INTERSECTION OF CONVEX OBJECTS IN TWO AND THREE DIMENSIONS

B. Chazelle

D. P. Dobkin

CS-TR-025-86

INTERSECTION OF CONVEX OBJECTS IN TWO AND THREE DIMENSIONS

by

B. Chazelle[1)] and D.P. Dobkin[2)]

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 where the objects are given as input and their intersection is returned as output. For many applications, however, we may assume 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 we propose algorithms requiring sublinear time for their solution in two and three dimensions.

This research was supported in part by the National Science Foundation under Grants MCS79-03428, MCS81-14207, MCS83-03925, and MCS-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 MCS78-01689. Portions of this research appeared in the first author's PhD thesis [5] and were presented at the 1980 STOC conference [6].

1) Department of Computer Science, Brown University, Providence, RI 02912.

2) Department of Computer Science, Princeton University, New Jersey 08544.

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

General Terms: Algorithms, Computational Geometry

Additional Key Words and Phrases: Convex Sets, Intersection, Fibonacci Search.

Table of Contents

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 quasilinear upper bounds [2,3,14,18,19,20]. Lower bounds for this problem use arguments claiming that linear time is required to read all inputs or report the output. For the problem which we pose, such arguments do not apply. We only require a witness to the intersection or non-intersection 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.

The table in Figure 1 summarizes our results, all 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 which 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 factor $\log n$ [9]. 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 the table are asymptotic, the constants involved are sufficiently small to make the algorithms viable in practice. Furthermore many of the applications to 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 [11]. For example, in computer graphics when we wish to clip or window a scene [15], algorithms of the form given here would be sufficient for identifying those polygons which would require further processing. Also, many applications to which such algorithms might be used in Design Rule Checking for VLSI [2], computer geography [10,21] computer aided design and computer animation require a gross procedure which 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 $p_1, \ldots, p_p$ and polygon Q vertices $q_1, \ldots, q_q$. We will assume that no three vertices of a polygon are collinear. All indices of P (resp. Q) are taken modulo p (resp. q) in the obvious fashion. A line will be specified by any two of its points and a segment by its two endpoints. AB will always refer to the segment from A to B, and "line(AB)" will represent the infinite line containing AB. We define $d(x, L)$ as the orthogonal distance from the point x 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 will represent the ith 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 $1, 2, \ldots, n$ is said to be *unimodal* if there exists an integer m $(1 \le m \le n)$ such that f is strictly increasing (resp. decreasing) on $[1, m]$ and decreasing (resp. increasing) on $[m+1, n]$, with $f(m) \ge f(m+1)$ (resp. $f(m) \le f(m+1)$). Kiefer [12] showed that Fibonacci search was an optimal method of finding m, the turning point of a unimodal function, requiring $1.44.. \log n$ probes. We extend his algorithm to find the turning point 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 $f(r), f(r+1), \ldots, f(n), f(1), \ldots, f(r-1)$ is unimodal. Our interest in bimodal functions stems from the following:

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

Proof: Let p_k be the vertex of P which minimizes $f(i)$ for $i = 1, \ldots, p$. In case of a tie, we choose k so that the only other integer which achieves the same value of f is $k-1$. We can do this because P is convex. We will show that the sequence $f(k), f(k+1), \ldots, f(k-1)$ 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 $i = 1, \ldots, p$ 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 $\sin(a_k), \sin(a_{k+1}), \ldots, \sin(a_{k-1})$ will be positive then negative, thus showing that $f(k), f(k+1), \ldots, f(k-1)$ is unimodal. ∎

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 as 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 $f(2), \ldots, f(n)$ 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 $f(x+1), f(x+2), \ldots, f(n)$ 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 $f(1), \ldots, f(n)$ can be computed in $O(\log n)$ time, which involves at most $2.88..\log_2 n + O(1)$ 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, 2 points, 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 which 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 $f(p_1), f(p_2), \ldots, f(w)$ and $f(w), f(w+1), \ldots, f(p_p)$ 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. ∎

Our algorithm involves approximately $2.8808 \log_2 p + O(1)$ computations of f. The extension to the case where 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 [19]. In what follows, we will 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 which 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 Q do not intersect and (YES, A) if they do, where A 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 which separates 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 which 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 5 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 2 polygons):

1) - "Cover Q (resp. P) with 2 lines of support intersecting in P (resp. 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$(p_1 q)$. 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 $q_i q_{i+1}$ and $q_j q_{j+1}$ respectively - See Figure 3-a.

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 $(p_1 q, p_1 q_k)$, 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 2 points, contradicting the convexity of Q. Similarly, by considering the sequence $q_{j+1}, \ldots, q_i$, we find the vertex u which minimizes the angle $(p_1 q, p_1 q_k)$. Call C_1 the pencil $(p_1 u, p_1 t)$ so defined - See Figure 3-b.

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 (resp. C_2) contains q_1) (resp. p_1), the intersection of C_1 and C_2 is a convex quadrilateral, $p_1 Y q_1 X$, as shown in Figure 3-b. 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 which 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 $L_v = \{v_1, \ldots, v_n\}$ 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 chapter, 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 $L_w = \{w_1, \ldots, w_m\}$. In what follows, we rename the former p_1 and q_1, A and B, respectively, as in Figure 3-b. 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 (resp. y) denote the polygonal line AXB (resp. 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 INTERSECT(L_v, L_w) defined recursively as follows:

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.

$i = \lfloor n/2 \rfloor$; $j = \lfloor m/2 \rfloor$;

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

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.

Case 1): Either GF or EH lies on the same side of AB (Fig. 4-a).

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

Case 2): From now on, F and E (resp. G and H) lie on x (resp. y) (Fig. 4-b).

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

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

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

Case 4): Av_i and Bw_j intersect (Fig. 4-d).

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

Case 5): (Fig. 4-e). Let R be the intersection of Av_i and HE.

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

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 with L_w and possibly outputs a point common to P and Q.

Case 1: Suppose that G and F lie on y (the 3 other cases being similar) - See Figure 4-a. By construction, line(BY) intersects P in 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 $\{v_i, \ldots, v_n\}$. By making L_v equal to $\{v_1, v_i, \ldots, v_n\}$, 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 4-b. Since P and Q are convex, their intersection lies totally in V.

Case 2: Corresponds to V empty - See Figure 4-b.

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 $EHBX$, L_w cannot intersect this segment, therefore if L_w intersects the polygonal line $v_1, \ldots, v_i$, it also intersects $v_1 v_i$. Thus the new setting of L_v is legitimate. Same with w_{j+1}. From now on, we know that both $v_i v_{i+1}$ and $w_j w_{j+1}$ lie on the boundary of V.

Case 4: Assumes that Av_i and Bw_j intersect - See Figure 4-d. Since these two segments lie in P and Q respectively, their intersection lies in the intersection of P and Q, which is then non-empty.

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 $v_1, \ldots, v_i$ without first crossing $v_1 v_i$. Similarly, L_v cannot intersect $w_1, \ldots, w_j$ without first crossing $w_1 w_j$. Conversely, if either L_w crosses $v_1 v_i$ or L_v crosses $w_1 w_j$, the intersection belongs to both P and Q. Finally, since $w_1, \ldots, w_j$ (resp. $v_1, \ldots, v_i$) cannot intersect $v_1 v_i$ (resp. $w_1 w_j$), the new setting of L_v and L_w is legitimate.

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 where this additional information is not needed. Instead of a complicated formal definition, Figure 5-a 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 4-b). Since P lies in the pencil (BY, BX), it does not intersect line(EH), therefore 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 for 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 6 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 will show that if $v_2 = p_l$ then, for each u between k and $l-1$, the intersection Q_k is either empty or positive. Starting with the obvious observation that initially v_1 and v_2 are consecutive around P $(l = k+1)$, 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 which reduce L_v from $\{v_1, \ldots, v_n\}$ to $\{v_1, v_i, \ldots, v_n\}$. 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 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 non-empty intersection between L_v and L_w, which is excluded.

Let us now come back to the final stage where L_v has fewer than 6 vertices. Let p_i be the vertex v_2 and p_j the vertex v_{n-1}. We have just showed 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 $Q_{i-1}, \ldots, Q_j$ (note that there are at most 4 of them to consider) is empty, L_P can be set to Q_k - See Figure 5-a. Otherwise, there exists a pair (Q_{k-1}, Q_k) with Q_{k-1} positive and Q_k negative (Fig. 5-b). Observing that the angles $(p_k q_1, p_k q_l)$ are bimodal for $l = 1, \ldots, q$ (here we measure angles counterclockwise with values between -180 and $+180$ degrees), we can find the vertex x (resp. y) of Q which minimizes (resp. 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. A common point is returned when the polygons intersect and a pair of parallel separating lines otherwise.

While 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 the boundaries of P and Q intersect or not.

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

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

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 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.

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 where 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 which we will 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 will be used to reduce the size of a problem by a constant factor.

Since all these techniques assume some kind of 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, it has been a standard method to represent convex polyhedra by a description of the planar graph along with the geometric location of the vertices [14]. 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 $P_1, \ldots, P_p$ 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 which we will 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 which 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, IGG 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 [9] 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 ith vertex of the jth 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 which are also vertices of P may be adjacent 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 jth 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 $P_{i,i+1}$ 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 having 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 (non-degenerate) 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 non parallel to K and output an intersection point. So, let's assume in the following that the intersection $K \bigcap T$ 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 (Fig.8).

Let a_i (resp. b_i) be the vertex of P_i with minimum (resp. maximum) coordinate in the direction $K \bigcap M$. 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 (resp. b_i) on the M-plane, denoted a'_i (resp. b'_i) is unique. We first show that the polygon $Q = a'_1 \dots a'_p b'_p \dots b'_1$ 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 $K \bigcap M$-coordinate cannot be greater than the $K \bigcap M$-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. ∎

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 which 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. ∎

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 $M \bigcap K$-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.

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, $P_l, \ldots, P_m$ $(l \leq m)$. Let w_i, w'_i be the endpoints of the intersection of T and P_i, and W denote the polygon $w'_l \ldots w'_m w_m \ldots w_l$ - See Figure 11. 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 10.

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 11.

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. ∎

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 $q_1, \ldots, q_k$ the vertices of $Q_{b,a}$, that is, the vertices of Q lying between b and a. Note that $q_1, \ldots, q_k$ are the intersections of the plane T with consecutive lateral edges of some drum $P_{i,i+1}$, say, $e_1, \ldots, e_k$. Since all the edges $e_1, \ldots, e_k$ must pass through consecutive vertices of P_i, $x_1, \ldots, x_k$, 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 $x^+_{i,1}, x^+_{i,2}, \ldots$ 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 (wlog). 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 without intersecting P_{i+1} - See Figure 12-a. 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 $x_1, \ldots, x_k$ 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 12-b. Let $x_u x_{u+1}$ (resp. $y_v y_{v+1}$) be the edge of P_i (resp. P_{i+1}), given in clockwise order, which contains a (resp. 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 will consider the polygon P_i (resp. P_{i+1}) with its vertices $x^+_{i,1}, x^+_{i,2}, \ldots$ (resp. $x^-_{i+1,1}, x^-_{i+1,2}, \ldots$). 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 $q_1, \ldots, q_k$ are exactly the intersections of T with the lateral edges emanating from $x_{l+1}, x_{l+2}, \ldots, x_u$ - See Figure 12-b. Otherwise, if the lateral edge emanating from x_{u+1} intersects T, the vertices $q_1, \ldots, q_k$ of Q are determined by the set of vertices $x_{u+1}, \ldots, x_l$ - See Figure 12-c. 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.

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}"

Wlog, assume that T intersects P_i. Since T intersects a set of consecutive preprocessing polygons $P_l, \ldots, P_m$, 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.

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 with 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 non-intersection 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 13-a.

If v is b or c (say b, wlog), 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 adjacent to the vertex a of Q, and which separates W and R. Call L^+, D^+, F^+ the halfspaces delimited by L, D, F respectively, which do not contain the vertex c - See Figure 13-b. 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, then calls 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 which 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 which lies in P in $O(\log p)$ time. There are essentially two cases to consider:

1. If an intersection is detected while intersecting $Q_{b,a}$ (resp. $Q_{a,c}$) with L, then S is exactly the intersection of $Q_{b,a}$ (resp. $Q_{a,c}$) with L, and we can compute it in $O(\log p)$ time (IGL) - See Figure 14-a.

2. If W and L intersect then IGL will provide the two edges of W which intersect L, say, ab and $a'b'$ (fig.14-b). It is clear that if A (resp. B) is the point on the boundary of $Q_{a,b}$ (resp. $Q_{a'b'}$) which intersects L and does not lie on ab (resp. $a'b'$), 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 respectively p and q vertices. 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 16-a. Thus we can maintain a coordinate-free environment. If either P or 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 which 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 2 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 15-a.
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 15-b. Let $e_1, \ldots, e_k$ 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 denote a plane parallel to L which intersects an endpoint of e_1 but does not intersect v. Call $w_1, \ldots, w_k$ the intersections of the plane U with the infinite lines passing through $e_1, \ldots, e_k$, respectively (note that $e_1 = vw_1$) - See Figure 15-c. Since $w_1, \ldots, w_k$ form a convex polygon, the distance from w_l to the plane T passing through L and perpendicular to U gives a bimodal sequence if it is counted positive on one side of T and negative on the other. Thus, its extrema can be found in $O(\log p)$ time. Let w_l be one of them; since the plane passing through L and vw_l is a plane of support of the polyhedron formed by $v, w_1, \ldots, w_k$, it is also a plane of support of P, which completes the proof. ∎

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 (resp. S) the intersection of T and P (resp. Q), and let (L_P, L_Q) be a pair of parallel separating lines for R and S respectively. If T_P (resp. T_Q) is a plane of support of P (resp. Q) passing through L_P (resp. 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 16. 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 non-empty) 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 16-b,c. The portion of P which does not lie on the same side of T as L can be rejected since it cannot intersect with Q. This gives us a means to reduce 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 16-b. Since in this last case, IHG will also indicate edges of P (resp. Q) which intersect L_P (resp. 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 16-c. 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 will 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) There now remains to test the intersection of Q and $P_{i,i+1}$. Let $x_1, \ldots, x_k$ 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 which contains the edge $x_u x_{u+1}$). We have seen 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 $P_{i,i+1}$ which intersect

T_j. Having an implicit description of S_j, we can apply the procedure described earlier, using first 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 18-a.

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 18-a illustrates only the simplest case since F is not necessarily a triangle. However, we can remedy this discrepancy easily. Let y_1 be the endpoint of the edge e which lies on P_{i+1} ($e = x_1y_1$) and let $y_1, \dots, y_n$ denote the vertices of P_{i+1} in clockwise order. F is a convex polygon $y_1, y, y_l, \dots, y_m, y', y_1$ - See Figure 18-b,c. We can determine y and y' in $O(\log p)$ time by intersecting P_{i+1} with T_j and T_{j+1}, using the IGL algorithm (which actually must have been done already). If y and y' do not lie on the same edge of P_{i+1}, we carry on the previous binary search on the planes $T_l', \dots, T_m'$, where T_j' is now the plane containing e and y_j. If the algorithm does not return, it will reduce P to a convex polyhedron B with two parallel faces on P_i and P_{i+1}, denoted x_1ab and $y_1a'b'$ respectively, both of which are triangles - See Figure 19. Let F_j (resp. F_{j+1}) be the face of B lying on T_j (resp. T_{j+1}). In addition to ab and $a'b'$, B may contain other edges $f_1, \dots, f_t$ intersecting both F_j and F_{j+1}. These edges lie on consecutive lateral edges of $P_{i,i+1}$, say, $e_1, \dots, e_t$ in clockwise order. Our next task is to compute an implicit description of this set of edges, that is, to determine e_1 and e_t.

The following fact will permit us to compute e_1 and e_t in constant time. We can always assume that a, b (resp. a', b') occur in clockwise order in a traversal of the boundary of P_i (resp. P_{i+1}). Let g be a lateral edge of $P_{i,i+1}$ intersecting both F_j and F_{j+1}, with g_1 (resp. g_2) the endpoint of g lying on P_i (resp. P_{i+1}). Note that by construction of B, any edge intersecting F_j intersects F_{j+1} as well, and vice-versa. We can observe that if g_1 occurs between x_1 and a (resp. b and x_1) in clockwise order, g_2 must lie between b' and y_1 (resp. y_1 and a') in clockwise order. Wlog, suppose that g_1 occurs between x_1 and a. Let $x^+_{i,l}$ be the vertex of P_i such that a lies on the edge $x^+_{i,l}x^+_{i,l+1}$. Since lateral edges can only intersect at their endpoints, the lateral edge of $P_{i,i+1}$ adjacent to $x^+_{i,l}$ (which is uniquely defined by the preprocessing) also intersects both F_j and F_{j+1}. This shows that lateral edges of $P_{i,i+1}$ intersect F_j and F_{j+1} if and only if the lateral edge adjacent to $x^+_{i,l}$ or $x^+_{i+1,l}$ intersects T_j. This gives us a convenient way to determine e_1 and e_k in constant time with the technique already used in the IHG algorithm. Namely, let $x^-_{i+1,u}x^-_{i+1,u+1}$ be the edge of P_{i+1} which intersects F_j and let $x^+_{i,m}$ be the vertex of P_i in one-to-one correspondence with $x^-_{i+1,u+1}$. It is then clear that e_1 is the edge $x^+_{i,m}x^-_{i+1,u+1}$ and e_t the lateral edge adjacent to $x^+_{i,l}$. All the lateral edges between e_1

and e_t also intersect F_j and F_{j+1}, that is, the edges adjacent to $x^+_{i,m}, x^+_{i,m+1}, \ldots, x^+_{i,l}$. Recall that the one-to-one correspondence between $\{x^+_{i,1}, x^+_{i,2}, \ldots\}$ and $\{x^-_{i+1,1}, x^-_{i+1,2}, \ldots\}$ established in the preprocessing allows random-access to the lateral edges of $P_{i,i+1}$.

4) Having an implicit description of $e_1, \ldots, e_t$, we can define U_j as the plane containing x_1 and e_j and apply the procedure of 2) on this set of planes - See Figure 19. This will either return a point of the intersection of P, Q, and U_j, or produce 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 which 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 5 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 the 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 = ⌊(l+m)/2⌋
        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 be 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 $T_l', \ldots, T_m'$ (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 5 faces"

Check if Q lies entirely inside K by testing if q_1 does. In the affirmative, return (YES, q_1). Otherwise, apply the IHG algorithm to test if 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. 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. Same complexity as (2), since computing an implicit description of $y_l, \ldots, y_m$ takes constant time.
4. Same as (2).
5. Essentially a repeated application of IHG to Q and a triangle or a quadrilateral ($\log^2 q$). ∎

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 [19], 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 [9], 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 [17] or algorithms efficient on the average [4].

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 [7].

Acknowledgment: We wish to thank Garret Swart for pointing out a flaw 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 Publishing Company, Reading, Massachussetts, 1974.

[2] Bentley,J.L., Haken,D., and Hon,R., Fast geometric algorithms for VLSI tasks, Proc. Comput. Conf., 1981, pp. 88-92.

[3] Bentley,J.L. and Ottmann,T., Algorithms for reporting and counting geometric intersections, IEEE Trans. Comput., Vol. C–28, pp. 643–647, Sept. 1979.

[4] Bentley,J.L. and Shamos,M.I., Divide and Conquer for linear expected time, Information Processing Letters, Vol. 7, pp. 87–91, 1978.

[5] Chazelle,B., Computational Geometry and Convexity, PhD thesis, Yale University, 1980.

[6] Chazelle,B. and Dobkin,D.P., Detection is easier than computation, Proceedings of the 12th Annual ACM Symposium on Theory of Computing, Los Angeles, California, May, 1980, pp.146-153.

[7] Dobkin,D.P. and Kirkpatrick, D.G., Fast detection of polyhedral intersection, Theoretical Computer Science, 27, 1983, pp. 241–253.

[8] Dobkin,D.P. and Lipton,R.L., Multidimensional searching problems, SIAM Journal on Computing, Vol.5, No.2, June, 1976, pp.181-186.

[9] Dobkin,D.P. and Munro,J.I., Efficient uses of the past, Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science, Syracuse, pp. 200-206, Oct. 1980.

[10] Dobkin,D.P. and Tomlin,D., Cartographic modelling techniques in environmental planning: an efficient system design, Submitted for publication.

[11] Forrest,A., private communication, March 29, 1979.

[12] Kiefer,J., Sequential minimax search for a maximum, Proceedings of the American Mathematical Society, Vol.4, 1953, pp.502-506.

[13] Knuth,D.E., The art of computer programming: fundamental algorithms, Addison-Wesley Publishing Company, Reading, Massachussetts, 1968.

[14] Muller,D.E. and Preparata,F.P., Finding the intersection of two convex polyhedra, Theoretical Computer Science, Vol. 7, pp. 217–236, 1978.

[15] Newman,W. and Sproull,R., Principles of Interactive Computer Graphics, Second Edition, McGraw Hill, New York, 1979.

[16] Nievergelt,J. and Preparata,F.P., Plane-sweeping algorithms for intersecting geometric figures, Comm. ACM, Vol. 25, No. 10, pp. 739–747, 1982.

[17] Rabin,M., Probabilistic algorithms, in Algorithms and Complexity: New directions and Recent Results, Traub,J., Ed. Academic Press, 1976.

[18] Shamos,M.I., Geometric Complexity, Proceedings of the 7th Annual ACM Symposium on Theory of Computing, Albuquerque, New Mexico, May, 1975, pp.224-233.

[19] Shamos,M.I., Computational Geometry, PhD thesis, Yale University, May, 1978.

[20] Shamos,M.I. and Hoey,D., Geometric intersection problems, 17th Annual IEEE Symposium on Foundations of Computer Science, Houston, Texas, October, 1976, pp.208-215.

[21] Tomlin,D., Private communication to D. Dobkin, 1978.

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				$\log^3 n$

Figure 1: The time bounds of our algorithms.

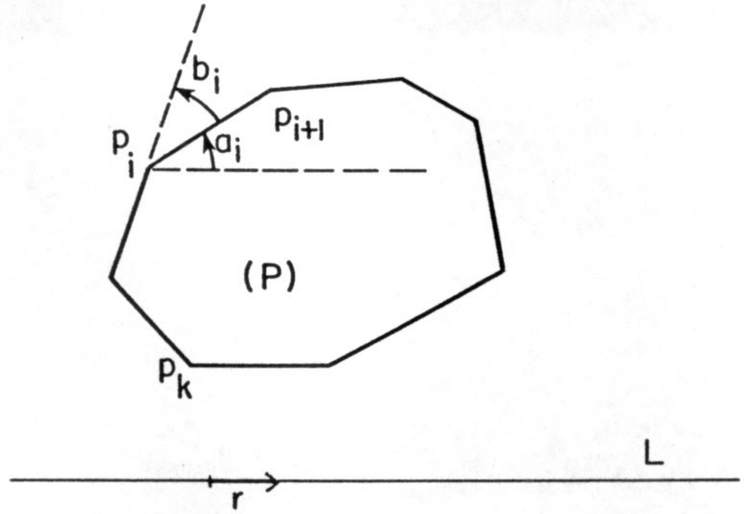
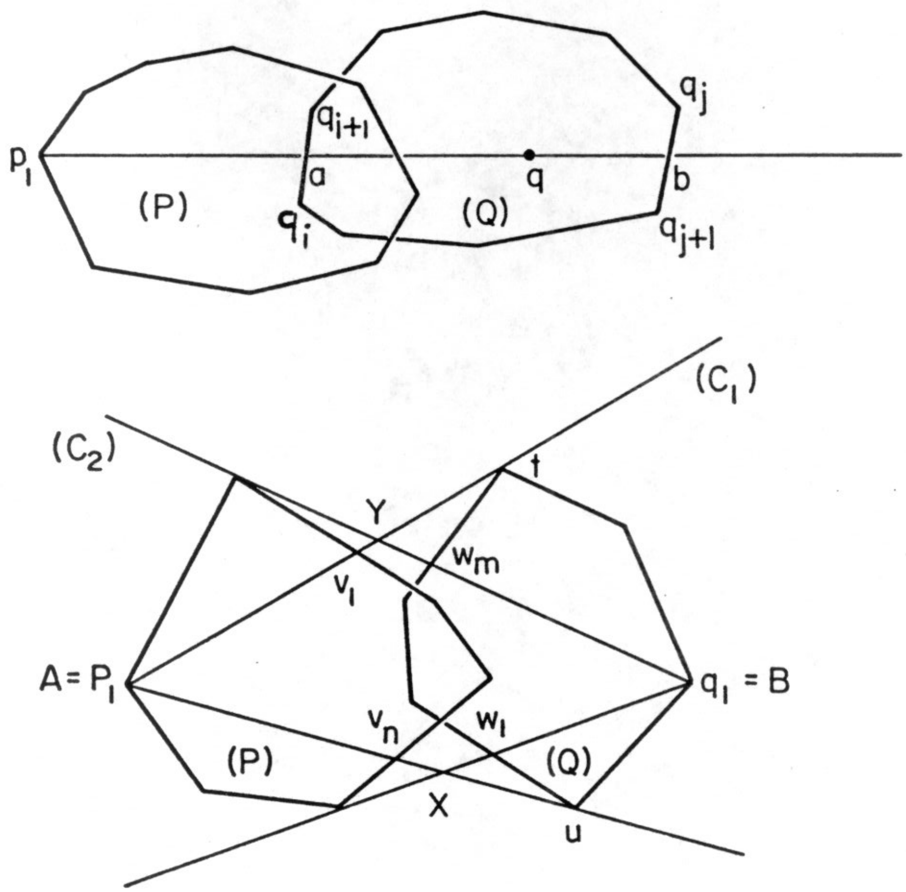


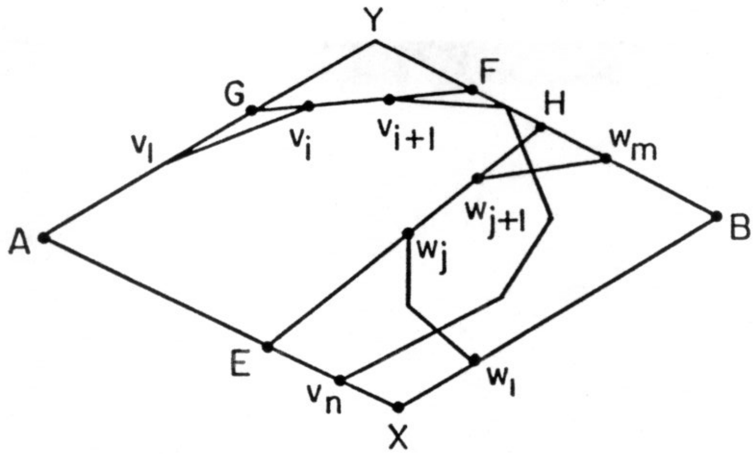
Figure 2: The distance from a convex polygon to a line is bimodal.



a)

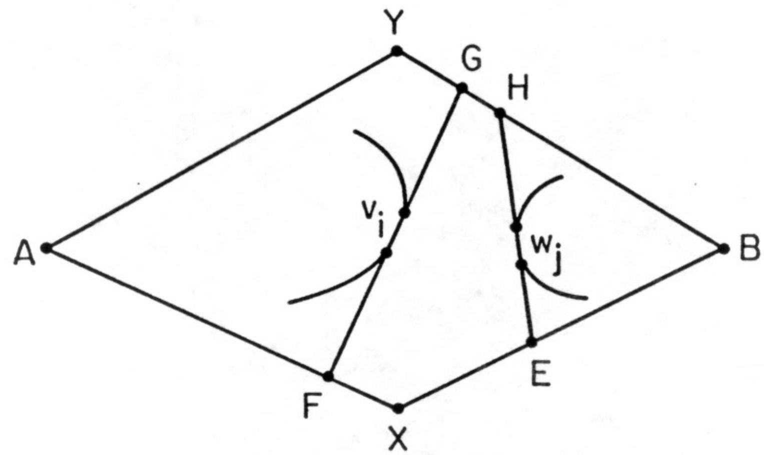
b)

Figure 3: We form AYBX as a bounding quadrilateral in which PAQ lies if it exists.



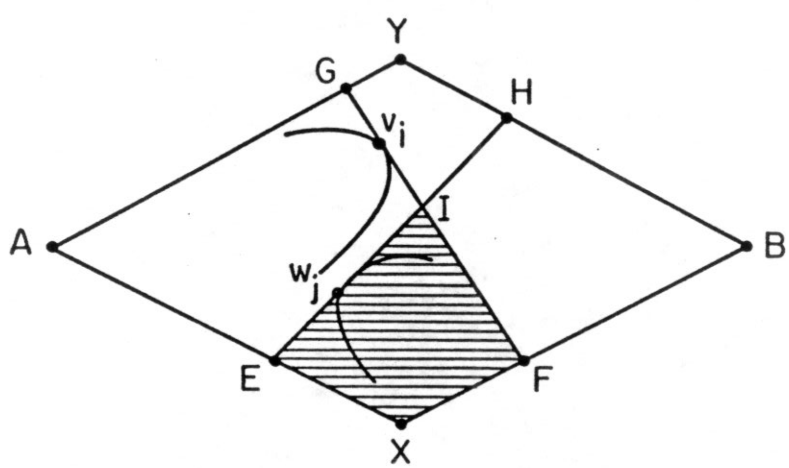
a)

<u>Case 1</u>: GF (resp. EH) lies on the same side of AB, in which case we eliminate half of L_v (resp. L_w).



b)

<u>Case 2</u>: 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.

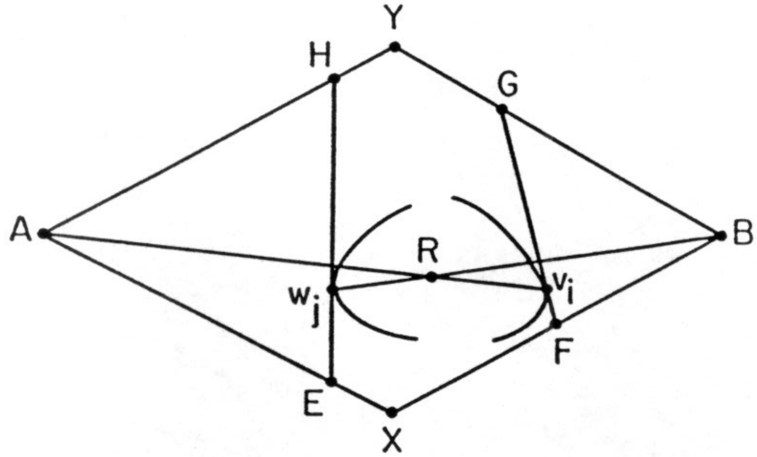


c)

<u>Case 3</u>: GF and EH intersect.

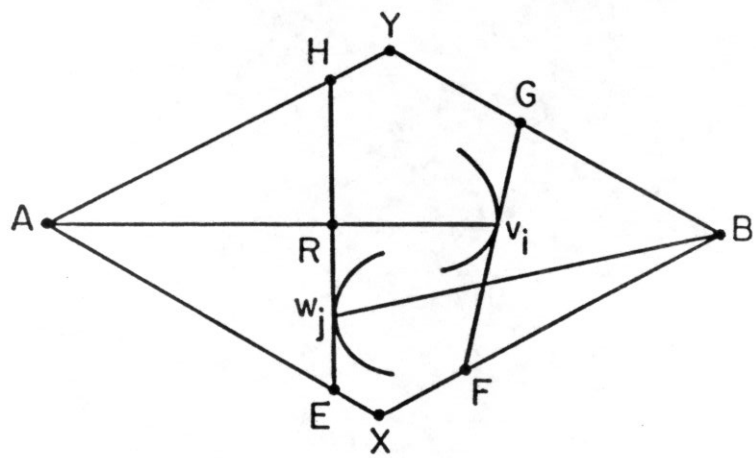
(Figure 4 .../...)

d)



Case 4: Av_i and Bw_j intersect, in which case POQ contains their intersection point.

e)



Case 5: Av_i lies strictly "above" (or "below") Bw_j.

Figure 4: The algorithm INTERSECT

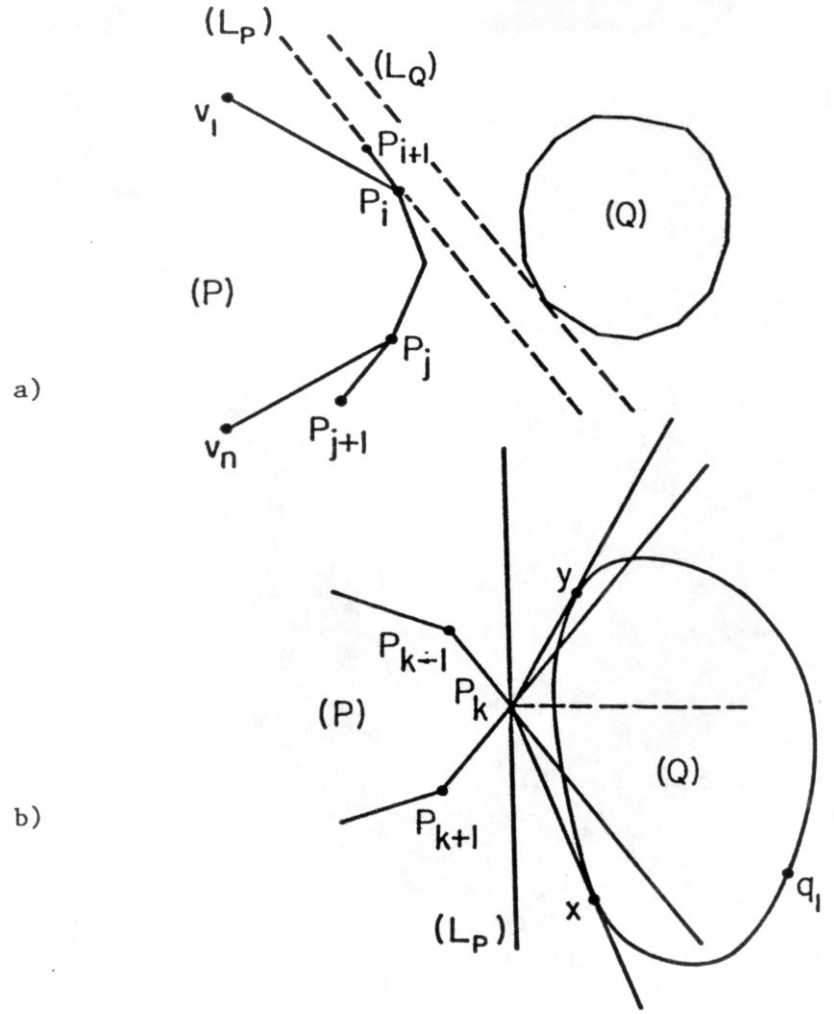


Figure 5: Computing a pair of separating lines.

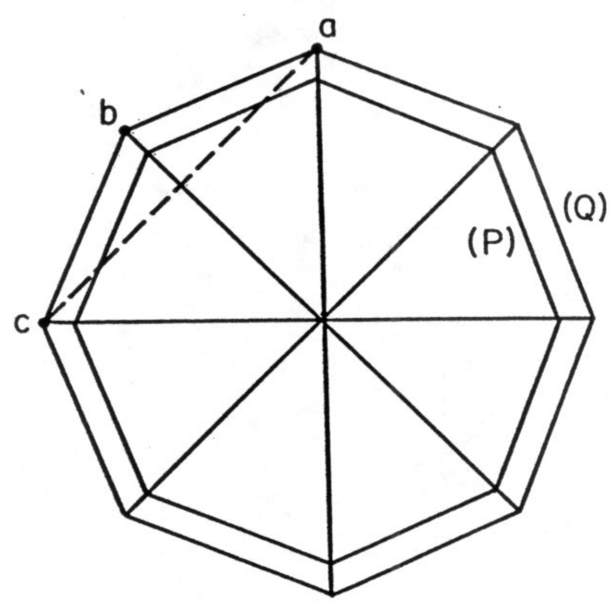


Figure 6: Intersecting polygonal lines.

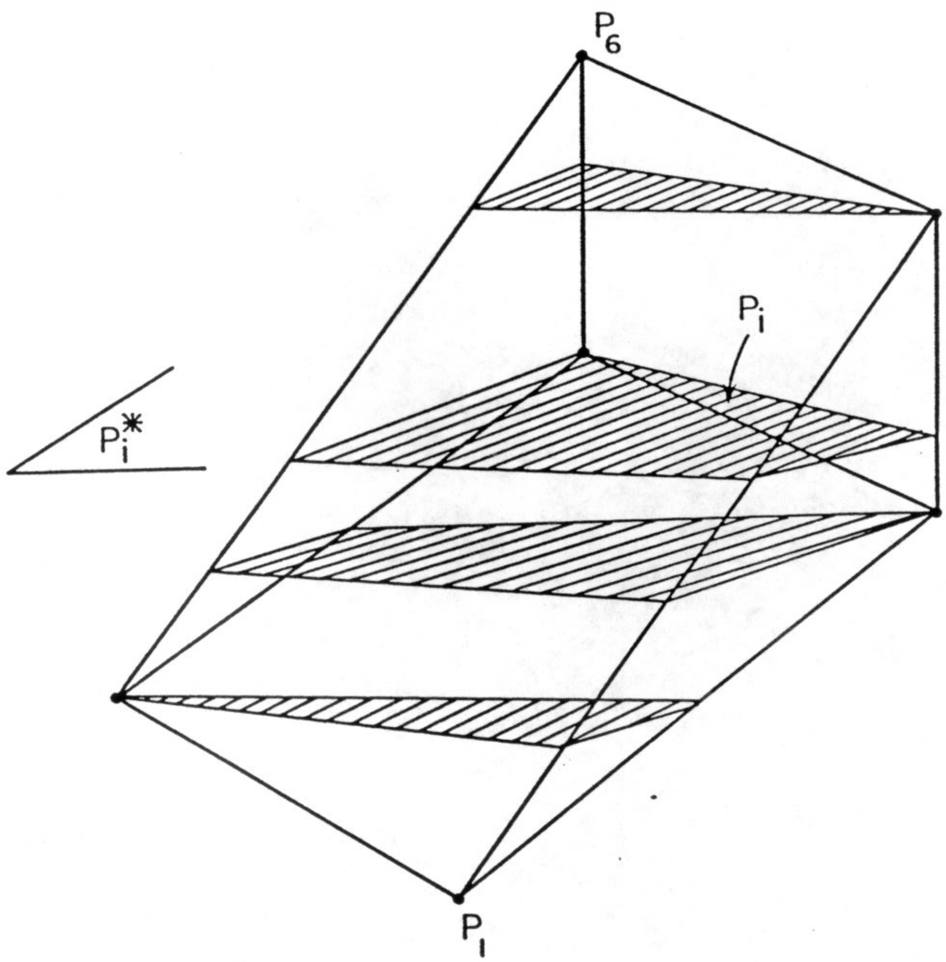


Figure 7: Preprocessing three-dimensional objects.

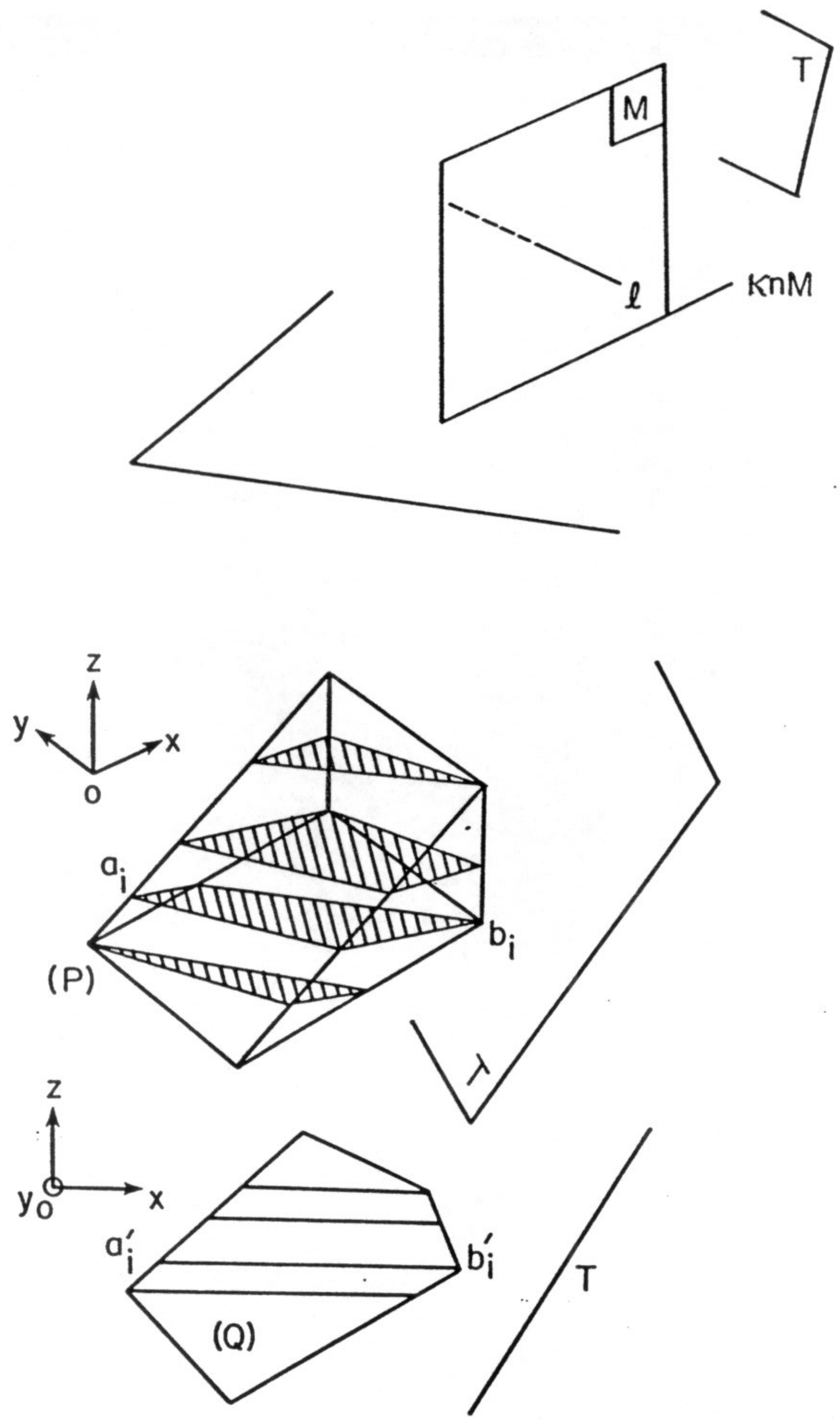


Figure 8: The algorithm IHP.

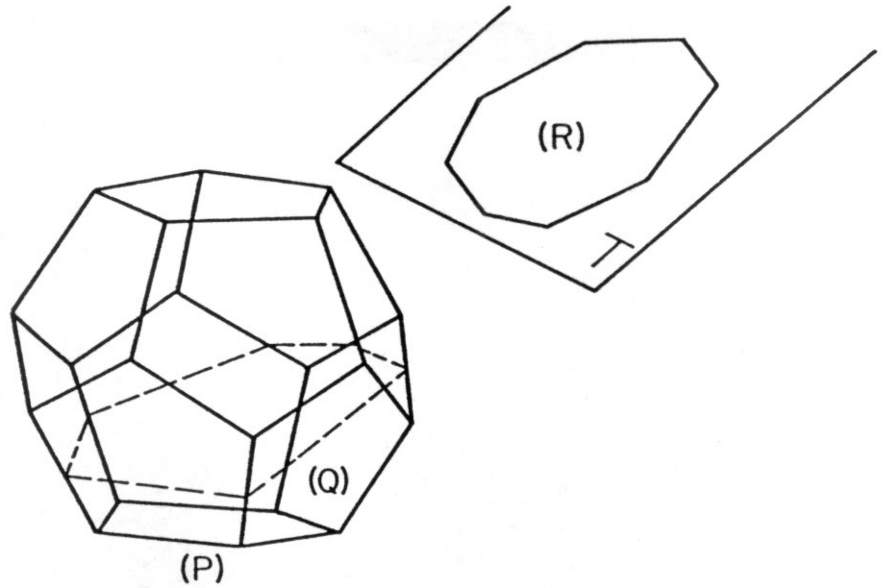


Figure 9: Intersection of a polyhedron and a polygon.

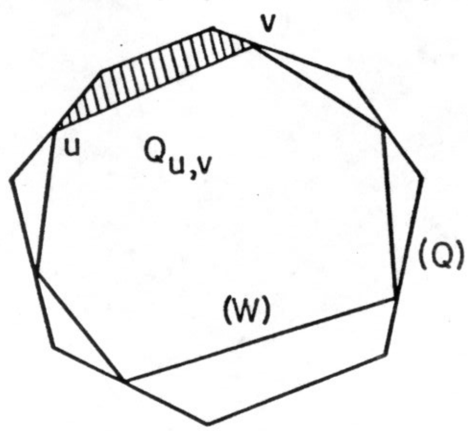


Figure 10: The polygons $Q, W, Q_{u,v}$.

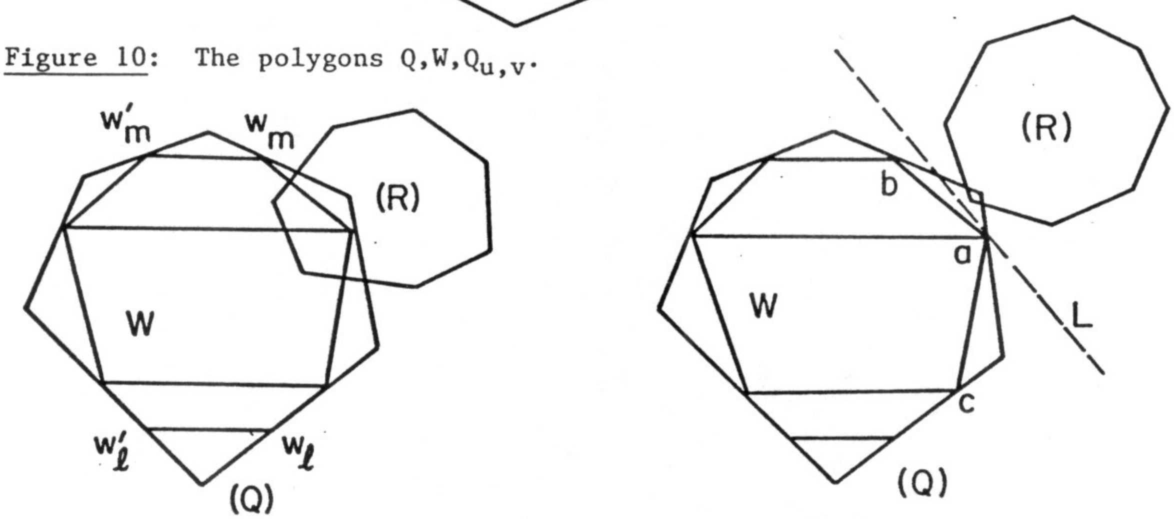
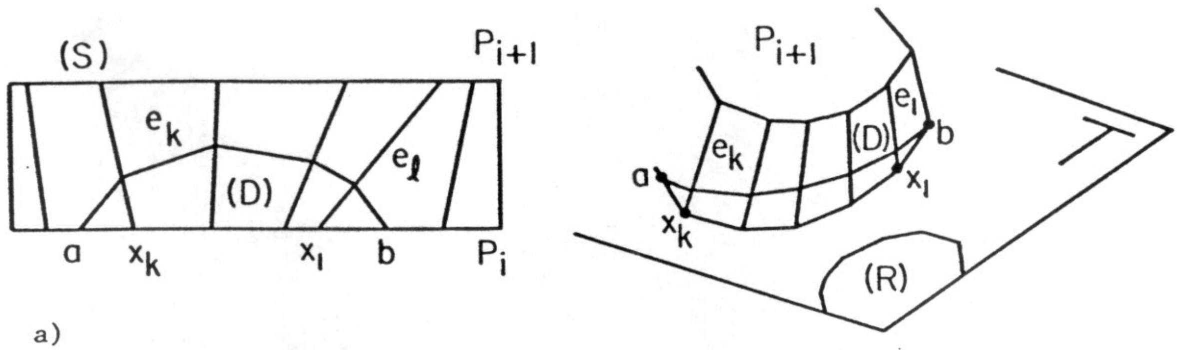
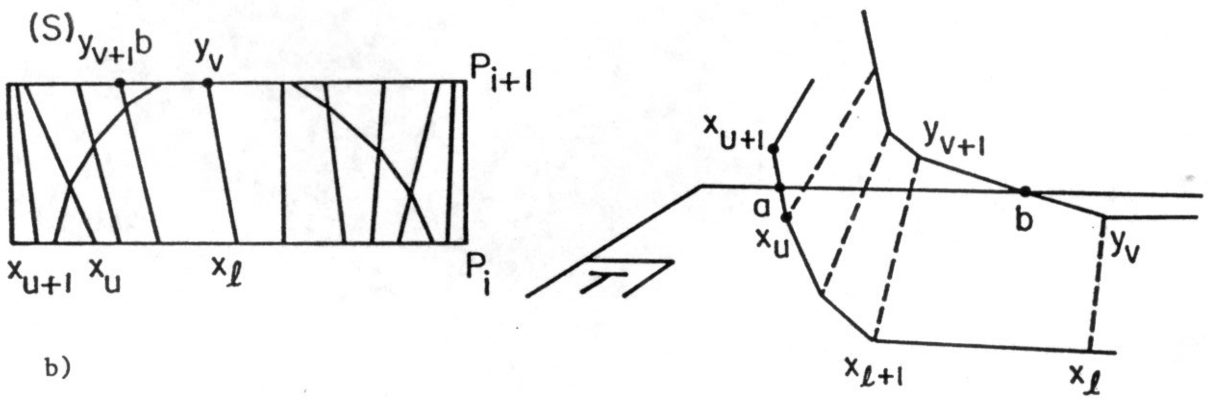


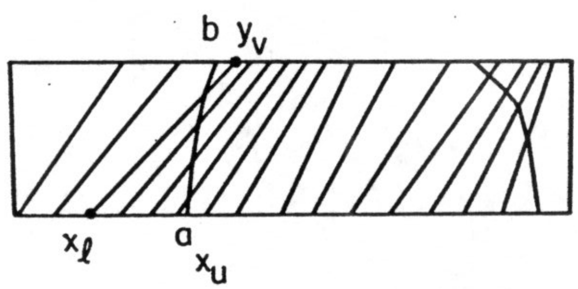
Figure 11: The two cases of interesction of Q and R.



a)



b)



c)

Figure 12: Computing $Q_{b,a}$.

a)

b)

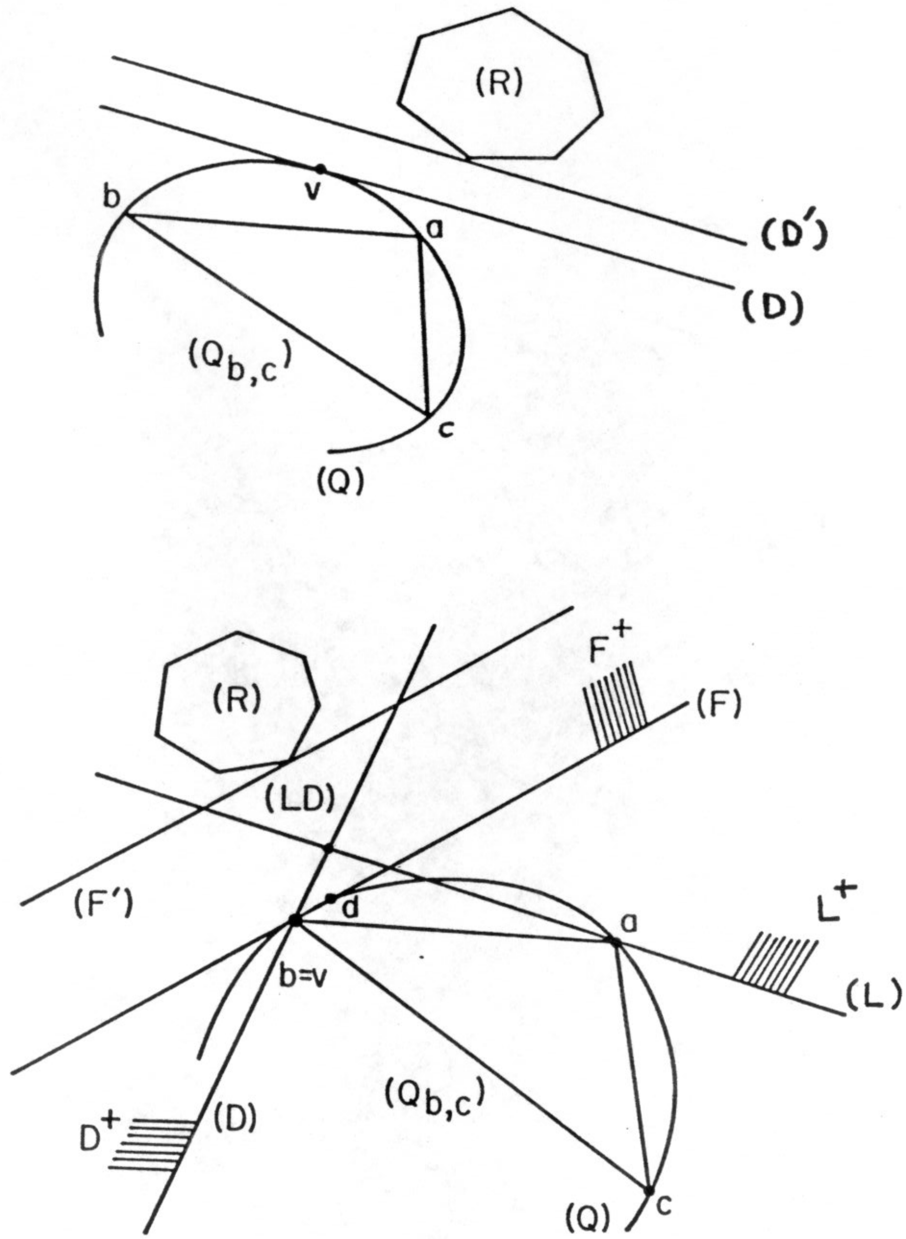
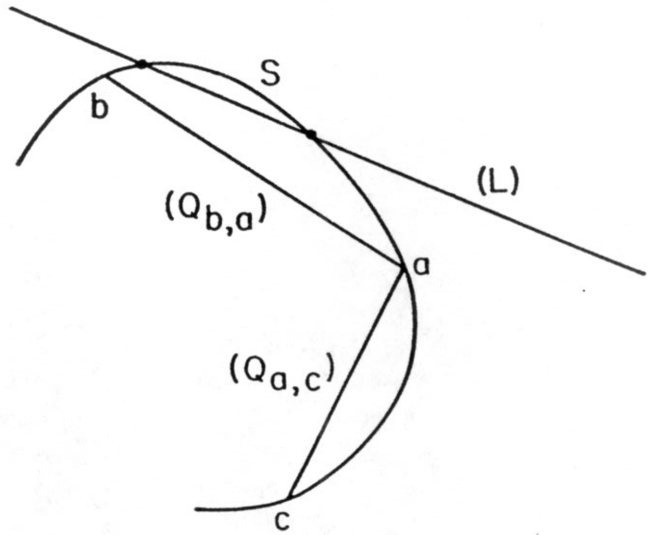
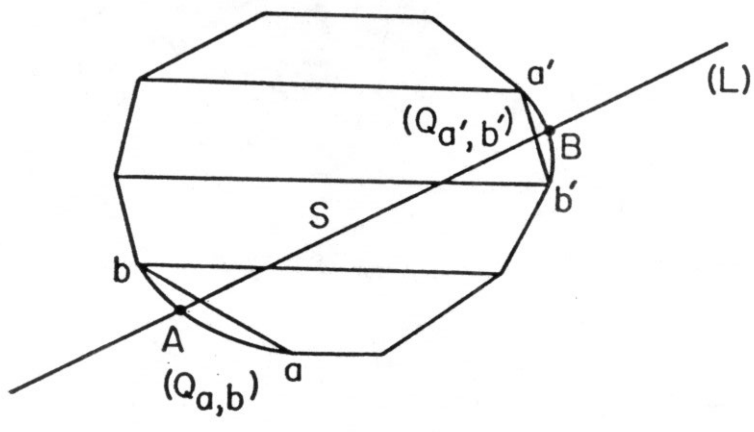


Figure 13: Computing a pair of separating lines for Q and R.



a)



b)

<u>Figure 14:</u> The algorithm IHL.

a)

b)

c)

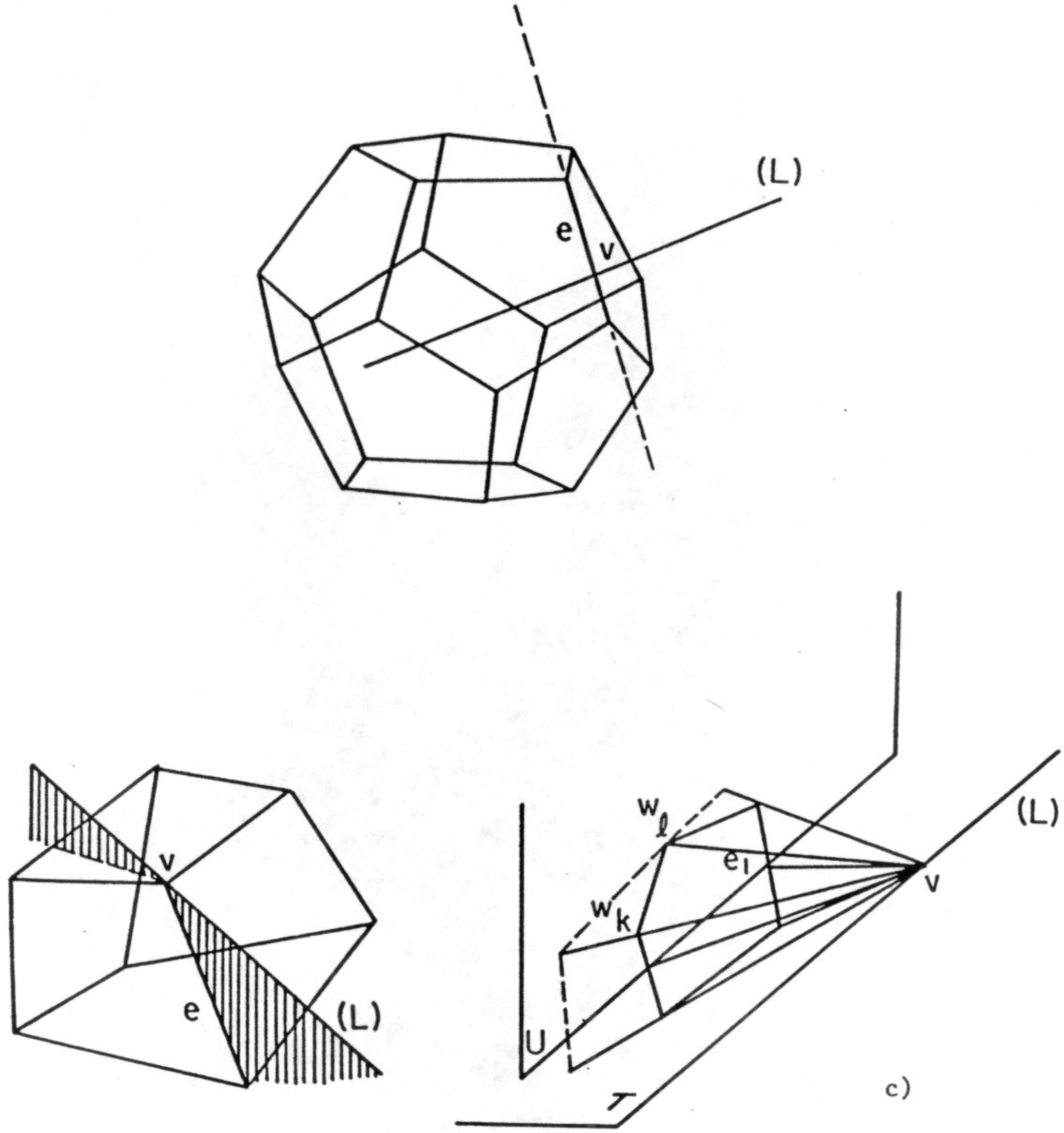


Figure 15: Computing a plane of support of P.

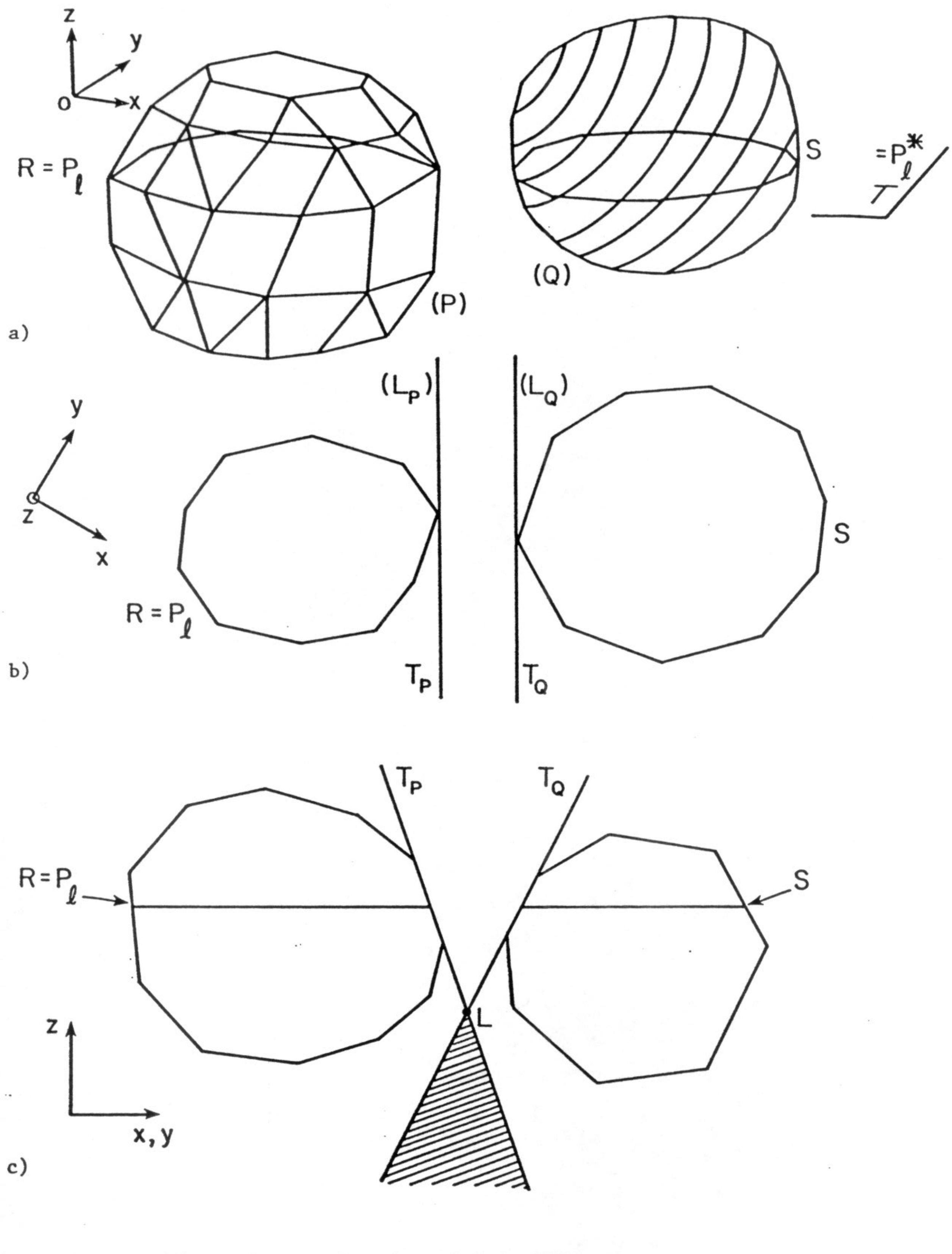


Figure 16: Reducing the size of P in IHH.

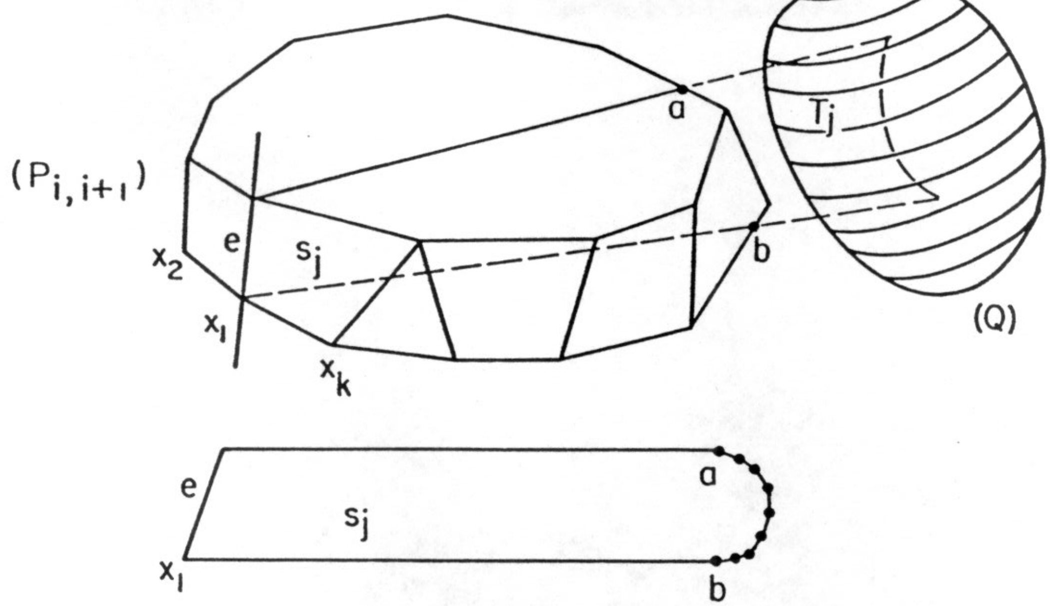


Figure 17: Reducing the cap $P_{i,i+1}$ in IHH.

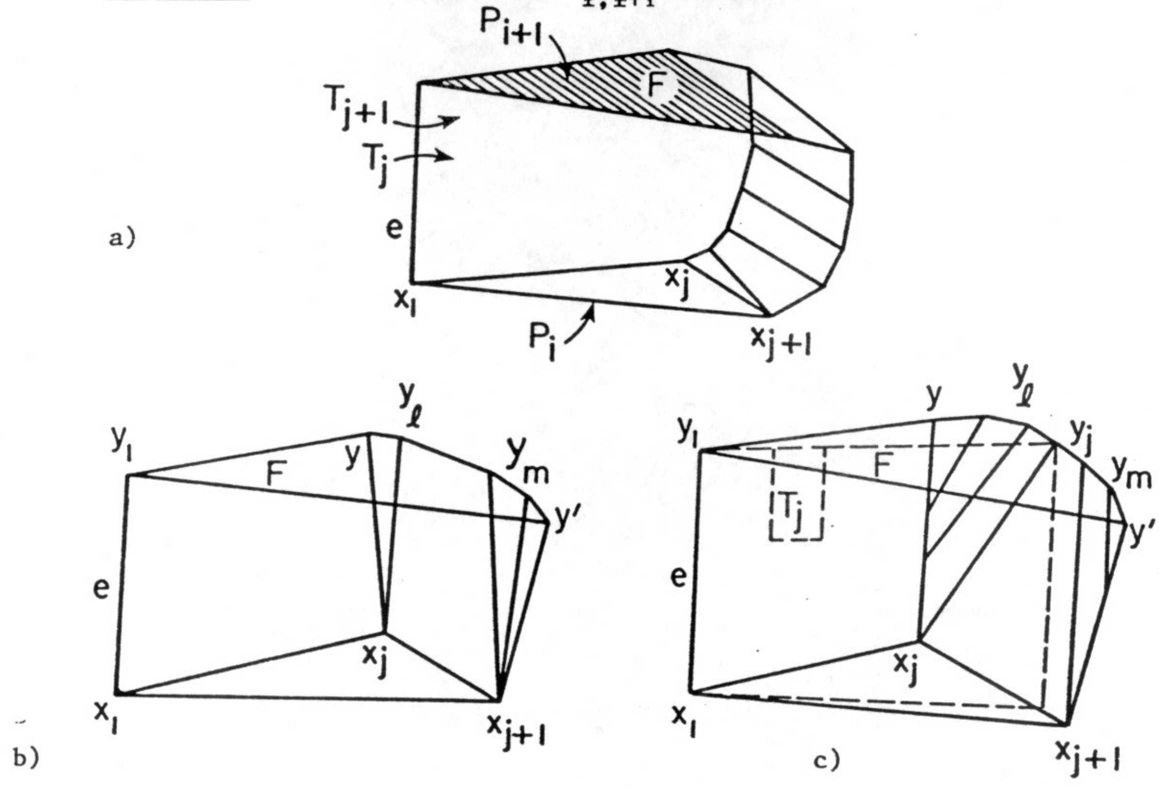
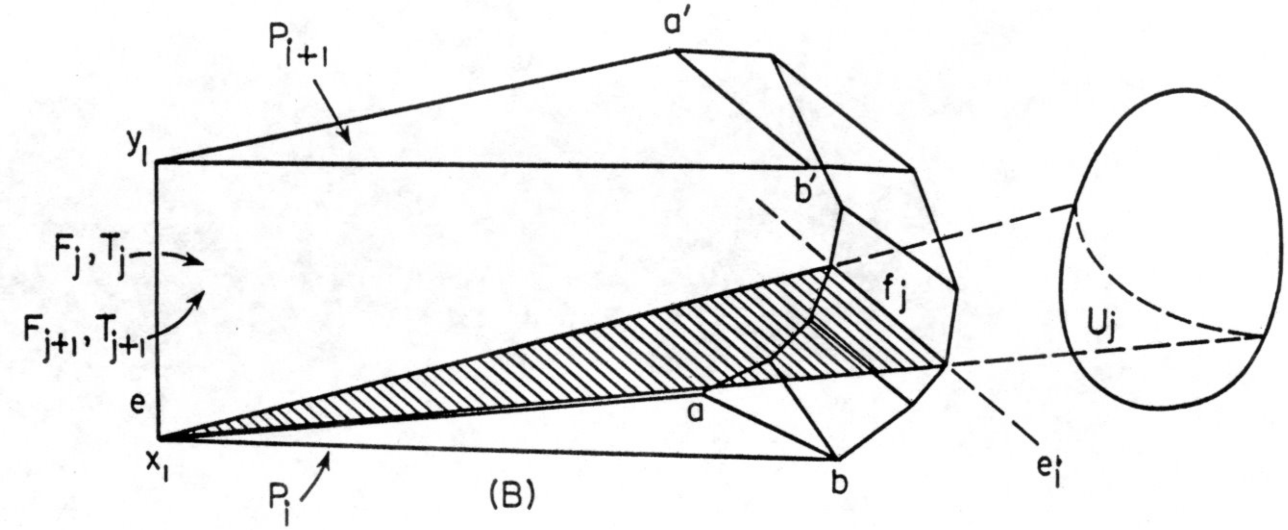


Figure 18: The cap $P_{i,i+1}$ after reduction.



<u>Figure 19</u>: Reducing the slice A in IHH.