a plane containing a (nondegenerate) preprocessing polygon. If  $K$  and  $T$  are parallel, then we can find which drum the plane  $T$  intersects by binary search and, in the affirmative, intersect  $T$  with any edge of the drum nonparallel to  $K$  and output an intersection point. So, let us assume in the following that the intersection  $K \cap T$  is a line  $l$ . Let  $M$  be a plane normal to  $l$ ;  $P$  and  $T$  intersect if and only if their projections on  $M$  intersect (see Figure 8).

Let  $a_i$  (respectively,  $b_i$ ) be the vertex of  $P_i$  with minimum (respectively, maximum) coordinate in the direction  $K \cap M$ . In general,  $a_i$  and  $b_i$  are unique, although there may be two of them if  $P_i$  has an edge parallel to the  $l$ -axis. In any case, the orthogonal projection of  $a_i$  (respectively,  $b_i$ ) on the  $M$ -plane, denoted  $a'_i$  (respectively,  $b'_i$ ) is unique. We first show that the polygon  $Q = a'_1 \cdots a'_p b'_p \cdots b'_1$  is convex (see Figure 8).

LEMMA 6. *The polygon  $Q$  is convex.*

PROOF. We show that none of the angles  $(b'_k b'_{k+1}, b'_k b'_{k-1})$  is reflex. Let  $B$  be the intersection of the segment  $b_{k-1} b_{k+1}$ , with the plane  $P_k^*$  supporting  $P_k$ . Since  $P$  is convex,  $B$  lies on  $P_k$ ; therefore its  $K \cap M$ -coordinate cannot be greater than the  $K \cap M$ -coordinate of  $b_k$ . The projection of  $B$  on  $M$  being also the intersection of  $b'_{k-1} b'_{k+1}$  with  $P_k^*$ , it follows that the angle  $(b'_k b'_{k+1}, b'_k b'_{k-1})$  is no greater than 180 degrees. We have the same result with the vertices  $a'_k$ , and it is easy to conclude that  $Q$  is convex.  $\square$

This leads to

LEMMA 7. *Let  $L$  be the intersection of  $T$  with  $M$ . Then  $P$  and  $T$  intersect if and only if  $Q$  and  $L$  intersect.*

PROOF. If  $P$  and  $T$  intersect, we distinguish between two cases:

- (1)  $T$  intersects some  $P_i$ . Then the intersection of  $P_i$  and  $T$  is a line segment parallel to  $l$ , and its projection on  $M$  is a point that lies on the segment  $a'_i b'_i$ . It follows that  $Q$  and  $L$  intersect.
- (2) If  $T$  does not intersect any  $P_i$ , it lies strictly between two consecutive preprocessing polygons  $P_i$  and  $P_{i+1}$ ; thus  $L$  intersects  $a'_i a'_{i+1}$ , that is, intersects  $Q$ .

Conversely, if  $L$  intersects  $Q$ , it must intersect one of its edges. Its endpoints are the projections on  $M$  of two vertices  $u$  and  $v$  on the boundary of  $P$ , and it is clear that  $T$  must intersect the segment  $uv$ , that is, intersect  $P$ . Note that  $u$  and  $v$  are not necessarily vertices of  $P$ .  $\square$

From the previous results, we can easily derive the algorithm IHP.

#### Algorithm IHP

If  $P$  and  $T$  do not intersect, the algorithm returns NO; otherwise, it returns (YES,  $A$ ), where  $A$  is a point of the intersection.

Lemma 7 shows that we can test the intersection of  $P$  with  $T$  by applying the IGL algorithm to  $Q$  and  $L$ . We have an implicit description of  $Q$ , since we have random access to any of its vertices in  $O(\log p)$  time. This is due to the fact that the  $M \cap K$ -coordinates of the vertices of any preprocessing polygon form a bimodal function since the polygon is convex. Therefore, any  $a_i$  or  $b_i$  can be obtained in  $O(\log p)$  time, from which  $a'_i$  and  $b'_i$  are computed in constant time. If  $Q$  and  $L$  do not intersect, IHP will return NO, else IGL provides, in  $O(\log p)$  time, an edge of  $Q$  intersecting  $L$ , say  $b'_i b'_{i+1}$ . Since knowing  $b'_i$  and  $b'_{i+1}$  implies that  $b_i$  and  $b_{i+1}$  have already been computed, we can immediately determine the intersection  $A$  of  $T$  with the segment  $b_i b_{i+1}$  and return (YES,  $A$ ). Note that in this case, the segment  $b_i b_{i+1}$  always intersects  $T$ . Since the algorithm IGL runs in logarithmic time and each basic step requires  $O(\log p)$  operations, we can conclude:

THEOREM 8. *The intersection of a plane with a preprocessed convex polyhedron of  $p$  vertices can be detected in  $O(\log^2 p)$  operations.*

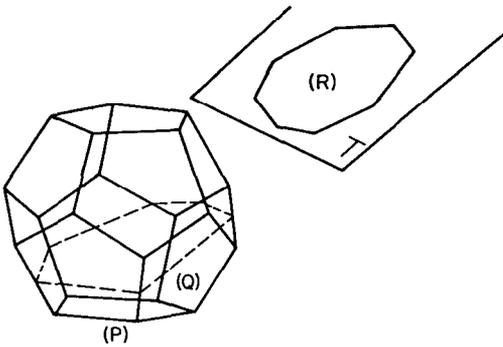


FIG. 9. Intersection of a polyhedron and a polygon.

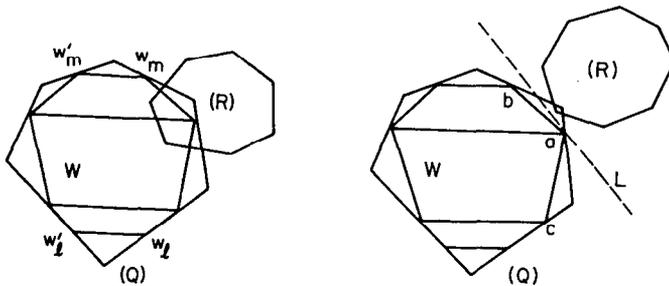


FIG. 10. The two cases of intersection of  $Q$  and  $R$ .

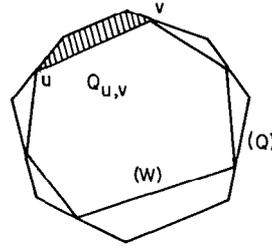
3.4 INTERSECTION OF A POLYGON WITH A POLYHEDRON (IHG). We start with an analysis of the problem, concentrating only on the most difficult points. Let  $P$  be a preprocessed convex polyhedron of  $p$  vertices and  $R$  a convex polygon of  $q$  vertices. Call  $Q$  the intersection of  $P$  with the plane  $T$  supporting  $R$  (see Figure 9). By first calling upon IHP, we can check whether  $Q$  is empty. Assume that this is not the case. It is equivalent to test the intersection of  $P$  and  $R$  or  $Q$  and  $R$ . Although  $Q$  is not readily available, the preprocessing of  $P$  permits us to compute an implicit description of it. We first observe that from the convexity of  $P$ ,  $T$  intersects a set (possibly empty) of consecutive  $P_i$ , say,  $P_l, \dots, P_m$  ( $l \leq m$ ). Let  $w_i, w'_i$  be the endpoints of the intersection of  $T$  and  $P_i$ , and  $W$  denote the polygon  $w'_1 \dots w'_m w_m \dots w_l$  (see Figure 10). Since  $W$  is a subpolygon of  $Q$  (i.e.,  $W$  lies inside  $Q$  and all its vertices lie on the boundary of  $Q$ ), it is easy to see that the convexity of  $Q$  implies the convexity of  $W$ . If  $u, v$  are two consecutive vertices of  $W$  in clockwise order, we define  $Q_{u,v}$  to be the convex polygon (outside of  $W$ ) delimited by the edge  $uv$  and the boundary of  $Q$  (see Figure 11).

The following result shows how to reduce our main problem to two easier subproblems.

LEMMA 9. *If  $Q$  and  $W$  are not empty,  $P$  and  $R$  intersect if and only if either of the following conditions is satisfied:*

- (1)  $W$  and  $R$  intersect.
- (2) Let  $L$  be a separating line of  $W$  and  $R$  passing through a vertex  $a$  of  $W$ , and let  $b, c$  be the vertices of  $W$  adjacent to  $a$  ( $b = c$  if  $W'$  is reduced to a line segment). Then  $R$  intersects  $Q_{b,a}$  or  $Q_{a,c}$  (see Figure 10).

PROOF. It suffices to observe that when  $R$  intersects  $Q$  but not  $W$ , the only parts of  $Q$  that  $L$  does not separate from  $R$  are  $Q_{b,a}$  and  $Q_{a,c}$ . The remainder of the proof is straightforward.  $\square$

FIG. 11. The polygons  $Q$ ,  $W$ ,  $Q_{u,v}$ .

Case 1 being easy to handle, let us turn to the other case. We wish to compute an implicit description of  $Q_{b,a}$  and  $Q_{a,c}$  in order to test these polygons for intersection with  $R$ . We describe the method for  $Q_{b,a}$ , the other case being similar. Call  $q_1, \dots, q_k$  the vertices of  $Q_{b,a}$ , that is, the vertices of  $Q$  lying between  $b$  and  $a$ . Note that  $q_1, \dots, q_k$  are the intersections of the plane  $T$  with consecutive lateral edges of some drum  $P_{i,i+1}$ , say,  $e_1, \dots, e_k$ . Since all the edges  $e_1, \dots, e_k$  must pass through consecutive vertices of  $P_i$ ,  $x_1, \dots, x_k$ , it suffices to determine  $x_1$  and  $x_k$  to have an implicit description of  $Q_{b,a}$ . In order to have a one-to-one correspondence between the  $x_i$  and the  $e_i$ , we must consider  $P_i$  with its vertices of the form  $x_{i,1}^+, x_{i,2}^+, \dots$ . We distinguish between two cases:

(1) The segment  $ab$  is parallel to the preprocessing polygons (horizontal); it is then the top or bottom edge of  $W$ , say, the top edge (without loss of generality). Consider the three-dimensional strip  $S$  of  $P_{i,i+1}$  formed by all its lateral faces. The intersection of  $T$  with this strip is a continuous broken line  $D$  running from  $P_i$  to  $P_{i+1}$  without intersecting  $P_{i+1}$  (see Figure 12a). Therefore any path from the portion of the boundary of  $P_i$  between  $a$  and  $b$  to  $P_{i+1}$  must intersect  $D$ . It follows that  $x_1, \dots, x_k$  are exactly all the vertices of  $P_i$  between  $a$  and  $b$ . To decide whether it is between  $a$  and  $b$  or  $b$  and  $a$  in clockwise order around  $P_i$ , we simply observe that on one part of the boundary all the lateral edges intersect  $T$ , whereas none does on the other. Thus, testing any lateral edge for intersection with  $T$  will resolve the ambiguity in constant time.

(2) The segment  $ab$  is not a horizontal edge of  $W$ . Then  $P_{i,i+1}$  now designates the drum lying between  $a$  and  $b$ . The intersection of  $T$  with the strip  $S$  consists of two broken lines, one of which runs from  $a$  to  $b$  (see Figure 12b). Let  $x_u, x_{u+1}$  (respectively,  $y_v, y_{v+1}$ ) be the edge of  $P_i$  (respectively,  $P_{i+1}$ ), given in clockwise order, which contains  $a$  (respectively,  $b$ ). Note that these edges will have already been computed when  $a$  and  $b$  are obtained. Since we wish to access the edges of  $P_{i,i+1}$  from the vertices of  $P_i$ , it is important to have a one-to-one correspondence between the vertices of  $P_i$  and  $P_{i+1}$ ; therefore, we consider the polygon  $P_i$  (respectively,  $P_{i+1}$ ) with its vertices  $x_{i,1}^+, x_{i,2}^+, \dots$  (respectively,  $x_{i+1,1}^-, x_{i+1,2}^-, \dots$ ). Let  $x_l$  be the vertex of  $P_i$  in correspondence with  $y_v$ , that is, the vertex lying with  $y_v$  on the same lateral edge of  $P_{i,i+1}$ . It is clear that if the lateral edge of  $P_{i,i+1}$  passing through  $x_u$  intersects  $T$ , then  $q_1, \dots, q_k$  are exactly the intersections of  $T$  with the lateral edges emanating from  $x_{l+1}, x_{l+2}, \dots, x_u$  (see Figure 12b). Otherwise, if the lateral edge emanating from  $x_{u+1}$  intersects  $T$ , the vertices  $q_1, \dots, q_k$  of  $Q$  are determined by the set of vertices  $x_{u+1}, \dots, x_l$  (see Figure 12c). Finally, if neither of the above cases arises, no lateral edge intersects  $T$  between  $a$  and  $b$ , and  $Q_{b,a}$  is reduced to the single edge  $ab$ ; therefore no testing is necessary.

Putting all these results together and handling the remaining cases is straightforward. We can now set out the algorithm IHG, whose correctness is established by these results.

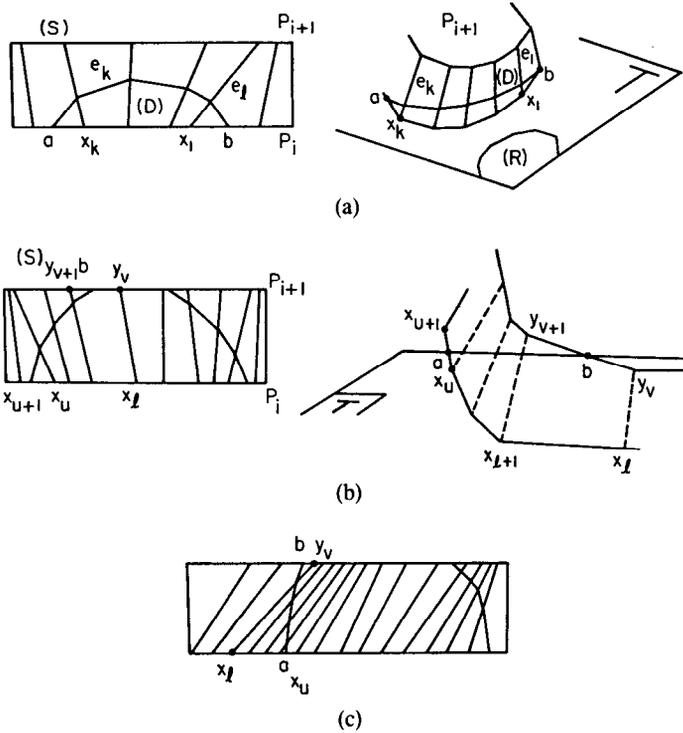


FIG. 12. Computing  $Q_{b,a}$ .

**Algorithm IHG**

The algorithm takes a convex polygon  $R$  and a preprocessed convex polyhedron  $P$  as input and returns NO if  $P$  and  $R$  do not intersect or (YES,  $A$ ) if they do, with  $A$  a point of the intersection.

*Step 1.* Test the intersection of  $P$  with the plane  $T$  supporting  $R$  by calling upon IHP. If  $P$  and  $T$  do not intersect, return NO; else the algorithm IHP will provide a point  $I$  of the intersection, as well as the preprocessing plane  $P_i^*$ , such that  $I$  lies in the drum  $P_{i,i+1}$ . If IGL indicates that  $T$  intersects neither  $P_i$  nor  $P_{i+1}$ , go to Step 2; else go to Step 3.

*Step 2.* “ $T$  lies strictly between  $P_i$  and  $P_{i+1}$ .”  $Q$  being the intersection of  $T$  and  $P$ , the vertices of  $Q$  are exactly the intersections of  $T$  with all the lateral edges of  $P_{i,i+1}$ . Therefore  $P_i$  gives an implicit description of  $Q$ , and it is possible to test the intersection of  $Q$  and  $R$  with the IGG algorithm, returning NO if it is empty or (YES,  $A$ ) if it is not, where  $A$  is a point of the intersection returned by IGG.

*Step 3.* “ $T$  intersects  $P_i$  or  $P_{i+1}$ .” Without loss of generality, assume that  $T$  intersects  $P_i$ . Since  $T$  intersects a set of consecutive preprocessing polygons  $P_l, \dots, P_m$ , we can determine  $P_l$  and  $P_m$  through a binary search by testing the intersection of  $P_k$  and  $T$  with the IGL algorithm. This gives an implicit description of  $W$ , from which we can test the intersection of  $R$  and  $W$  with IGG. Note that to access a vertex of  $W$ , we must compute the intersection of  $T$  with some preprocessing polygon, using the IGL algorithm. If the intersection of  $R$  and  $W$  is not empty, IGG will provide a common point  $A$ , and we can return (YES,  $A$ ). Otherwise, IGG will return a separating line  $L$  of  $W$  and  $R$  passing through  $W$ , thus providing the vertices  $a, b, c$ .

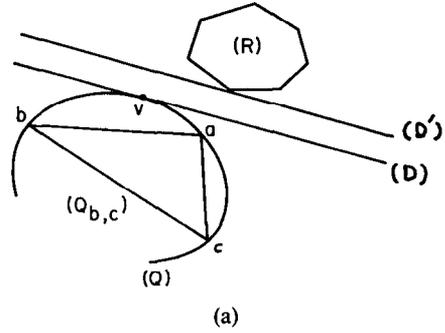
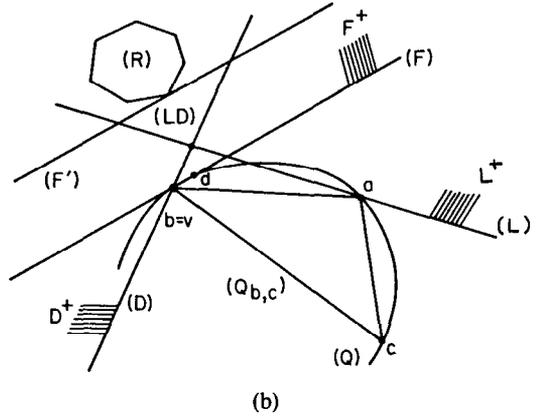


FIG. 13. Computing a pair of separating lines for  $Q$  and  $R$ .



*Step 4.* “If  $R$  intersects  $P$ , it intersects  $Q_{b,a}$  or  $Q_{a,c}$ .” Apply the procedure described above for  $Q_{b,a}$  and  $Q_{a,c}$  successively, and test these polygons for intersection of  $R$  (IGG), returning NO or a common point accordingly.

Before analyzing the running time of IHG, we wish to extend the algorithm slightly so that it returns a pair of parallel separating lines when  $P$  and  $R$  do not intersect, that is, a pair of separating lines for  $Q$  and  $R$ . When IHG returns NO in step 1, no such pair can be defined, but the plane  $T$  is itself a separating hyperplane and is sufficient information for our purposes. In all of the other cases, a nonintersection of  $P$  and  $R$  is detected after testing both  $Q_{b,a}$  and  $Q_{a,c}$  for intersection has failed. Instead of testing these two polygons successively, we can simply use the implicit description of  $Q_{b,a}$  and  $Q_{a,c}$  to test the intersection of  $Q_{b,c}$  with  $R$  ( $Q_{b,c}$  is defined as the union of  $Q_{b,a}$ ,  $Q_{a,c}$ , and the triangle  $abc$ ). If no intersection is found, the algorithm IGG will return a pair of separating lines  $(D, D')$  for  $Q_{b,c}$  and  $R$ . Let  $v$  be the vertex of  $Q_{b,c}$  lying on the separating line  $D$ .

If  $v$  is distinct from  $b$  and  $c$ ,  $(D, D')$  is also a pair of separating lines for  $Q$  and  $R$  since  $Q$  is convex, and fits our purposes (see Figure 13a).

If  $v$  is  $b$  or  $c$  (say  $b$ , without loss of generality),  $D$  may intersect  $Q$  outside of  $v$ , thus not separate  $Q$  and  $R$ . In that case, let  $d$  be the vertex of  $Q_{b,c}$  adjacent to  $b$  and distinct from  $c$ . We can show that the line  $F$  passing through  $bd$  separates  $Q$  from  $R$ . Then computing a line  $F'$  adjacent to  $R$  and parallel to  $F$  so that  $(F, F')$  forms a pair of separating lines will take only  $O(\log q)$  time, as described earlier. We now prove our claim.

Recall that the algorithm has already computed a line  $L$  that is adjacent to the vertex  $a$  of  $Q$  and that separates  $W$  and  $R$ . Call  $L^+$ ,  $D^+$ ,  $F^+$  the halfspaces delimited by  $L$ ,  $D$ ,  $F$ , respectively, that do not contain the vertex  $c$  (see Figure 13b). Since

both  $L$  and  $D$  separate  $R$  from the triangle  $abc$ ,  $R$  lies in the intersection of  $L^+$  and  $D^+$ , denoted  $LD$ . Since  $Q_{b,c}$  does not intersect the interior of  $D^+$ , the line  $F$  cannot intersect  $LD$ ; therefore  $R$  is completely inside  $F^+$ . This implies that  $F$  is a separating line of  $R$  and  $Q$ , which proves our claim.

Step 1 calls upon IHP and IGL and thus requires  $O(\log^2 p)$  operations. Step 2 is a simple application of IGG and takes  $O(\log(p + q))$  time. Step 3 involves a binary search on the preprocessing polygons with a call on IGL at each step, which amounts to  $O(\log^2 p)$  time. Testing the intersection of  $W$  and  $R$  takes  $O((\log p)\log(p + q))$  time, since each vertex of  $W$  is obtained by intersecting  $T$  with some  $P_k$  (IGL), which takes  $O(\log p)$  time. Finally, step 4 performs a constant-time case analysis and then call on IGG, which requires  $O(\log(p + q))$  operations. We can finally state our main result.

**THEOREM 10.** *The intersection of a preprocessed convex polyhedron of  $p$  vertices with a convex polygon of  $q$  vertices can be detected in  $O((\log p)\log(p + q))$  operations, that is, in  $O(\log^2 N)$  time, where  $N$  is the total number of vertices in both objects.*

**3.5 INTERSECTION OF A LINE WITH A POLYHEDRON (IHL).** We now consider the problem of detecting an intersection between an infinite line (or a line segment)  $L$  and a convex polyhedron of  $p$  vertices preprocessed as usual. We can contemplate a solution that is a straightforward application of the method described in the previous section.

We first test the intersection of  $P$  with any plane  $T$  supporting the line  $L$ , using IHP. If we fail to detect an intersection, we obviously return NO. Otherwise, we define the polygon  $Q$  as usual (i.e., the intersection of  $P$  and  $T$ ), and we compute an implicit description of its subpolygon  $W$  formed by the preprocessing polygons of  $P$ . Next, we test the intersection of  $W$  and  $L$  (IGL), and in the event of a failure compute a separating line adjacent to  $W$  and derive the polygons  $Q_{b,a}$  and  $Q_{a,c}$ . Finally, we test these polygons for intersection with  $L$ , calling upon IGL.

Note that in the case of an intersection, we can compute the segment  $S$  of  $L$  that lies in  $P$  in  $O(\log p)$  time. There are essentially two cases to consider:

- (1) If an intersection is detected while intersecting  $Q_{b,a}$  (respectively,  $Q_{a,c}$ ) with  $L$ , then  $S$  is exactly the intersection of  $Q_{b,a}$  (respectively,  $Q_{a,c}$ ) with  $L$ , and we can compute it in  $O(\log p)$  time (IGL) (see Figure 14a).
- (2) If  $W$  and  $L$  intersect, then IGL will provide the two edges of  $W$  that intersect  $L$ , say,  $ab$  and  $a'b'$  (see Figure 14b). It is clear that, if  $A$  (respectively  $B$ ) is the point on the boundary of  $Q_{a,b}$  (respectively,  $Q_{a'b'}$ ) that intersects  $L$  and does not lie on  $ab$  (respectively,  $a'b'$ ), then  $S$  is the segment  $AB$ . To obtain this segment, we need to compute implicit descriptions of  $Q_{a,b}$  and  $Q_{a'b'}$  and intersect  $L$  with these two polygons (see Algorithm IHG for details of the procedure). Finally, IGL will provide  $A$  and  $B$  in  $O(\log p)$  time.

The total running time of the algorithm is clearly  $O(\log^2 p)$ , and we conclude:

**THEOREM 11.** *We can compute (explicitly) the intersection of a preprocessed polyhedron of  $p$  vertices with an infinite line or a line segment in  $O(\log^2 p)$  operations.*

**3.6 INTERSECTION OF TWO POLYHEDRA (IHH).** We now turn to the problem of detecting the intersection of two convex polyhedra  $P$  and  $Q$  of  $p$  and  $q$  vertices, respectively. We assume that both polyhedra have been preprocessed, yet we do not require that the preprocessing planes of  $P$  should be parallel to those of  $Q$  (see Figure 15a). Thus we can maintain a coordinate-free environment. If either  $P$  or

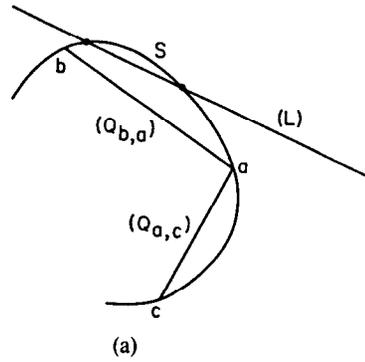
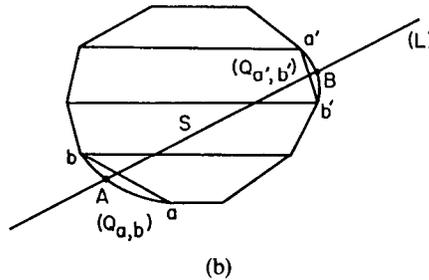


FIG. 14. The algorithm IHL.



$Q$  is rotated, no new preprocessing will be necessary. The algorithm IHH proceeds by a series of binary searches, all very similar in nature, and reduces  $P$  to a drum, a “slice,” and a pentahedron successively. For the clarity of exposition, we start our analysis of the problem with some preliminary results related to lines and planes of support. We redefine a line of support of  $P$  more precisely as a line having exactly one point or one segment (not necessarily an edge) in common with the boundary of  $P$ . Similarly, a plane of support of  $P$  is defined as a plane with exactly one edge or one face in common with the boundary of  $P$ .

For later purposes, we need to extend the preprocessing of  $P$  slightly. We require the existence, for each vertex of  $P$ , of an array listing the edges incident to it in clockwise order. This additional information is readily obtained once the polyhedron has been preprocessed. We begin with a preliminary result:

**LEMMA 12.** *If  $L$  is a line of support of  $P$  and one edge of  $P$  that intersects  $L$  is known, then it is possible to determine a plane of support of  $P$  containing  $L$  in  $O(\log p)$  operations.*

**PROOF.** Call  $v$  the intersection of  $L$  with that edge  $e$  of  $P$  known to intersect  $L$ . We distinguish between two cases:

- (1) If  $v$  is not a vertex of  $P$  (check whether  $v$  is an endpoint of  $e$ ), then the plane containing both  $e$  and  $L$  is a plane of support of  $P$  (see Figure 16a).
- (2) If  $v$  is a vertex of  $P$ , then the plane passing through  $e$  and  $L$  may unfortunately intersect the interior of  $P$ , and further analysis is needed (see Figure 16b). Let  $e_1, \dots, e_k$  be a list of the edges of  $P$  adjacent to  $v$ , in clockwise order. Recall that the preprocessing of  $P$  ensures random access to these edges. Let  $U_i$  be the plane containing both  $L$  and  $e_i$ . The planes  $U_i$  ( $i = 1, \dots, k$ ) such that  $L$  and  $e_i$  are not collinear form a bimodal angular sequence. The extrema are valid planes of support and can be computed in  $O(\log p)$  time.  $\square$

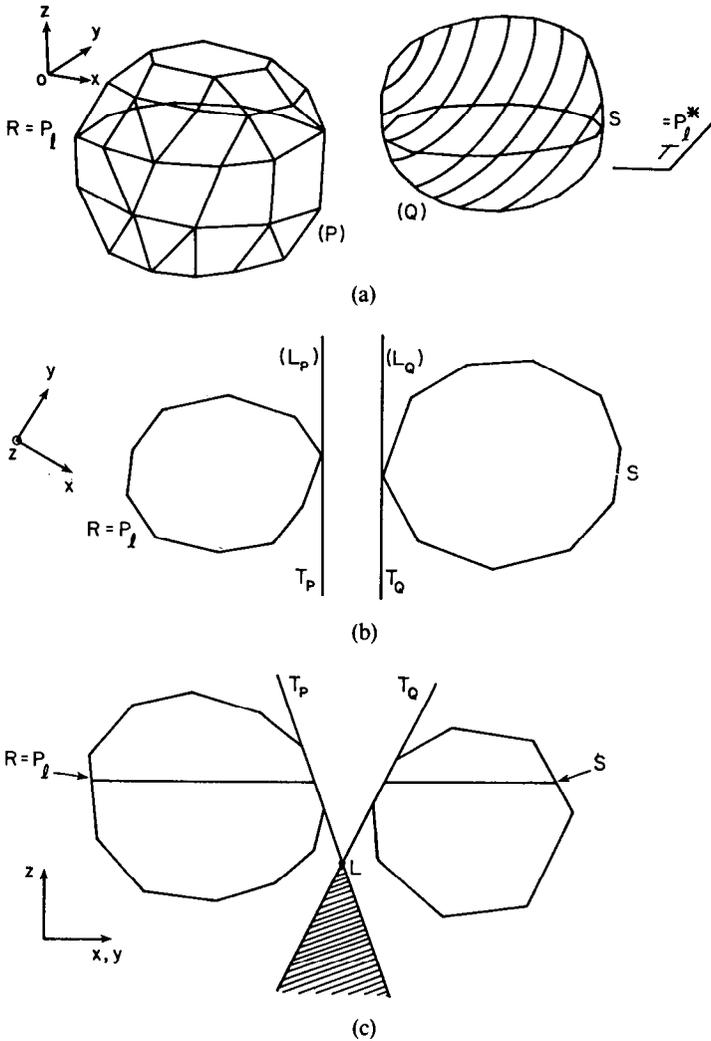
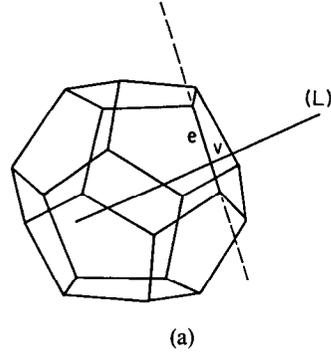
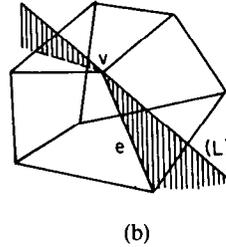


FIG. 15. Reducing the size of  $P$  in IHH.

We now turn to the crux of the algorithm IHH. Let us assume that  $P$  and  $Q$  intersect but neither contains the other. Let  $T$  be a plane intersecting  $P$  and  $Q$  but not their intersection. Call  $R$  (respectively,  $S$ ) the intersection of  $T$  and  $P$  (respectively,  $Q$ ), and let  $(L_P, L_Q)$  be a pair of parallel separating lines for  $R$  and  $S$ , respectively. If  $T_P$  (respectively,  $T_Q$ ) is a plane of support of  $P$  (respectively,  $Q$ ) passing through  $L_P$  (respectively,  $L_Q$ ), observing the relative position of  $T_P$  and  $T_Q$  will indicate on which side of  $T$  the intersection of  $P$  and  $Q$  lies. Indeed, since  $P$  and  $Q$  intersect but  $R$  and  $S$  do not, the intersection of  $P$  and  $Q$  lies entirely in one of the halfspaces delimited by  $T$  (see Figure 15). To determine which, we first observe that the intersection of  $P$  and  $Q$  must lie in the intersection  $H$  of the halfspace delimited by  $T_P$ , which contains  $P$  with the halfspace delimited by  $T_Q$  containing  $Q$ . Since  $L_P$  and  $L_Q$  are parallel,  $H$  lies totally on one side of  $T$ , and the intersection  $L$  of  $T_P$  and  $T_Q$  (which must exist since  $H$  is nonempty) may be computed in constant time and indicates which side of  $T$  contains the intersection of  $P$  and  $Q$ . Note that  $L$  is an infinite line parallel to  $T$  (see Figure 15b, c). The portion of  $P$  that does not lie on the same side of  $T$  as  $L$  can be rejected since it

FIG. 16. Computing a plane of support of  $P$ .

cannot intersect with  $Q$ . This gives us a means of reducing the size of the problem, so carrying out this process in a binary search fashion will guarantee efficiency. We now proceed to describe the algorithm.

(1) Let  $P_l$  be the middle preprocessing polygon of  $P$  ( $l = \lceil p/2 \rceil$ ). The first step consists of reducing  $P$  to  $P_{1,l}$  or  $P_{l,p}$ . To do so, we test the intersection of  $Q$  with the preprocessing plane  $P_l^*$  passing through  $P_l$ , using the IHP algorithm. If it fails to detect an intersection,  $Q$  lies entirely on one side of  $P_l^*$ , which can be determined in constant time. We then iterate on this process with  $P_{1,l}$  or  $P_{l,p}$ , whichever lies on the same side of  $P_l^*$  as  $Q$ . If  $P_l^*$  and  $Q$  intersect, we call upon IHG to test the intersection of  $Q$  with the polygon  $P_l$ , returning (YES,  $A$ ) if IHG finds a point  $A$  of the intersection or providing a pair of separating lines ( $L_P, L_Q$ ) (see Figure 15b). Since in this last case, IHG will also indicate edges of  $P$  (respectively,  $Q$ ) that intersect  $L_P$  (respectively,  $L_Q$ ), we can apply the result of Lemma 12 and compute a plane of support of  $P$  passing through  $L_P$ , which we denote  $T_P$ . A similar computation will give a plane of support of  $Q$  passing through  $L_Q$ ,  $T_Q$  (see Figure 15c). Finally, our discussion above shows how locating the intersection of  $L_P$  and  $L_Q$  with respect to  $P_l^*$  permits us to substitute  $P_{1,l}$  or  $P_{l,p}$  for  $P$  accordingly. Of course, if  $T_P$  and  $T_Q$  do not intersect (i.e., are parallel), neither do  $P$  and  $Q$ , so we can terminate.

Iterating on this process will either produce a point of the intersection or reduce  $P$  to a convex drum  $P_{i,i+1}$ . Note that we may have  $i + 1 = 1$  or  $i = p$ , in which case the algorithm can return NO since  $P$  and  $Q$  do not then intersect.

(2) It now remains to test the intersection of  $Q$  and  $P_{i,i+1}$ . Let  $x_1, \dots, x_k$  be the vertices of  $P_i$  in clockwise order. We choose a lateral edge  $e$  of  $P_{i,i+1}$ , say, an edge passing through  $x_1$ , and consider the plane  $T_j$  containing both  $x_j$  and the edge  $e$ . For any  $u, v$ ,  $1 < u < v \leq k$ , we define  $T_{u,v}$  as the portion of  $P_{i,i+1}$  comprised between  $T_u$  and  $T_v$  (i.e., the portion that contains the edge  $x_u x_{u+1}$ ). We have seen

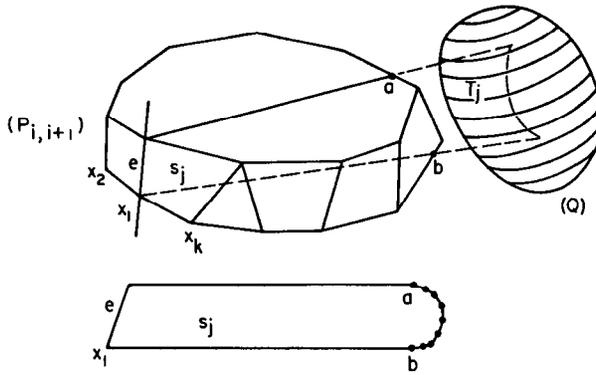


FIG. 17. Reducing the cap  $P_{i,i+1}$  in IHH.

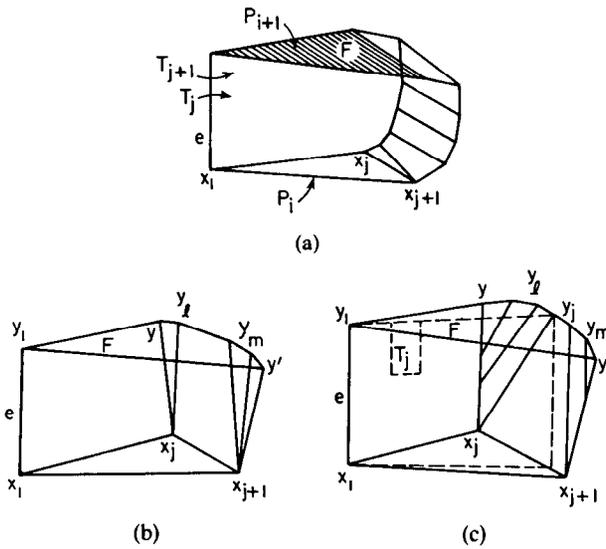


FIG. 18. The cap  $P_{i,i+1}$  after reduction.

in the description of the IHG algorithm how to compute an implicit description of the polygon  $S_j$  formed by the intersection of  $P_{i,i+1}$  and  $T_j$  (see Figure 17). Recall that this involves computing the points  $a$  and  $b$ , as well as the lateral edges of  $P_{i,i+1}$  that intersect  $T_j$ . Having an implicit description of  $S_j$ , we can apply the procedure described earlier, first using IHP with arguments  $T_j, Q$  and then IHG with arguments  $S_j, Q$ . This will either return a point of the intersection of  $S_j$  and  $Q$ , in which case we are done, or produce a pair of planes of support for  $P$  and  $Q$ , respectively, containing two parallel lines separating  $S_j$  and  $Q$ .

Once again, locating the intersection of these two planes will permit us to substitute  $T_{2,j}$  or  $T_{j,k}$  for  $P$  accordingly. We can perform a binary search on  $j$  in the interval  $[2, k]$ . If the algorithm does not terminate before, it will reduce  $P_{i,i+1}$  to the convex polyhedron  $T_{j,j+1}$  for some  $j$  (see Figure 18a).

(3)  $T_{j,j+1}$  has one face lying on  $P_i$  (the triangle  $x_1x_jx_{j+1}$ ) and a parallel face on  $P_{i+1}$ , denoted  $F$ . Unfortunately, Figure 18a illustrates only the simplest case since  $F$  is not necessarily a triangle. However, we can remedy this discrepancy easily.



a pair of planes of support from which we can decide which side of  $U_j$  contains the intersection of  $P$  and  $Q$ , if it exists. Note that the intersection of  $B$  and  $U_j$  is simply a triangle that we can compute in constant time. If the algorithm does not return, it will eventually reduce  $P$  to a pentahedron  $K$  lying between  $T_j$ ,  $T_{j+1}$ , two consecutive triangles  $U_l$ ,  $U_{l+1}$ , and a lateral face of  $P_{i,i+1}$ . In fact,  $k$  can be “degenerate” and have fewer than five faces.

(5) Finally, we have to test the intersection of  $K$  and  $Q$ . To do so, we can test each face of  $K$  successively, using the IHG algorithm. If we fail to detect an intersection, we determine whether  $Q$  lies entirely inside or outside of  $K$  by testing the inclusion of any point of  $Q$  in  $K$ , which can be done in constant time.

We now give a more formal outline of Algorithm IHH, which will also serve as a summary.

### Algorithm IHH

The input consists of two preprocessed convex polyhedra  $P$  and  $Q$ , and the output is NO if  $P$  and  $Q$  do not intersect or (YES,  $A$ ) if they do, where  $A$  is a point of the intersection.

#### Step 1

```

 $l = 1; m = p$ 
while  $l < m - 1$ 
  begin
     $i = \lfloor (l + m)/2 \rfloor$ 
    if  $P_i^*$  does not intersect  $Q$  [IHP]
      then
        if  $q_1$  lies above  $P_i^*$ 
          then  $l = i$ 
          else  $m = i$ 
        else
          if  $P_i$  intersects  $Q$  [IHG]
            then return (YES,  $A =$  point returned by IHG)
          “IHG provides a pair of separating lines from which  $T_P$  and  $T_Q$  are computed”
          if  $T_P$  and  $T_Q$  do not intersect
            then return (NO)
          if  $T_P$  and  $T_Q$  intersect above  $P_i^*$ 
            then  $l = i$ 
            else  $m = i$ 
  end
 $i = l$ 

```

*Step 2.* “ $P$  is reduced to a convex drum  $P_{i,i+1}$ .” Let  $e$  be a lateral edge of  $P_{i,i+1}$  and  $T_j$  the plane containing  $e$  and the vertex  $x_j$  of  $P_i$ . Apply step 1 with respect to the planes  $T_j$ . Finally set  $j$  to  $l$ .

*Step 3.* “ $P$  is reduced to a convex polyhedron  $T_{j,j+1}$ .” If the face of  $T_{j,j+1}$  lying on  $P_{i+1}^*$  is not a triangle, apply step 2 with respect to the planes  $T'_1, \dots, T'_m$  (defined in the previous discussion).

*Step 4.* “ $P$  is reduced to a polyhedron  $B$  bordered by two triangles, subpolygons of  $P_i$  and  $P_{i+1}$ .” Apply the procedure of step 1 with respect to the planes  $U_j$  passing through  $x_1$  and  $e_j$ , where  $x_1$  is a vertex of  $P_i$  and  $e_j$  is the  $j$ th lateral edge of  $B$ .

*Step 5.* “ $P$  is reduced to a polyhedron  $K$  with at most five faces.” Check whether  $q_1$  lies inside  $K$ . If affirmative, return (YES,  $q_1$ ). Otherwise, apply the IHG

algorithm to test whether  $Q$  intersects any of the faces of  $K$ . If this is the case, return (YES,  $A$ ), where  $A$  is a point of the intersection; else return (NO).

We can now state our main result

**THEOREM 13.** *The intersection of two preprocessed convex polyhedra of  $p$  and  $q$  vertices, respectively, can be detected in  $O((\log p)(\log q)\log(p + q))$  operations, that is,  $O(\log^3 N)$  time, where  $N$  is the total number of vertices in  $P$  and  $Q$ .*

**PROOF.** At this stage, we simply have to evaluate the execution time of the algorithm. We review its various phases and derive its complexity:

- (1) This stage involves  $O(\log p)$  applications of IHP ( $\log^2 q$ ), IHG ( $(\log q)\log(p + q)$ ), and the algorithm of Lemma 12 ( $\log pq$ ).
- (2) We can obtain an implicit description of  $S_j$  in constant time, once the intersection of  $T_j$  with  $P_i$  and  $P_{i+1}$  has been computed ( $\log p$ ). The remainder of this step is similar to the previous one.
- (3) This stage has the same complexity as (2), since computing an implicit description of  $y_1, \dots, y_m$  takes constant time.
- (4) This stage is similar to (2).
- (5) This is essentially a repeated application of IHG to  $Q$  and a triangle or a quadrilateral ( $\log^2 q$ ).  $\square$

#### 4. Conclusions

We have described a complete set of algorithms for detecting intersections in two and three dimensions. In all cases, we have avoided issues of efficiency beyond the asymptotic level. Although the algorithm for computing planar intersections is asymptotically optimal [16], we believe that a more sophisticated treatment of bimodal functions may improve its running time. Also, a more refined case analysis might permit us to reduce not only one of the polygons by half, but always both of them.

In three dimensions, aside from speeding up the preprocessing [7], we believe that algorithm IHH would benefit from a more symmetric treatment of the two polyhedra (along the lines of the algorithm IGG, for example). There also remains the question of proving lower bounds, since none of these algorithms has been shown to be optimal.

In all cases, we believe that improvements can be best discovered by implementing the algorithms and observing their behavior on real problems. There is also the possibility of using the methods presented here as the basis of fast probabilistic algorithms [14] or algorithms efficient on the average [3].

*Note:* While this paper was being refereed, some of the results were improved. In particular, terms of  $O(\log^3 n)$  in Section 3.6 have been reduced to  $O(\log^2 n)$  and terms of  $O(\log^2 n)$  in Sections 3.3 and 3.5 have been reduced to  $O(\log n)$  in [5].

**ACKNOWLEDGMENT.** We wish to thank Garret Swart for pointing out an omission in algorithm IGG in an earlier draft.

#### REFERENCES

1. AHO, A. V., HOPCROFT, J. E., AND ULLMAN, J. D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
2. BENTLEY, J. L., AND OTTMANN, T. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.* C-28 (Sept. 1979), 643–647.

3. BENTLEY, J. L., AND SHAMOS, M. I. Divide and conquer for linear expected time. *Inf. Proc. Lett.* 7 (1978), 87-91.
4. BENTLEY, J. L., HAKEN, D., AND HON, R. Fast geometric algorithms for VLSI tasks. In *Proceedings of the Computational Conference*, 1981, pp. 88-92.
5. DOBKIN, D. P., AND KIRKPATRICK, D. G. Fast detection of polyhedral intersection. *Theor. Comput. Sci.* 27 (1983), 241-253.
6. DOBKIN, D. P., AND LIPTON, R. L. Multidimensional searching problems. *SIAM J. Comput.* 5, 2 (June 1976), 181-186.
7. DOBKIN, D. P., AND MUNRO, J. I. Efficient uses of the past. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science* (Syracuse, N.Y., Oct.). IEEE, New York, 1980, pp. 200-206.
8. DOBKIN, D. P., AND TOMLIN, D. Cartographic modelling techniques in environmental planning: An efficient system design. Unpublished manuscript.
9. KIEFER, J. Sequential minimax search for a maximum. *Proc. Am. Math. Soc.* 4 (1953), 502-506.
10. KNUTH, D. E. *The Art of Computer Programming: Fundamental Algorithms*. Addison-Wesley, Reading, Mass., 1968.
11. MULLER, D. E., AND PREPARATA, F. P. Finding the intersection of two convex polyhedra. *Theor. Comput. Sci.* 7 (1978), 217-236.
12. NEWMAN, W., AND SPROULL, R. *Principles of Interactive Computer Graphics*. 2nd Ed. McGraw-Hill, New York, 1979.
13. NIEVERGELT, J., AND PREPARATA, F. P. Plane-sweeping algorithms for intersecting geometric figures. *Commun. ACM* 25, 10 (1982), 739-747.
14. RABIN, M. Probabilistic algorithms. In *Algorithms and Complexity: New Directions and Recent Results*, J. Traub, Ed. Academic Press, Orlando, Fla., 1976.
15. SHAMOS, M. I. Geometric complexity. In *Proceedings of the 7th Annual ACM Symposium on Theory of Computing* (Albuquerque, N.M., May). ACM, New York, 1975, pp. 224-233.
16. SHAMOS, M. I. Computational geometry. Ph.D. dissertation, Yale Univ., New Haven, Conn., May, 1978.
17. SHAMOS, M. I., AND HOEY, D. Geometric intersection problems. In *Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science* (Houston, Tex., Oct.). IEEE, New York, 1976, pp. 208-215.

RECEIVED OCTOBER 1983; REVISED JANUARY 1986; ACCEPTED JUNE 1986