Theoretical Computer Science 7 (1978) 217–236.
© North-Holland Publishing Company

FINDING THE INTERSECTION OF TWO CONVEX POLYHEDRA*

D. E. MULLER[1] and F. P. PREPARATA[2]
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, IL, U.S.A.

Communicated by M. Nivat
Received November 1977
Revised March 1978

Abstract. Given two convex polyhedra in three-dimensional space, we develop an algorithm to (i) test whether their intersection is empty, and (ii) if so to find a separating plane, while (iii) if not to find a point in the intersection and explicitly construct their intersection polyhedron. The algorithm runs in time $O(n \log n)$, where n is the sum of the numbers of vertices of the two polyhedra. The part of the algorithm concerned with (iii) (constructing the intersection) is based upon the fact that if a point in the intersection is known, then the entire intersection is obtained from the convex hull of suitable geometric duals of the two polyhedra taken with respect to this point.

1. Introduction

Finding the intersection of two convex polyhedra in three-dimensional space is a classical problem in computational geometry [1]. A simple but time-consuming solution to this problem is known, and it is so trivial that it is not even worth a literature reference. It goes as follows. Let $\mathscr{A}$ and $\mathscr{B}$ be two convex polyhedra; test each face of $\mathscr{A}$ against each face of $\mathscr{B}$ to see if they intersect; if no intersection is found, then the intersection of the two polyhedra is empty, otherwise it can be simply constructed. It is clear that such an algorithm could use $O(n^2)$ operations if n is the sum of the numbers of vertices of $\mathscr{A}$ and $\mathscr{B}$.

The search for a more efficient procedure has in the past met with no success. Several facts, however, suggested that a more efficient method should exist: first, it is well-known that two polygons in the plane can be intersected in time linear in the sum of their numbers of vertices [2]; second, several analogies exist between the plane and the space, i.e., convex hulls of n-point sets [3] and maxima of sets of n vectors [4] can be found in time $O(n \log n)$ in both two and three dimensions. In spite of this, no generalization of the polygon intersection algorithm has been found.

* This work was supported in part by the National Science Foundation under Grant MCS76-17321 and in part by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259.
[1] Also, Departments of Mathematics and of Computer Science.
[2] Also, Departments of Electrical Engineering and of Computer Science.

In special cases, however, better than quadratic time methods have been known for some time. Specifically, after the development of the Lee–Preparata algorithm for locating a point in a planar subdivision [5], it was realized[3] that the intersection of two polyhedra can be found in time $O(n \log^2 n)$ if a vertex of one polyhedron lies inside the other polyhedron.

Alternately (see [1], p. 162), Shamos conjectured that polyhedron intersection could be obtained as a merge step of a divide-and-conquer algorithm for the intersection of half-spaces.

In this paper we present an algorithm for solving the problem of intersecting two polyhedra in time $O(n \log n)$. The algorithm tests whether the intersection is empty and, if not, explicitly constructs it. The approach is based on the fact that, if a point p in the intersection is known, the intersection can be obtained through geometric dualization. Specifically, the two polyhedra are both transformed into their geometric duals with respect to p, and the convex hull of the dual, which can be found in time $O(n \log n)$, is the dual of the intersection polyhedron (Section 3). When such a special point p is not available, by deploying known techniques in time $O(n \log n)$ we can test whether the intersection is nonempty and, if so, obtain a point in the intersection (Section 4). The algorithm requires that each polyhedron be represented by a very versatile data structure called the doubly connected edge list, which can be obtained from a more conventional representation – produced by the convex hull algorithm – in time linear in the number of vertices (Section 2).

2. Derivation of a doubly connected edge list for a planar graph

Let $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$ be the sets of vertices and edges respectively, of a planar graph embedded in the plane without crossing edges. We assume that (V, E) is represented as follows. To vertex $v_j \in V$ there corresponds cell $H[j]$ of an array $H[1:n]$, which contains a pointer to the first term of the cyclic list of the edges incident on v_j arranged in the order in which they appear as one proceeds counterclockwise around v_j. The latter lists are realized by means of two arrays VERTEX$[1:2m]$ and NEXT$[1:2m]$ so that (VERTEX$[i]$, NEXT$[i]$) is the format of the list nodes. This representation of the graph (V, E) is precisely the one obtained by the algorithm of Preparata and Hong [3] which constructs the convex hull of a set of points in three dimensions: indeed the surface of a convex polyhedron is topologically a planar graph. We shall call this collection of lists the *vertex-to-edge* representation of a planar graph.

Although the vertex-to-edge list is one of the most commonly used representations for a planar graph, it has the disadvantage that the dual graph, i.e., the graph whose vertices correspond to faces of the original graph, is not readily available.

[3] Private communications between M. I. Shamos and F. P. Preparata, May 1976. See also [1] p. 160.

For this one would have to develop the *f*
graph in which each face refers to a cyclic

A more convenient representation for
doubly connected edge list (DCEL), fr
information either about the edges incid
face. We will now describe the DCEL a
which to obtain it from the more conventi
graph in time proportional to the numbe

The main component of the DCEL o
There is a one-to-one correspondence b
edge is represented only once, whereas
appeared twice. An edge node consists of
$F2$, and two pointer fields $P1$ and $P2$: the
easily implemented with six arrays with th
The meanings of these fields are as follow
vertex which is the *origin* of the edge, w
manner, the edge receives a conventional
the names of the faces which lie respectiv
oriented from $V1$ to $V2$. The pointer $P1$
the first edge encountered after $(V1V2$
around $V1$ ($V2$). Names of faces and v
example, a fragment of a graph and the
shown in Fig. 1.

It is now easy to see how the edges
enclosing a given face can be obtained fr
and f faces, we can assume we have two
of the vertex and face lists: these arrays c
in time $O(n)$. The following straightforw
sequence of edges incident on v_j as a seq

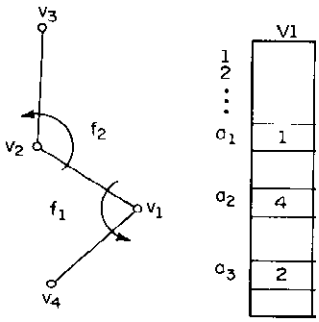


Fig. 1. Illustratio

For this one would have to develop the *face-to-edge* representation of the original graph in which each face refers to a cyclically ordered list of edges which enclose it.

A more convenient representation for this purpose is one which we shall call the *doubly connected edge list* (DCEL), from which we can obtain equally easily information either about the edges incident on a vertex or the edges enclosing a face. We will now describe the DCEL and give in an appendix the algorithm by which to obtain it from the more conventional vertex-to-edge representation of the graph in time proportional to the number n of vertices.

The main component of the DCEL of a planar graph (V, E) is the *edge node*. There is a one-to-one correspondence between edges and edge nodes, i.e., each edge is represented only once, whereas in the vertex-to-edge list each edge appeared twice. An edge node consists of *four* information fields $V1$, $V2$, $F1$, and $F2$, and two pointer fields $P1$ and $P2$: therefore the corresponding data structure is easily implemented with six arrays with the same names, each consisting of m cells. The meanings of these fields are as follows. The field $V1$ contains the name of the vertex which is the *origin* of the edge, whereas $V2$ contains the *terminus*; in this manner, the edge receives a conventional orientation. The fields, $F1$ and $F2$ contain the names of the faces which lie respectively on the left and on the right of the edge oriented from $V1$ to $V2$. The pointer $P1$ ($P2$) points to the edge node containing the first edge encountered after ($V1V2$) when one proceeds counterclockwise around $V1$ ($V2$). Names of faces and vertices may be taken as integers. As an example, a fragment of a graph and the corresponding fragment of the DCEL is shown in Fig. 1.

It is now easy to see how the edges incident on a given vertex or the edges enclosing a given face can be obtained from the DCEL. If the graph has n vertices and f faces, we can assume we have two arrays $HV[1:n]$ and $HF[1:f]$ of headers of the vertex and face lists: these arrays can be filled by a scan of arrays V1 and F1 in time $O(n)$. The following straightforward procedure, VERTEX(j), obtains the sequence of edges incident on v_j as a sequence of addresses stored in an array A.

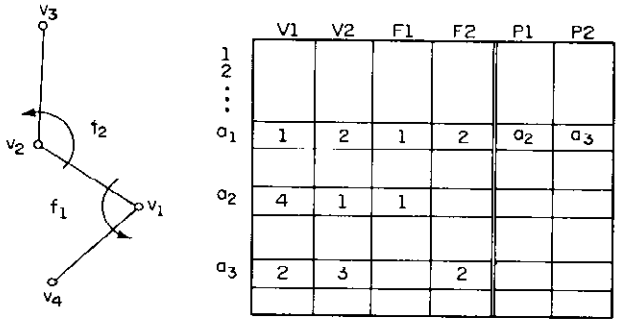


Fig. 1. Illustration of the DCEL.

VERTEX(j)

```
1. begin A[1] ← a0 ← a ← HV[j], i ← 2
2.       If V1[a] = j then a ← P1[a] else a ← P2[a]
3.       While a ≠ a0 do
4.         begin A[i] ← a
5.               If V1[a] = j then a ← P1[a] else a ← P2[a]
6.               i ← i + 1
           end
   end
```

Clearly VERTEX(j) runs in time proportional to the number of edges incident on v_j. Analogously, we can develop a procedure, FACE(j), which obtains the sequence of edges enclosing f_j, by replacing HV and $V1$ with HF and $F1$, respectively, in the above procedure VERTEX(j). Notice that the procedure VERTEX traces the edges counterclockwise about a vertex while FACE traces them clockwise about a face.

In an appendix to this paper we shall show that the DCEL of a planar graph can be obtained from its vertex-to-edge list in time linear in the number of vertices.

3. Finding the intersection of two polyhedra when a point in the intersection is known

Assume that the two convex polyhedra $\mathscr{A}$ and $\mathscr{B}$ in three-dimensional Euclidean space are represented by DCEL's. They are taken as having vertex sets $V_{\mathscr{A}}$ and $V_{\mathscr{B}}$ and face sets $F_{\mathscr{A}}$ and $F_{\mathscr{B}}$ respectively. With each vertex $v \in V_{\mathscr{A}} \cup V_{\mathscr{B}}$ we associate its three Cartesian coordinates $x_1(v), x_2(v), x_3(v)$. With each face $f \in F_{\mathscr{A}} \cup F_{\mathscr{B}}$ we associate a half-space bounded by its plane and containing the corresponding polyhedron. The half-space of f is described by the inequality

$$n_1(f)x_1 + n_2(f)x_2 + n_3(f)x_3 + d(f) \geqslant 0 \tag{1}$$

where $n_1(f), n_2(f), n_3(f)$ and $d(f)$ are four parameters characteristic of f, normalized so that $d(f)$ is either 1, 0, or -1. Our objective is to obtain a DCEL for the intersection of $\mathscr{A}$ and $\mathscr{B}$ along with the coordinates of its vertices and the parameters for its faces.

In this portion of the analysis it is assumed that a point in the intersection of $\mathscr{A}$ and $\mathscr{B}$ is known: the problem of finding such a point will be discussed later. Without loss of generality, the point in the intersection will be taken as the origin, since if it is not the origin a simple translation of coordinates will make it the origin. We begin by assuming that the origin is actually in the interior of each of the two polyhedra and hence in the interior of their intersection. In this case each face

$f \in F_{\mathscr{A}} \cup F_{\mathscr{B}}$ represents a half-space of the
In other words, the constant $d(f) = 1$ for

Now, for each of the polyhedra $\mathscr{A}$ or $\mathscr{B}$
or $\mathscr{B}^{(D)}$, respectively, which we shall call it
reinterpreting the coefficients $n_1(f), n_2(f$
dinates of a corresponding vertex v_f of
$x_2(v)$, $x_3(v)$ of each vertex of $V_{\mathscr{A}}$ are re
responding face f_v of $\mathscr{A}^{(D)}$. This transform
dualization about the unit sphere with cen
l from the origin are transformed into pl
vice versa ([7, p. 233]). A similar procedu

We note that this dualization procedure
the interior of the polyhedron. In this cas
containing the origin. If the origin is not i
ties (1) would require $d(f) = 0$ or -1, and
half-spaces.

Let $V_{\mathscr{A}}^{(D)}$ and $V_{\mathscr{B}}^{D}$ be the vertex sets
seen that the convex hull of the union
intersection of $\mathscr{A}$ and $\mathscr{B}$. Hence, in order
may simply use the algorithm of Preparat
the set of vertices $V_{\mathscr{A}}^{(D)} \cup V_{\mathscr{B}}^{(D)}$ in time O
result we obtain the desired polyhedron.

Now let us assume that the given point
origin, is not in the interior of their inte
have $d(f) = 0$. In fact, these are exactly the
F' be the set of such faces. To each $f' \in F'$
form

$$n_1(f')x_1 + n_2(f')x_2 + n_3(f')x_3 \geqslant 0,$$

obtained from (1) by replacing $d(f')$ by 0.
strictly satisfy all inequalities of type (1)
equality. Such a point x exists if and only
can be satisfied strictly by some point. To
we first fix $x_3 = 1$ and write the strict form

$$n_1(f')x_1 + n_2(f')x_2 > -n_3(f').$$

Here, by normalizing the coefficients, we
The question of whether or not (2') ca
dimensional problem of the type we are
case. The inequalities of (2') collectively
which can be found ([1, p. 158]) in time O
of this convex set is possible: the inequaliti

$f \in F_{\mathscr{A}} \cup F_{\mathscr{B}}$ represents a half-space of the form $n_1(f)x_1 + n_2(f)x_2 + n_3(f)x_3 + 1 \geq 0$. In other words, the constant $d(f) = 1$ for all faces f.

Now, for each of the polyhedra $\mathscr{A}$ or $\mathscr{B}$ there is a corresponding polyhedron $\mathscr{A}^{(D)}$ or $\mathscr{B}^{(D)}$, respectively, which we shall call its *dual*. The dual $\mathscr{A}^{(D)}$ of $\mathscr{A}$ is obtained by reinterpreting the coefficients $n_1(f)$, $n_2(f)$, $n_3(f)$ of each face $f \in F_{\mathscr{A}}$ as the coordinates of a corresponding vertex v_f of $\mathscr{A}^{(D)}$. Conversely, the coordinates $x_1(v)$, $x_2(v)$, $x_3(v)$ of each vertex of $V_{\mathscr{A}}$ are reinterpreted as the coefficients of a corresponding face f_v of $\mathscr{A}^{(D)}$. This transformation may be regarded as a conventional dualization about the unit sphere with center at the origin, where points at distance l from the origin are transformed into planes at distance $1/l$ from the origin and *vice versa* ([7, p. 233]). A similar procedure allow us to form the dual of $\mathscr{B}$.

We note that this dualization procedure is only possible because the origin is in the interior of the polyhedron. In this case, the dual is also a convex polyhedron containing the origin. If the origin is not in the interior, then some of the inequalities (1) would require $d(f) = 0$ or -1, and we have not defined dual points for such half-spaces.

Let $V_{\mathscr{A}}^{(D)}$ and $V_{\mathscr{B}}^{D}$ be the vertex sets of $\mathscr{A}^{(D)}$ and $\mathscr{B}^{(D)}$ respectively. It is easily seen that the convex hull of the union of $\mathscr{A}^{(D)}$ and $\mathscr{B}^{(D)}$ is the dual of the intersection of $\mathscr{A}$ and $\mathscr{B}$. Hence, in order to find the intersection of $\mathscr{A}$ and $\mathscr{B}$ we may simply use the algorithm of Preparata and Hong [1] to find the convex hull of the set of vertices $V_{\mathscr{A}}^{(D)} \cup V_{\mathscr{B}}^{(D)}$ in time $O(n \log n)$, and upon taking the dual of the result we obtain the desired polyhedron.

Now let us assume that the given point in the intersection of $\mathscr{A}$ and $\mathscr{B}$, i.e., the origin, is not in the interior of their intersection. Then certain faces $f \in F_{\mathscr{A}} \cup F_{\mathscr{B}}$ have $d(f) = 0$. In fact, these are exactly the faces which pass through the origin. Let F' be the set of such faces. To each $f' \in F'$ there is a corresponding inequality of the form

$$n_1(f')x_1 + n_2(f')x_2 + n_3(f')x_3 \geq 0, \tag{2}$$

obtained from (1) by replacing $d(f')$ by 0. A point x in the interior of $\mathscr{A} \cap \mathscr{B}$ must strictly satisfy all inequalities of type (1) with $f \in F_{\mathscr{A}} \cup F_{\mathscr{B}}$, that is, none can be an equality. Such a point x exists if and only if all inequalities of type (2) with $f' \in F'$ can be satisfied strictly by some point. To determine whether there is such a point we first fix $x_3 = 1$ and write the strict form of (2) as

$$n_1(f')x_1 + n_2(f')x_2 > -n_3(f'). \tag{2'}$$

Here, by normalizing the coefficients, we can take $-n_3(f')$ as either 1, 0, or -1.

The question of whether or not (2') can be satisfied for every $f' \in F'$ is a two-dimensional problem of the type we are solving here for the three-dimensional case. The inequalities of (2') collectively represent a two-dimensional convex set which can be found ([1, p. 158]) in time $O(n \log n)$. Actually, a faster computation of this convex set is possible: the inequalities of (2') can be partitioned into two sets,

depending upon which of the two polyhedra they pertain to; each such set corresponds to a polygon in the plane $x_3 = 1$, and the desired convex set is the intersection of the two polygons. It is known that finding these polygons and their intersection runs in time at most proportional to n[1].

If no solution is found in the above case, the case $x_3 = -1$ must also be tried. This problem is similar to the previous one except that $-n_3(f')$ is replaced by $n_3(f')$ in all the inequalities (2').

Let us suppose that we have been able to strictly satisfy all inequalities of the type (2) with $x_3 = 1$ or -1 using the above method. Clearly, they will remain satisfied if the vector x is multiplied by a positive scalar. To strictly satisfy all the remaining inequalities in (1) whose right hand sides are all 1, we simply choose such a scalar which makes all the left hand sides less than 1. The resulting point is in the interior of $\mathscr{A} \cap \mathscr{B}$.

If it is impossible to strictly satisfy all the inequalities of (2), then any one which cannot be strictly satisfied, say

$$n_1(f'')x_1 + n_2(f'')x_2 + n_3(f'')x_3 = 0$$

represents a plane through the origin which contains the intersection of $\mathscr{A} \cap \mathscr{B}$. Thus, the intersection of $\mathscr{A}$ and $\mathscr{B}$ may be found entirely within this plane. This problem is analogous to the one discussed before and can be solved, as we saw, in time $O(n)$.

4. Finding a point in the intersection of two polyhedra

In the preceding section we have shown that the intersection of two convex polyhedra can be obtained when a point in the intersection is known. Thus, if the intersection is nonempty, all that is needed is to find one such point. The objective of this section is the implementation of this task.

Given a convex polyhedron $\mathscr{A}$, a plane is called a *plane of support* of $\mathscr{A}$ if it has at least one point in common with $\mathscr{A}$ and all interior points of $\mathscr{A}$ lie on one side of the plane. Hereafter we shall only consider planes of support parallel to the x_3-axis and briefly refer to them as *vertical*. The intersection of $\mathscr{A}$ with its vertical planes of support is, in general, an annular region $R(\mathscr{A})$ of the surface $\mathscr{A}$ which, in the absence of degeneracies, reduces to a cycle of edges. The projection of $R(\mathscr{A})$ on the (x_1, x_2) plane is a convex polygon $\mathscr{A}^*$ (Fig. 2), which is the convex hull of the projections of the points of $\mathscr{A}$ on this plane.

The region $R(\mathscr{A})$ is easily obtained from the DCEL description of $\mathscr{A}$ as follows. For any face f_i of $\mathscr{A}$, the *normal* to f_i is the vector $(n_1(f_i), n_2(f_i), n_3(f_i))$. It is perpendicular to f_i and points toward the interior of $\mathscr{A}$. Given any edge e of $\mathscr{A}$, let f_i and f_j be its adjacent faces. Then $e \in R(\mathscr{A})$ if and only if

$$n_3(f_i) \cdot n_3(f_j) \leq 0. \tag{3}$$

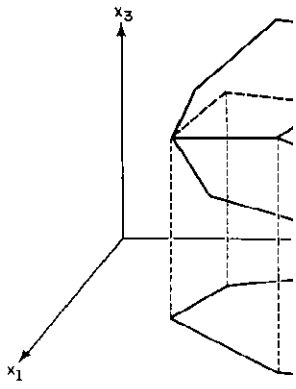


Fig. 2. A convex polyhedron $\mathscr{A}$, the annular

Therefore, we begin by scanning the edg
belongs to $R(\mathscr{A})$, by verifying condition (
vertices of e, call it v. Among the edges i
new edges, different from e, which belo
applying the VERTEX procedure descri
advance in the construction of $R(\mathscr{A})$, whi
ing the initial edge e. Once $R(\mathscr{A})$ has bee
of these operations can be carried out in
the vertices of $\mathscr{A}$. Thus we have the first s
and $\mathscr{B}$ with $|V_{\mathscr{A}}| + |V_{\mathscr{B}}| = n$:

Step 1. Find $\mathscr{A}^*$ and $\mathscr{B}^*$. (This step run

Step 2. Using Shamos–Hoey's polygon
section of $\mathscr{A}^*$ and $\mathscr{B}^*$. If the intersection
Else let p^* be a point in the intersectio
$O(n)$, according to [1].)

Under the projection of $\mathscr{A}$ to $\mathscr{A}^*$, $p^* =$
a vertical segment of $\mathscr{A}$ which reduces to
the preimage of p^* in $\mathscr{A}$ is easily found in
we determine the x_3-coordinate of the
projects to p^*; specifically, this x_3-coordi

$$\alpha(f) = -(n_1(f)x_1(p^*) + n_2(f)x_2(p$$

Let

$$\alpha' = \min_{n_3(f)<0} \alpha(f) \quad \text{and} \quad \alpha'' = \max_{n_3(f)}$$

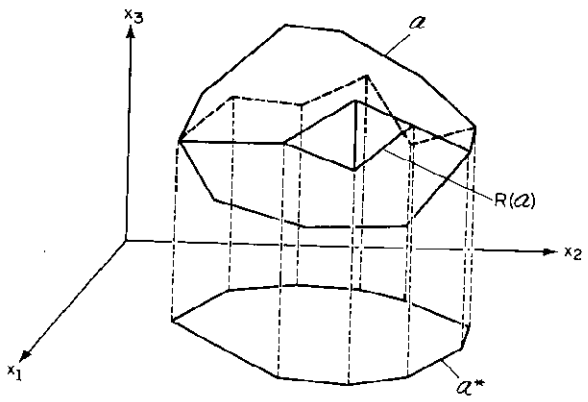


Fig. 2. A convex polyhedron $\mathscr{A}$, the annular region $R(\mathscr{A})$ and the projection polygon $\mathscr{A}^*$.

Therefore, we begin by scanning the edge set of $\mathscr{A}$ until we find an edge e which belongs to $R(\mathscr{A})$, by verifying condition (3). At this point, we select one of the two vertices of e, call it v. Among the edges incident on v there are either one or two new edges, different from e, which belong to $R(\mathscr{A})$ and can be easily found by applying the VERTEX procedure described earlier, to the DCEL. Thus we can advance in the construction of $R(\mathscr{A})$, which will be completed upon re-encountering the initial edge e. Once $R(\mathscr{A})$ has been computed, $\mathscr{A}^*$ is trivially obtained. All of these operations can be carried out in time proportional to the number $|V_{\mathscr{A}}|$ of the vertices of $\mathscr{A}$. Thus we have the first steps of the algorithm, given polyhedra $\mathscr{A}$ and $\mathscr{B}$ with $|V_{\mathscr{A}}|+|V_{\mathscr{B}}|=n$:

Step 1. Find $\mathscr{A}^*$ and $\mathscr{B}^*$. (This step runs in time $O(n)$.)

Step 2. Using Shamos–Hoey's polygon intersection algorithm [1], find the intersection of $\mathscr{A}^*$ and $\mathscr{B}^*$. If the intersection is empty, halt, for $\mathscr{A}\cap\mathscr{B}$ is also empty. Else let p^* be a point in the intersection of $\mathscr{A}^*$ and $\mathscr{B}^*$. (This step runs in time $O(n)$, according to [1].)

Under the projection of $\mathscr{A}$ to $\mathscr{A}^*$, $p^*=(x_1(p^*), x_2(p^*))$ is in general the image of a vertical segment of $\mathscr{A}$ which reduces to a single point in some cases. In any case, the preimage of p^* in $\mathscr{A}$ is easily found in time $O(n)$ as follows. For each face $f\in F_{\mathscr{A}}$ we determine the x_3-coordinate of the point on the corresponding plane which projects to p^*; specifically, this x_3-coordinate is

$$\alpha(f)=-(n_1(f)x_1(p^*)+n_2(f)x_2(p^*)+d(f))/n_3(f).$$

Let

$$\alpha'=\min_{n_3(f)<0}\alpha(f) \quad \text{and} \quad \alpha''=\max_{n_3(f)>0}\alpha(f).$$

Then α' and α'', with $\alpha' \geq \alpha''$, are the x_3-coordinates of the extremes of the segment which is the preimage of p^* in $\mathscr{A}$; we similarly define β' and β'', with $\beta' \geq \beta''$, for the analogous segment in $\mathscr{B}$. If the two segments overlap, then any point in their common portion also belongs to the nonempty intersection of $\mathscr{A}$ and $\mathscr{B}$. Otherwise assume, without loss of generality, that $\alpha'' > \beta'$. Then we define the *near-sides* of $\mathscr{A}$ and $\mathscr{B}$ to be the sets of faces $\{f_i | f_i$ is a face of $\mathscr{A}, n_3(f_i) < 0\}$ and $\{g_i | g_i$ is a face of $\mathscr{B}, n_3(g_i) > 0\}$, respectively. Clearly both near-sides are obtained in time $O(|V_{\mathscr{A}}| + |V_{\mathscr{B}}|) = O(n)$ by traversing, in a straightforward manner, the DCEL descriptions of $\mathscr{A}$ and $\mathscr{B}$. By projecting the near-sides of $\mathscr{A}$ and $\mathscr{B}$ on the (x_1, x_2)-plane we obtain two planar straight-line graphs (PSLG's) $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$, with respective vertex sets $V'_{\mathscr{A}}$ and $V'_{\mathscr{B}}$. Thus we have:

Step 3. If the pre-images of p^* in $\mathscr{A}$ and $\mathscr{B}$ under an x_3-projection have a common intersection, then halt, for any point in this intersection is internal to $\mathscr{A} \cap \mathscr{B}$. Otherwise obtain $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$. (The pre-images of p^* are found in constant time; $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$ are found in time $O(n)$.)

Let $\mathscr{D}$ be the closed domain contained in the intersection of $\mathscr{A}^*$ and $\mathscr{B}^*$. For each point $u \in \mathscr{D}$ it is convenient to define the function $\delta(u)$, called x_3-distance, as follows. If $\alpha(u)$ and $\beta(u)$ are the x_3-coordinates of the points on the faces of the near-sides of $\mathscr{A}$ and $\mathscr{B}$, respectively, which both project to u, then

$$\delta(u) = \alpha(u) - \beta(u).$$

Let us now analyze the function $\delta(u)$ defined on $\mathscr{D}$. Imagine superimposing $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$ to create a new vertex, conveniently called a *pseudo-vertex*, at the intersection of each edge of $G_{\mathscr{A}}$ with an edge of $G_{\mathscr{B}}$. Denoting by V^* the set of pseudo-vertices thus obtained, we can define a new PSLG G^* with vertex set $V'_{\mathscr{A}} \cup V'_{\mathscr{B}} \cup V^*$. The vertices of $V'_{\mathscr{A}}$ and $V'_{\mathscr{B}}$ will be called *true vertices*. Thus the domain $\mathscr{D}$ is subdivided into regions by G^*. Notice that inside any region of $G_{\mathscr{A}}$ the function $\alpha(v)$ is linear in the (x_1, x_2) coordinates at v; similarly, for the function $\beta(v)$ inside any region of $G_{\mathscr{B}}$. Thus in any region induced by G^* in $\mathscr{D}$, the function $\delta(v) = \alpha(v) - \beta(v)$ is linear in $x_1(v)$ and $x_2(v)$. Moreover, $\alpha(v)$ is convex-downward and $\beta(v)$ is convex-upward; it follows that $\delta(v)$ is a convex-downward function. We conclude that the minimum of δ occurs at a vertex of G^*. Notice that $|V^*|$, and hence $|V'_{\mathscr{A}} \cup V'_{\mathscr{B}} \cup V^*|$, could be $O(n^2)$: in fact, it is not hard to construct two planar graphs, each with ν vertices, so that, when superimposed, $(\nu - 1)^2$ intersections of edges are obtained.

Since, by hypothesis, $\alpha(p^*) = \alpha''$ and $\beta(p^*) = \beta'$, we conclude that $\delta(p^*) = \alpha'' - \beta' > 0$. It follows that the intersection of $\mathscr{A}$ and $\mathscr{B}$ is nonempty if and only if, for some $v \in \mathscr{D}$, $\delta(v) \leq 0$. Therefore, either we find one such point, or show that $\min_{v \in \mathscr{D}} \delta(v) > 0$.

To this end, we begin by evaluating δ at true vertices of G^* in $\mathscr{D}$. This is easily done if, for each $a \in V'_{\mathscr{A}}$ we determine a region $r(a)$ of $G_{\mathscr{B}}$ to which a belongs. If

$r(a)$ is not the infinite region of the plane
$r(a)$ corresponds to a unique face of the pc
Similarly, we can compute $\delta(b)$ for each
has been called the location of point a in
and could be done one point at a time.
recently developed [6] for collectively loc
this technique the members of $V'_{\mathscr{A}}$ can
subdivision induced by $G_{\mathscr{B}}$ in time $O((c|V$
and, reciprocally, the members of $V'_{\mathscr{B}}$ ca
subdivision induced by $G_{\mathscr{A}}$ in time $O((|V$
total time $O((|V'_{\mathscr{A}}| + |V'_{\mathscr{B}}|) \log (|V'_{\mathscr{A}}| \cdot |V$
$O(n \log n)$ all the true vertices can be loc
located in $G_{\mathscr{B}}$, the x_3-coordinates of its
constant time. This is summarized as follc

Step 4. Locate each true vertex of $G_{\mathscr{A}}$
and *vice versa*. (This can be done in time
there are no true vertices in $\mathscr{D}$ go to Step
G^*. (This can be done in additional time

Suppose at first that there are true vert
vertex v (say, $v \in V'_{\mathscr{A}}$) we have $\delta(v) \leq 0$.
near-side of $\mathscr{A}$ in a vertex a and the ne
$\alpha(v) = x_3(a) \leq x_3(b) = \beta(v)$. Thus, we have
point v for which $\alpha(v) \leq \beta(v)$. Consider
containing the points p^* and v. The inte
hedra $\mathscr{A}$ and $\mathscr{B}$ is shown in Fig. 3 (where
By convexity, the segments ap' and bp'' a
tively, and so their point of intersection
The coordinates of q are thus obtained b

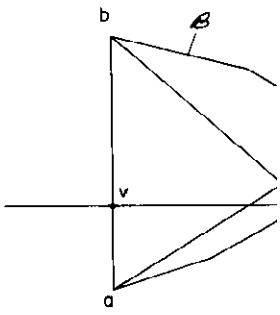


Fig. 3. Finding a point in the int

$r(a)$ is not the infinite region of the plane in the subdivision induced by $G_{\mathscr{B}}$, then $r(a)$ corresponds to a unique face of the polyhedron $\mathscr{B}$, and $\delta(a)$ is easily computed. Similarly, we can compute $\delta(b)$ for each $b \in V'_{\mathscr{B}}$ in $\mathscr{D}$. The determination of $r(a)$ has been called the location of point a in the planar subdivision induced by $G_{\mathscr{B}}$ [5] and could be done one point at a time. However, a faster algorithm has been recently developed [6] for collectively locating all the points of a set. According to this technique the members of $V'_{\mathscr{A}}$ can all be located in regions of the planar subdivision induced by $G_{\mathscr{B}}$ in time $O((c|V'_{\mathscr{A}}|+|V'_{\mathscr{B}}|)\log|V'_{\mathscr{A}}|\cdot|V'_{\mathscr{B}}|)$, for constant c, and, reciprocally, the members of $V'_{\mathscr{B}}$ can all be located in regions of the planar subdivision induced by $G_{\mathscr{A}}$ in time $O((|V'_{\mathscr{A}}|+c|V'_{\mathscr{B}}|)\log|V'_{\mathscr{A}}|\cdot|V'_{\mathscr{B}}|)$. Therefore in total time $O((|V'_{\mathscr{A}}|+|V'_{\mathscr{B}}|)\log(|V'_{\mathscr{A}}|\cdot|V'_{\mathscr{B}}|))=O((|V_{\mathscr{A}}|+|V_{\mathscr{B}}|)\log(|V_{\mathscr{A}}|\cdot|V_{\mathscr{B}}|))=O(n\log n)$ all the true vertices can be located. Once a vertex, say, of $V'_{\mathscr{A}}$ has been located in $G_{\mathscr{B}}$, the x_3-coordinates of its preimages in $\mathscr{A}$ and $\mathscr{B}$ are obtained in constant time. This is summarized as follows:

Step 4. Locate each true vertex of $G_{\mathscr{A}}$ in the planar subdivision induced by $G_{\mathscr{B}}$ and *vice versa*. (This can be done in time $O(n\log n)$ using the algorithm of [6].) If there are no true vertices in $\mathscr{D}$ go to Step 7. Else evaluate δ at each true vertex of G^*. (This can be done in additional time $O(n)$.)

Suppose at first that there are true vertices in $\mathscr{D}$, and assume that for some true vertex v (say, $v \in V'_{\mathscr{A}}$) we have $\delta(v)\leq 0$. The vertical line through v intercepts the near-side of $\mathscr{A}$ in a vertex a and the nearside of $\mathscr{B}$ in a point b, and obviously $\alpha(v)=x_3(a)\leq x_3(b)=\beta(v)$. Thus, we have a point p^* for which $\alpha(p^*)>\beta(p^*)$ and a point v for which $\alpha(v)\leq\beta(v)$. Consider now the plane, parallel to the x_3-axis and containing the points p^* and v. The intersection of this plane with the two polyhedra $\mathscr{A}$ and $\mathscr{B}$ is shown in Fig. 3 (where the points p' and p'' have been defined). By convexity, the segments $\overline{ap'}$ and $\overline{bp''}$ are entirely contained in $\mathscr{A}$ and $\mathscr{B}$ respectively, and so their point of intersection q belongs to the intersection of $\mathscr{A}$ and $\mathscr{B}$. The coordinates of q are thus obtained by straightforward calculations.

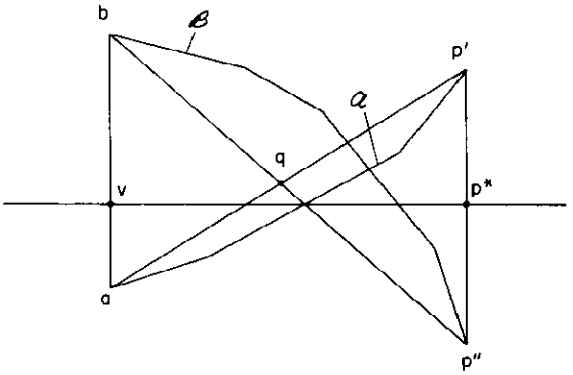


Fig. 3. Finding a point in the intersection when $\delta(p^*)>0$ and $\delta(v)\leq 0$.

Assume next that $\delta(v)>0$ for all true vertices $v \in V'_{\mathscr{A}} \cup V'_{\mathscr{B}}$, and let v^* be a true vertex such that $\delta(v^*)=\min\{\delta(v) \mid v \in V'_{\mathscr{A}} \cup V'_{\mathscr{B}}\}$. We cyclically test each of the edges of $\mathscr{A}$ incident upon v^* to determine whether the function δ decreases as one moves along the edge away from v^*. If it fails to decrease for all edges incident upon v^*, then v^* is an absolute minimum of the function δ. Since $\delta(v^*)>0$, the polyhedra $\mathscr{A}$ and $\mathscr{B}$ do not intersect.

Step 5. Obtain v^*. If v^* is an absolute minimum and $\delta(v^*)>0$, halt, for $\mathscr{A} \cap \mathscr{B} = \mathscr{V}$; if $\delta(v^*)>0$ but v^* is not an absolute minimum, go to Step 6; if $\delta(v^*)\leqslant 0$, then there is a point $q \in \mathscr{A} \cap \mathscr{B}$, obtained as the intersection of the diagonals of the trapezoid formed by the x_3-projection pre-images of p^* and v^* in $\mathscr{A}$ and $\mathscr{B}$. (All of this work can be done in time $O(n)$.)

The remaining case is when $\delta(v^*)>0$ but v^* is not an absolute minimum. Then δ decreases as one moves along at least one of the edges – call it e – incident upon v^*, and on this edge we locate a pseudo-vertex p. One such pseudo-vertex must exist, for otherwise the minimality of $\delta(v^*)$ would be contradicted. Let $r(v^*)$ be the region of $G_{\mathscr{B}}$ to which v^* belongs (known from Step 4); to locate pseudo-vertex p, we cyclically test each edge of $r(v^*)$ in turn and find the one which intersects e.[4] Clearly $\delta(p)<\delta(v^*)$.

Step 6. Locate a pseudo-vertex p adjacent to v^*, such that $\delta(p)<\delta(v^*)$. If $\delta(p)\leqslant 0$, then, since $\delta(v^*)>0$, there is a point $q \in \mathscr{A} \cap \mathscr{B}$, which may be found as in Step 5; otherwise go to Step 7.

We must now consider two cases. The first is when there are no true vertices in $\mathscr{D}$ (Step 4); then the boundaries of $\mathscr{A}^*$ and $\mathscr{B}^*$ must intersect, so the point p^* may be chosen at an intersection of these boundaries and is therefore a pseudo-vertex. The second case is when $\delta(p)>0$ (Step 6). Both these cases are treated by using an algorithm, called the *wandering algorithm*, which wanders among the pseudo-vertices of G^* and which uses at most $O(n)$ time. Thus we have:

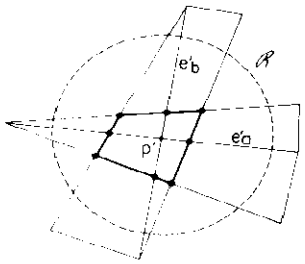
Step 7. If the test of Step 4 fails use p^*, while if the test of Step 6 fails use p, as the starting point of the wandering algorithm (to be described below), either to find a pseudo-vertex $\bar{p}$ such that $\delta(\bar{p})\leqslant 0$, to which the method of Step 5 can be applied, or find a pseudo-vertex p_m such that $\delta(p_m)=\min_{v \in V^*}\delta(v)$. (We shall show below that the wandering algorithm runs in time $O(n)$.)

Before describing the wandering algorithm, we observe that the starting point of it is a pseudo-vertex, either p or p^*, which has a smaller value of δ than any true vertex. If we imagine, for purposes of proof, a contour line of δ passing through p or p^* we enclose a region $\mathscr{R} \subseteq \mathscr{D}$ which contains a pseudo-vertex p_m having minimum $\delta(p_m)$. We note that $\mathscr{R}$ must be convex. Also let $E'_{\mathscr{A}}$ and $E'_{\mathscr{B}}$ be the sets

[4] If v^* happens to belong to more than one region of $G_{\mathscr{B}}$, then the edges of all such regions may have to be tested to find the unique one which intersects e. In any case, the number of such tests is $O(n)$.

of edges of $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$ respectively whi
$\mathscr{R}$, each edge in $E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$ must separate
in $E'_{\mathscr{A}}$ can intersect in $\mathscr{R}$, nor can two e
downward as one travels along any edg
somewhere in $\mathscr{R}$, because the boundary
a point on an edge $e \in E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$ where
of the edge e. It is unique except in degen
be called the *minimum value of the edge*

We assume momentarily that the fac
they are not, that we shall triangulate
angulated polyhedra the number of face
vertices and the number of edges remai
$O(n)$. Each pseudo-vertex p' in $\mathscr{R}$ is t
$e'_b \in E'_{\mathscr{B}}$ and is therefore shared by fou
regions is referred to as the *crown* of p'
reached from p' without crossing any
triangular faces of $G_{\mathscr{A}}$, whose union is a
for e'_b. Thus the crown is the intersection
crown boundary contains either 8, or 1
respectively). The fact that the num
consequence of the hypothesis that the



(a)

Fig. 4. Illustration of the possible ca

Given any pseudo-vertex p' as the inter
bordering these two edges can be obt
connected edge lists describing $\mathscr{A}$ and
available, the pseudo-vertices in the crow
a constant, and so can their values of δ.

Advancing step of the wandering algor
pseudo-vertex p' to a pseudo-vertex p
among all pseudo-vertices in the crown

of edges of $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$ respectively which intersect $\mathscr{R}$. Since no true vertices lie in $\mathscr{R}$, each edge in $E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$ must separate $\mathscr{R}$ into two convex regions. No two edges in $E'_{\mathscr{A}}$ can intersect in $\mathscr{R}$, nor can two edges in $E'_{\mathscr{B}}$. Also, the function δ is convex downward as one travels along any edge of $E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$ and its minimum must lie somewhere in $\mathscr{R}$, because the boundary of $\mathscr{R}$ is a contour line for δ. We shall call a point on an edge $e \in E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$ where δ has a minimum value, a *minimum point of the edge e*. It is unique except in degenerate cases. The value of δ at this point will be called the *minimum value of the edge e* and denoted by min(e).

We assume momentarily that the faces of both polyhedra are triangles and, if they are not, that we shall triangulate both polyhedra. Notice that in the triangulated polyhedra the number of faces remains less than twice the number n of vertices and the number of edges remains less than three times n, so they remain $O(n)$. Each pseudo-vertex p' in $\mathscr{R}$ is the intersection of two edges $e'_a \in E'_{\mathscr{A}}$ and $e'_b \in E'_{\mathscr{B}}$ and is therefore shared by four regions in G^*; the union of these four regions is referred to as the *crown* of p' and is the locus of the points which can be reached from p' without crossing any edge. Notice that e'_a is shared by two triangular faces of $G_{\mathscr{A}}$, whose union is a quadrilateral region; a similar remark holds for e'_b. Thus the crown is the intersection of these two quadrilateral regions, and the crown boundary contains either 8, or 10, or 12 pseudo-vertices (see Fig. 4a, b, c, respectively). The fact that the number of crown vertices is bounded is a consequence of the hypothesis that the polyhedra have been triangulated.

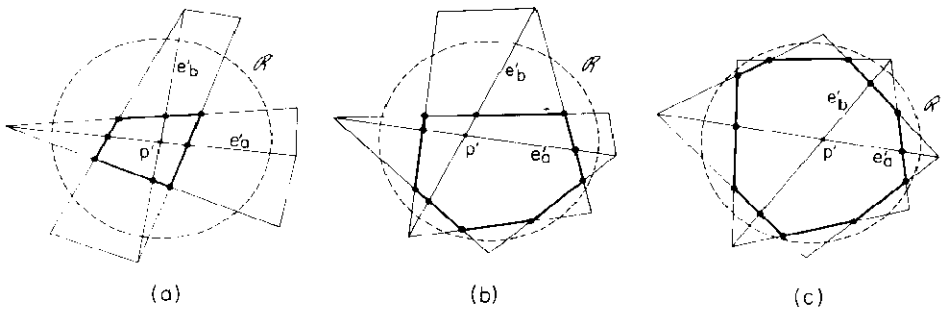


Fig. 4. Illustration of the possible cases for the crown of a pseudo-vertex.

Given any pseudo-vertex p' as the intersection of e'_a and e'_b, the pairs of triangles bordering these two edges can be obtained in constant time from the doubly-connected edge lists describing $\mathscr{A}$ and $\mathscr{B}$ respectively. Once these triangles are available, the pseudo-vertices in the crown can also be obtained in time bounded by a constant, and so can their values of δ. We now give the

Advancing step of the wandering algorithm: A pointer is moved from the current pseudo-vertex p' to a pseudo-vertex p'' which attains the minimum value of δ among all pseudo-vertices in the crown of p'.

Of course, the step is voided and the algorithm terminates if p' attains the minimum value of δ along the edges e'_a and e'_b intersecting in p': in this case if δ is positive the two polyhedra do not intersect (case (iii) in Section 5). In the other case (δ decreases either along e'_a or e'_b) the advancing step is effected, and in actual practice can be carried out without exploring the entire crown of p', but simply following a path of edges along which δ decreases.

An additional algorithm simplification is that, as we shall show, polyhedra $\mathscr{A}$ and $\mathscr{B}$ need not be triangulated before applying the wandering algorithm. In fact, only those faces of $G_{\mathscr{A}}$ and $G_{\mathscr{B}}$ will be triangulated which are actually traversed by the wandering algorithm. Specifically, let p', the intersection of e'_a and e'_b, be the current pseudo-vertex (see Fig. 5). Referring for simplicity only to polyhedron $\mathscr{A}$,

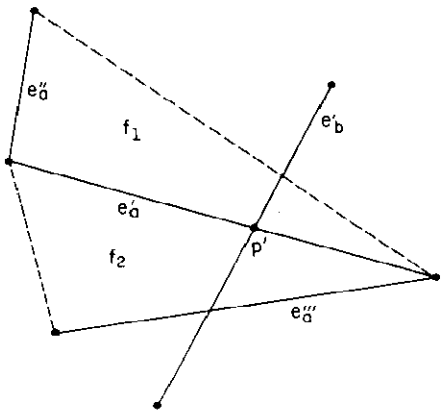


Fig. 5. Partial triangulation of $\mathscr{A}$.

let f_1 and f_2 be the two faces of $G_{\mathscr{A}}$ sharing e'_a. In the doubly connected edge list of $G_{\mathscr{A}}$ we can obtain in constant time the edges e''_a and e'''_a which follow e'_a in the edge-sequences of f_1 and f_2, respectively. If f_1 is not a triangle, we connect the non-overlapping extremes of e''_a and e'_a, and we do likewise for f_2. The introduction of any such new edge in the doubly-connected edge list requires the modification of two pointers and the use of two other cells for construction of the appropriate record. All this can also be done in constant time. We conjecture that this insertion is not really necessary, but the present proof on the time performance of the algorithm depends upon it.

Since the wandering algorithm moves from p' to p'' only if $\delta(p'')<\delta(p')$, it is obvious that the algorithm will terminate at a point p_m such that $\delta(p_m)$ is the minimum value of δ for all pseudo-vertices in $\mathscr{R}$. Even though the total number of pseudo-vertices in $\mathscr{R}$ could be $O(n^2)$, we shall now prove that the number of advancing steps is at most $O(n)$.

Recall that for an edge e in either $G_{\mathscr{A}}$ of δ on e. Let pseudo-vertex p' be the in define $m(p')=\max(\min(e'_a),\min(e'_b))$.

Lemma 4.1. *Let p' be a pseudo-vertex i* $\delta(p_m)$.

Proof. Let us assume the contrary and ol represent minimum points on e'_a and e'_b assumption, $\delta(a')=\delta(b')=\delta(p_m)$ and h line segment l between a' and b' alsc pseudo-vertex closest to a' in the portio p' coincide). The line segment l crosses the value of δ is linear within this regio at an interior point, it must have this v $\delta(a_1)=\delta(p_m)$, contradicting our assumpt p' on e'_a. This proves $\delta(p')=\delta(p_m)$ □

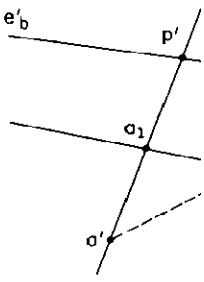


Fig. 6. Illustration for the proo

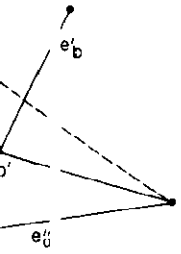
Assuming now that $\delta(p')>\delta(p_m)$, we the wandering algorithm is applied at p'

Lemma 4.2. $m(p'')<m(p')$.

Proof. We distinguish two cases:

(1) *p' and p'' do not belong to the sam* e'_b and let p'' be the intersection of e pseudo-vertex in $\mathscr{R}$ defined above. W

algo thm terminates if p' attains the
e'_b i tersecting in p': in this case if δ is
case iii) in Section 5). In the other case
vanc ng step is effected, and in actual
ng tl e entire crown of p', but simply
reas s.
hat, s we shall show, polyhedra $\mathscr{A}$ and
the vandering algorithm. In fact, only
ted v hich are actually traversed by the
he i itersection of e'_a and e'_b, be the
g fo simplicity only to polyhedron $\mathscr{A}$,



gulati n of $\mathscr{A}$.

e'_a. I the doubly connected edge list of
ges e''_a and e'''_a which follow e'_a in the
If f_1 is not a triangle, we connect the
we c o likewise for f_2. The introduction
ed ed ge list requires the modification of
ls fo construction of the appropriate
time We conjecture that this insertion
roof on the time performance of the

om p' to p'' only if $\delta(p'') < \delta(p')$, it is
at a point p_m such that $\delta(p_m)$ is the
in $\mathscr{R}$ Even though the total number of
shal now prove that the number of

Recall that for an edge e in either $G_{\mathscr{A}}$ or $G_{\mathscr{B}}$, min(e) denotes the minimum value of δ on e. Let pseudo-vertex p' be the intersection of $e'_a \in E'_{\mathscr{A}}$ and $e'_b \in E'_{\mathscr{B}}$; we now define $m(p') = \max(\min(e'_a), \min(e'_b))$. Clearly $\delta(p') \geq m(p')$.

Lemma 4.1. *Let p' be a pseudo-vertex in $\mathscr{R}$; if $m(p') = \delta(p_m)$, then $\delta(p') = m(p') = \delta(p_m)$.*

Proof. Let us assume the contrary and obtain a contradiction. In Fig. 6 let a' and b' represent minimum points on e'_a and e'_b respectively which are nearest to p'. By our assumption, $\delta(a') = \delta(b') = \delta(p_m)$ and hence by convexity, every point along the line segment l between a' and b' also has this same δ value. Let a_1 be the pseudo-vertex closest to a' in the portion of e'_a between a' and p' (possibly, a_1 and p' coincide). The line segment l crosses a region of G^* bordering with $a'a_1$. Since the value of δ is linear within this region and it achieves the minimum value $\delta(p_m)$ at an interior point, it must have this value throughout the entire region. Hence, $\delta(a_1) = \delta(p_m)$, contradicting our assumption that a' is the nearest minimum point to p' on e'_a. This proves $\delta(p') = \delta(p_m)$ □

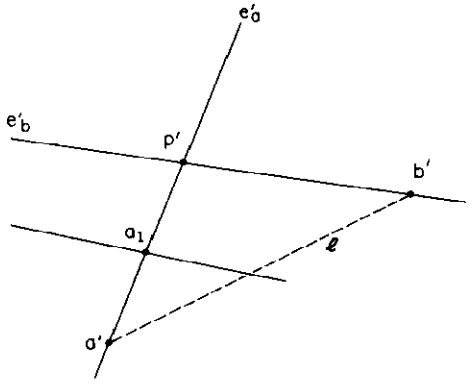


Fig. 6. Illustration for the proof of $m(p') = \delta(p_m) \Rightarrow \delta(p') = m(p')$.

Assuming now that $\delta(p') > \delta(p_m)$, we see by Lemma 1 that $m(p') > \delta(p_m)$. When the wandering algorithm is applied at p', it steps to a new pseudo-vertex p''.

Lemma 4.2. $m(p'') < m(p')$.

Proof. We distinguish two cases:

(1) *p' and p'' do not belong to the same edge.* Let p' be the intersection of e'_a and e'_b and let p'' be the intersection of e''_a and e''_b (see Fig. 7(a)). Let p_m be the pseudo-vertex in $\mathscr{R}$ defined above. We claim that a straight line from p_m to p''

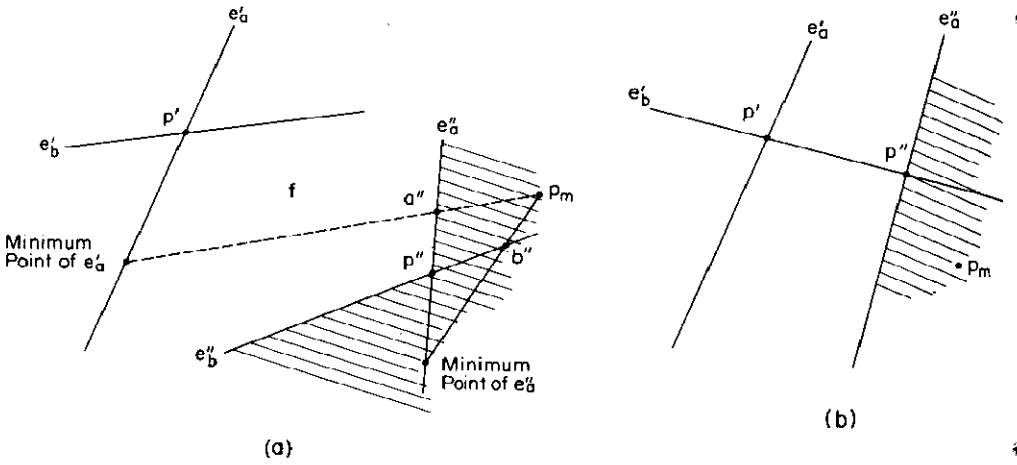


Fig. 7. Illustration of the proof that $m(p'') < m(p')$.

cannot intersect the interior of the region f of G^* to whose boundary p' and p'' belong, except at p''. In fact, if it did, any such point of intersection would, by convexity, have a value of δ as low as $\delta(p'')$. Since p'' is a minimum point of f, this would imply that all the points of f have the same value of δ, contradicting $\delta(p') > \delta(p'')$. As a consequence, either e''_a or e''_b separates p_m from p', so p_m belongs to the shaded regions in Fig. 7(a).

Assume, without loss of generality, that p', and hence e'_a, is separated from p_m by e''_a. Then, since e'_a does not cross e''_a in $\mathcal{R}$, the straight line between p_m and the minimum point of e'_a intersects e''_a in a point a''. By convexity, $\min(e'_a) \geq \delta(a'')$, with equality occurring only if $\min(e'_a) = \delta(p_m)$. Assuming equality, since we have seen that $m(p') > \delta(p_m)$ and we have $\min(e'_a) = \delta(a'') = \min(e''_a) = \delta(p_m)$, we obtain $m(p') > \min(e''_a)$. Assuming instead that $\min(e'_a) > \delta(a'')$, since by definition $m(p') \geq \min(e'_a)$ and $\delta(a'') \geq \min(e''_a)$, we also obtain $m(p') > \min(e''_a)$.

Two subcases must be considered. First, assume p', and hence e'_b, is *also* separated from p_m by e''_b. Then by an identical argument $m(p') > \min(e''_b)$, so $m(p') > m(p'') = \max(\min(e''_a), \min(e''_b))$. Second, assume it is not, as in Fig. 7(a). We now show that $\min(e''_b) \leq \min(e''_a)$ thus reaching the same conclusion. In fact, since $\delta(p'')$ is the minimum value in f, the minimum point on e''_a is either at p'' or along e''_a on the opposite side of p'' from f. A straight line drawn between this minimum point and p_m intersects e''_b at a point b'' such that $\delta(b'') \leq \min(e''_a)$, by the convexity argument used earlier. But $\delta(b'') \geq \min(e''_b)$, whence $\min(e''_a) \geq \min(e''_b)$, as claimed.

(2) *p' and p'' belong to the same edge.* Without loss of generality, let p' be the intersection of e'_a and e'_b and let p'' be the intersection of e''_a and e'_b (see Fig. 7(b)). By the convexity argument, e'_a is separated from p_m by e''_a (i.e., p_m belongs to the shaded region). As in case (1), we can show that $m(p') \geq \min(e'_a) > \min(e''_a)$. To

prove that $\min(e'_b) \leq \min(e''_a)$ we note th
otherwise p'' would not attain the minin
$\max(\min(e''_a), \min(e'_b)) = \min(e''_a) < m(p'$

Theorem 4.3. *The number of advacing st*
$O(n)$.

Proof. We have shown that as the wand
vertex p' to the next, the value of $m(p')$ c
is the minimum value of one of the e
distinct values which $m(p')$ can assume is
The number of steps taken by the algorit

Since the time taken by the wanderin
entire algorithm remains $O(n \log n)$.

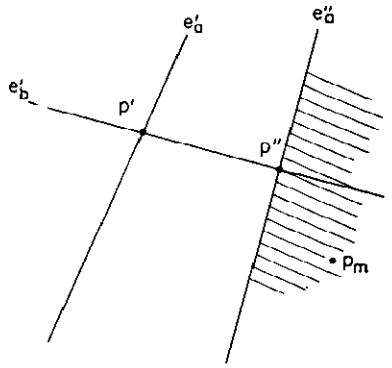
5. An application: Finding a separating

The preceding method can be used to
linear separability in three dimensions, i.
A and B are separable by means of a pla
Since two finite sets of points are line
hulls do not intersect [8], we begin by ob
sets A and B by means of the Preparata-
this task, which is completed in time O(
and $\mathcal{B}$ such that $|V_{\mathcal{A}}| + |V_{\mathcal{B}}| \leq n$. We now
in Section 4: any time the algorithm dec
construct a separating plane.

We now recall that $\mathcal{A}$ and $\mathcal{B}$ are four
already referred to in Section 4:

(i) after projecting $R(\mathcal{A})$ and $R(\mathcal{B})$ c
$\mathcal{B}^*$ are disjoint (Step 2);
(ii) after evaluating δ at all true
$\min_{v \in V_{\mathcal{A}} \cup V_{\mathcal{B}}} \delta(v) > 0$ and v^* is an
(iii) after applying the wandering algor

In case (i) it is sufficient to find a strai
plane containing l and perpendicular to t
line l can be found in time $O(n)$ by an ob
algorithm for planar convex hulls ([3, p. 9
Cases (ii) and (iii) can be handled joint
than constructing one separating plane,



(b)

prove that $\min(e'_b) \leq \min(e''_a)$ we note that the minimum point of e''_a must be p'', for otherwise p'' would not attain the minimum of δ in the crown of p'. Thus $m(p'') = \max(\min(e''_a), \min(e'_b)) = \min(e''_a) < m(p')$, as claimed. □

Theorem 4.3. *The number of advacing steps performed by the wandering algorithm is* $O(n)$.

Proof. We have shown that as the wandering algorithm moves from one pseudo-vertex p' to the next, the value of $m(p')$ decreases at each step. Each value of $m(p')$ is the minimum value of one of the edges in $E'_{\mathscr{A}} \cup E'_{\mathscr{B}}$. Hence, the number of distinct values which $m(p')$ can assume is no greater than $|E'_{\mathscr{A}}| + |E'_{\mathscr{B}}|$, which is $O(n)$. The number of steps taken by the algorithm therefore is $O(n)$. □

Since the time taken by the wandering algorithm is $O(n)$, the time taken by the entire algorithm remains $O(n \log n)$.

5. An application: Finding a separating plane

The preceding method can be used to solve efficiently the important problem of linear separability in three dimensions, i.e., testing whether two finite sets of points A and B are separable by means of a plane, and, if so, finding one such plane.

Since two finite sets of points are linearly separable if and only if their convex hulls do not intersect [8], we begin by obtaining the respective convex hulls of the sets A and B by means of the Preparata–Hong algorithm [3]. Letting $|A| + |B| = n$, this task, which is completed in time $O(n \log n)$, yields two convex polyhedra $\mathscr{A}$ and $\mathscr{B}$ such that $|V_{\mathscr{A}}| + |V_{\mathscr{B}}| \leq n$. We now apply to $\mathscr{A}$ and $\mathscr{B}$ the algorithm described in Section 4: any time the algorithm declares that $\mathscr{A}$ and $\mathscr{B}$ do not intersect, we construct a separating plane.

We now recall that $\mathscr{A}$ and $\mathscr{B}$ are found to be disjoint in three exclusive cases, already referred to in Section 4:

(i) after projecting $R(\mathscr{A})$ and $R(\mathscr{B})$ on the plane (x_1, x_2), the polygons $\mathscr{A}^*$ and $\mathscr{B}^*$ are disjoint (Step 2);

(ii) after evaluating δ at all true vertices of G^* we find that $\delta(v^*) = \min_{v \in V_{\mathscr{A}} \cup V_{\mathscr{B}}} \delta(v) > 0$ and v^* is an absolute minimum (Step 5);

(iii) after applying the wandering algorithm we find that $\delta(p_m) > 0$.

In case (i) it is sufficient to find a straight line l separating $\mathscr{A}^*$ and $\mathscr{B}^*$, since a plane containing l and perpendicular to the plane (x_1, x_2) separates $\mathscr{A}$ and $\mathscr{B}$. The line l can be found in time $O(n)$ by an obvious modification of the Preparata–Hong algorithm for planar convex hulls ([3, p. 90]).

Cases (ii) and (iii) can be handled jointly by the following considerations. Rather than constructing one separating plane, we construct a locus of separating planes

and make a selection in this locus. Let u be the point at which the algorithm terminates with the result that $\mathscr{A}$ and $\mathscr{B}$ do not intersect; obviously, either $u = v^*$ or $u = p_m$. Also let u' and u'' be the pre-images of u (with respect to x_3-projection) in $\mathscr{A}$ and $\mathscr{B}$, respectively. Assume at first that $u = v^*$ and, without loss of generality, let u' be a vertex (in $V_{\mathscr{A}}$). Consider the cycle F of the faces sharing u'; for each $f \in F$, imagine applying the vector $(n_1(f), n_2(f), n_3(f)) = \boldsymbol{n}(f)$ to the origin; recall that $\boldsymbol{n}(f)$ is normal to f and pointing toward the interior of $\mathscr{A}$. Then the set of directions $\{\boldsymbol{n}(f) \mid f \in F\}$ defines a convex cone $C_{\mathscr{A}}$ such that any direction internal to it is normal to a supporting plane of $\mathscr{A}$. Notice now that, when $u = p_m$, point u' belongs to some edge e_a of $\mathscr{A}$ and $C_{\mathscr{A}}$ degenerates into a plane wedge delimited by the normals to the two faces of $\mathscr{A}$ which share e_a.

For u'' the convex cone $C_{\mathscr{B}}$ is analogously defined, with the only modification that the directions of the vectors $\boldsymbol{n}(f)$ are reversed. The cone can assume the following forms: if $u = v^*$, then $C_{\mathscr{B}}$ is either nondegenerate, or a plane wedge, or a half-line, depending upon whether u'' in $\mathscr{B}$ is either a vertex, or a point in an edge, or a point in a face, respectively; if $u = p_m$, then $C_{\mathscr{B}}$ is a plane wedge.

The solution to our problem is $C_{\mathscr{A}} \cap C_{\mathscr{B}}$. Notice, however, that this intersection consists of a single ray in the following two cases:

(1) $u = p_m$, in which case the ray is the common normal to the edges which contain u' and u'' in $\mathscr{A}$ and $\mathscr{B}$, respectively;

(2) $u = v^*$ and u'' is a point in a face of $\mathscr{B}$, in which case the ray is the normal to this face. In the remaining cases ($u = v^*$, u' is a vertex, and u'' is either a vertex or a point in an edge) $C_{\mathscr{A}}$ is nondegenerate, hence it intersects either the plane $x_3 = 1$ or the plane $x_3 = -1$ in a polygon. Suppose without loss of generality, that $C_{\mathscr{A}}$ intersects the plane $x_3 = 1$: the polygon can be found in time $O(n)$ by scanning the cycle F of the faces sharing u'. Next we intersect $C_{\mathscr{B}}$ with $x_3 = 1$ and obtain either a polygon or a straight-line segment: in any case the problem is reduced to finding the intersection of two plane polygons, which can be solved in time $O(n)$ [1]. This enables us to find a vector orthogonal to a separating plane; the construction is completed by requiring that the plane contain a point internal to the segment $u'u''$.

Thus, we conclude that the construction of a separating plane of two three-dimensional sets of points, if it exists, can be effected in time $O(n \log n)$.

Appendix

As we said in Section 2, the vertex-to-edge list of a planar graph is a collection of edge lists, referred to as *input edge lists*, stored in arrays $H[1:n]$, VERTEX$[1:2m]$, and NEXT$[1:2m]$. In the DCEL, we can identify n cycles of edges around a vertex, called *vertex cycles*, and f cycles of edges around a face, called *face cycles*. The construction of the DCEL is carried out in two phases. In the first phase, we fill the arrays $V1$, $V2$, $P1$, and $P2$, hereby constructing the vertex cycles. In the second

phase we generate the names of the fa
constructing the face cycles.

Informally, phase-1 of the algorithm
scanned one at a time, in the order v_1, v_2
of v_j an edge (v_j, v_i) is entered into the D
that each edge is entered only once. T
present in the DCEL, since it was enter
earlier in the execution of the algorithm
realization of the appropriate linking of
cycle of v_j. To effect it we must determin
can be done as follows with additional s
the input edge list of v_h, the edge $(v_h,$
edge is linked permanently into the vert
edges of the form (v_r, v_j), with $r < j$. The
temporary list of v_j, are linked in reverse
execution of the algorithm. Thus this l
stored in an array LAST$[1:n]$. With t
easily obtained: in fact, prior to linking t
list of v_j starting from LAST$[j]$ and stor
auxiliary array $B[1:n]$. Notice that the l
be used repeatedly for each v_j. Therefor
the arrays LAST and B, both of size $O(r$

We can now give the algorithm.

Construct Vertex Cycles

```
1.  begin a ← 1
2.        for j ← 1 step 1 until n do
          LAST[j] ← Λ (Comment: in
3.        for j ← 1 step 1 until n do
4.          begin l ← LAST[j]
5.                While l ≠ Λ do
6.                  begin p ← V1[l]
7.                        B[p] ← l
8.                        l ← P2[l]
                  end
                  Comment: Loop
                  (v_r, v_j) with r < j
                  stores them into
                  j = 1.
9.                t ← H[j],
10.               r ← VERTEX[t]
```

phase we generate the names of the faces and fill the arrays $F1$ and $F2$, hereby constructing the face cycles.

Informally, phase-1 of the algorithm works as follows. The input edge lists are scanned one at a time, in the order $v_1, v_2, \ldots, v_n$. While scanning the input edge list of v_j an edge (v_j, v_i) is entered into the DCEL only if $i > j$: in this manner we ensure that each edge is entered only once. Thus any edge (v_j, v_h) with $h < j$ is already present in the DCEL, since it was entered while scanning the input edge list of v_h earlier in the execution of the algorithm. All that is needed now is therefore the realization of the appropriate linking of such (v_j, v_h) into its position in the vertex cycle of v_j. To effect it we must determine the location of (v_j, v_h) in the DCEL. This can be done as follows with additional storage $O(n)$. Suppose that, while scanning the input edge list of v_h, the edge (v_h, v_j) is to be entered (obviously $h < j$). This edge is linked permanently into the vertex cycle of v_h and temporarily into a list of edges of the form (v_r, v_j), with $r < j$. The members of the latter list referred to as the *temporary list* of v_j, are linked in reverse order to that of their occurrence during the execution of the algorithm. Thus this list can be managed with only one pointer stored in an array LAST$[1:n]$. With these provisions, the location of (v_h, v_j) is easily obtained: in fact, prior to linking the vertex-cycle of v_j we scan the temporary list of v_j starting from LAST$[j]$ and store the location of (v_h, v_j) into cell $B[h]$ of an auxiliary array $B[1:n]$. Notice that the latter array is only scratch memory and will be used repeatedly for each v_j. Therefore the additional storage needed consists of the arrays LAST and B, both of size $O(n)$, and of program variables a_1, a_0, u, t, r, l.

We can now give the algorithm.

Construct Vertex Cycles

```
1.  begin a ← 1
2.        for j ← 1 step 1 until n do
          LAST[j] ← Λ (Comment: initialize LAST)
3.        for j ← 1 step 1 until n do
4.          begin l ← LAST[j]
5.                While l ≠ Λ do
6.                  begin p ← V1[l]
7.                        B[p] ← l
8.                        l ← P2[l]
                    end
                    Comment: Loop 5–8 fetches the locations of all edges
                    (v_r, v_j) with r < j by scanning the temporary list of v_j and
                    stores them into the array B. This step is obviously void for
                    j = 1.
9.                t ← H[j],
10.               r ← VERTEX[t]
```

```
11.              If r > j then
12.                begin V1[a] ← j, V2[a] ← r
13.                      HV[j] ← a0 ← a, u ← 1
14.                      P2[a] ← LAST[r]
15.                      LAST[r] ← a
16.                      a ← a + 1
                   end
                 Comment: Steps 10–15 initialize the vertex cycle for v_j
17.              else HV[j] ← a0 ← B[r], u ← 2
                 Comment: Steps 8–17 process the first member (v_r, v_j) of the
                 input edge list of v_j. If this edge was not previously encoun-
                 tered steps 11–16 are executed; specifically, the edge is
                 entered in step 12. Variable a0 is used to denote the location
                 of the last member of the vertext cycle being constructed.
18.              While NEXT[t] ≠ H[j] do
19.                begin t ← NEXT[t]
20.                      r ← VERTEX[t]
21.                      If r > j then
22.                        begin V1[a] ← j, V2[a] ← r
23.                              P2[a] ← LAST[r]
24.                              LAST[r] ← Pu[a0] ← a,
25.                              a0 ← a, u ← 1
26.                              a ← a + 1
                           end
                         else
27.                        begin Pu[a0] ← B[r]
28.                              a0 ← B[r], u ← 2
                           end
                 end
29.              Pu[a0] ← HV[j]
    end
end
```

Comment: Steps 18–29 complete the construction of the vertex cycle for v_j. Specifically, loop 18–28 successively processes the edges incident on v_j and either enters them into the vertex cycle (Steps 21–26) or simply links them into it (Steps 27–28). Step 29 closes the vertex cycle.

To evaluate the running time of the algorithm just described, notice that each edge is processed exactly twice: once to be entered into a vertex cycle and into a

temporary list, the second time to be li
take constant time, and since the numbe
the arrays $V1$, $V2$, $P1$, and $P2$.

To complete the construction of the
The next algorithm, CONSTRUCT F
DCEL which is produced by the C
cedure. The algorithm will scan the D
$F1[a]$ and $F2[a]$ have already been fi
otherwise it generates the name of ne
edges enclosing it filling the appropr
when $2m$ filling operations have been p
this event.

Construct face cycles

```
1.  begin          for j ← 1 step 1 unti
2.                 a ← s ← k ← 1
3.                 While k ≤ 2m do
4.                   begin If F1[a] ≠
5.                         else begin
6.                                   F

7.                                   a
8.                                   V

9.
10.
11.

12.                                  s
                                   end
                     end
    end
```

Since in the latter algorithm each field
twice (once to be filled in steps 6 or 10
step 4), the running time is $O(n)$. This a
algorithm substantiate our claim that th
the original vertex-to-edge list.

temporary list, the second time to be linked appropriately. Both these operations take constant time, and since the number n of edges is $O(n)$, $O(n)$ time is used to fill the arrays $V1$, $V2$, $P1$, and $P2$.

To complete the construction of the DCEL we must construct the face cycles. The next algorithm, CONSTRUCT FACE CYCLES, starts from the partial DCEL which is produced by the CONSTRUCT VERTEX CYCLES procedure. The algorithm will scan the DCEL, using an integer a as a counter. If $F1[a]$ and $F2[a]$ have already been filled, it advances to the subsequent edge; otherwise it generates the name of new face (using a counter s) and traces the edges enclosing it filling the appropriate F-fields. The algorithm terminates when $2m$ filling operations have been performed: an integer k is used to control this event.

Construct face cycles

```
1.  begin   for j ← 1 step 1 until m do F1[j] ← F2[j] ← Λ
2.          a ← s ← k ← 1
3.          While k ≤ 2m do
4.            begin If F1[a] ≠ Λ and F2[a] ≠ Λ then a ← a + 1
5.                  else begin If F1[a] = Λ then u ← 1 else u ← 2
6.                             Fu[a] ← s, c ← Vu[a], HF[s] ← a0 ← a,
                                                          k ← k + 1
7.                             a ← Pu[a]
8.                             While a ≠ a0 do
9.                               begin If V1[a] = c then u ← 2 else u ← 1
10.                                    Fu[a] ← s, c ← Vu[a], k ← k + 1
11.                                    a ← Pu[a]
                                 end
12.                            s ← s + 1
                           end
                end
    end
```

Since in the latter algorithm each field $F1[a]$ or $F2[a]$ is being processed at most twice (once to be filled in steps 6 or 10, and possibly once to be just inspected in step 4), the running time is $O(n)$. This and the analogous result for the vertex cycle algorithm substantiate our claim that the DCEL can be obtained in time $O(n)$ from the original vertex-to-edge list.

References

[1] M.I. Shamos, *Computational Geometry*, Department of Computer Science, Yale University (1977): to be published by Springer-Verlag.
[2] M.I. Shamos, Geometric Complexity, *Proc. Seventh Ann. ACM Symp. on Theory of Computing* (May 1975) 224–233.
[3] F.P. Preparata and S.J. Hong, Convex hulls of finite sets in two and three dimensions, *Commun. ACM* **20** (1977) 87–93.
[4] H.T. Kung, F. Luccio, and F.P. Preparata, On finding the maxima of a set of vectors, *J. ACM* **22** (1975) 469–76.
[5] D.T. Lee and F.P. Preparata, 'Location of a point in a planar subdivision and its applications, *SIAM J. Comput.* **6** (1977) 594–606.
[6] F.P. Preparata, Location of a set of points in a planar subdivision (submitted for publication); Available in: Steps into Computational Geometry, Report ACT-1, Coordinated Science Laboratory, University of Illinois, Urbana (February 1977).
[7] G. Ewald, *Geometry: An Introduction* (Wadsworth, Belmont, Ca, 1971).
[8] J. Stoer and C. Witzgall, *Convexity and Optimization in Finite Dimensions*, I (Springer-Verlag, Berlin, 1970).

Theoretical Computer Science 7 (1978) 237–238.
© North-Holland Publishing Company

ANNOUNCEMENTS

IJCAI-79, SIXTH INTERNATIONAL ARTIFICIAL INTELLIGENCE

August 20–24, 1979, Tokyo, Japan

Deadline for submission of papers to I

Contact

Dr. Pat Hayes, Program Committee C
Department of Computer Science,
University of Essex,
Colchester CO4 3SQ,
Essex, U.K.
(0206) 44144, ext. 2371 or 2325.

Professor Raj Reddy, General Chairm
Department of Computer Science,
Carnegie-Mellon University,
Pittsburgh, PA 15213, U.S.A.

Sponsor

International Joint Conference on Art

ASLIB-TECHNICAL TRANSLATIO SEMINAR ON TRANSLATING AN

Tuesday, November 14, 1978, Lecture T
London EC1, U.K.

Machine translation has reemerged a growing evidence of its value. Compu problems of translation in an increas dictionaries. The primary purpose of the to this new dimension in their activities a systems are aware of practical needs. The