An Optimal Algorithm for Finding the Kernel of a Polygon

D. T. LEE AND F. P. PREPARATA

University of Illinois at Urbana-Champaign, Urbana, Illinois

ABSTRACT The kernel $K(P)$ of a simple polygon P with n vertices is the locus of the points internal to P from which all vertices of P are visible Equivalently, $K(P)$ is the intersection of appropriate half-planes determined by the polygon's edges Although it is known that to find the intersection of n generic half-planes requires time $O(n \log n)$, we show that one can exploit the ordering of the half-planes corresponding to the sequence of the polygon's edges to obtain a kernel finding algorithm which runs in time $O(n)$ and is therefore optimal

KEY WORDS AND PHRASES computational complexity, optimal algorithms, simple polygon, kernel of polygon

CR CATEGORIES 4 49, 5 25, 5 32

1. *Introduction*

The *kernel* $K(P)$ of a simple polygon P is the locus of the points internal to P which can be joined to every vertex of P by a segment totally contained in P. Equivalently, if one considers the boundary of P as a counterclockwise directed cycle, the kernel of P is the intersection of all the half-planes lying to the left of the polygon's edges.

Shamos and Hoey [1] have presented an algorithm for finding the kernel of an n-edge polygon in time $O(n \log n)$. Their algorithm is based on the fact that the intersection of n generic half-planes can be found in time $O(n \log n)$; they also show that $\Omega(n \log n)$ is a lower bound to the time for finding the intersection of n half-planes. However, this lower bound does not apply to the problem of finding the kernel, since in the latter case the half-planes are ordered according to the sequence of the edges of P, nor does their algorithm take advantage of this order. In this note we shall show that, indeed, this ordering can be exploited to yield an algorithm which runs in time linear in the number of the edges. Obviously, since each edge must be examined, the time of our algorithm is optimal within a multiplicative constant.

The model of computation used for the above results, which we shall also adopt in this paper, is a random-access machine (RAM) with real-number arithmetic, i.e. with the capability of performing comparisons of real numbers and rational operations on real numbers.

The input polygon P is represented by a sequence of vertices $v_0, v_1, \ldots, v_{n-1}$, with $n \geq 4$, in which each v_i has real-valued x- and y-coordinates (x_i, y_i), and (v_{i-1}, v_i) (see Footnote 1), for $i = 1, 2, \ldots, n$, is the edge of the polygon connecting vertices v_{i-1} and v_i. For ease of reference we shall describe P by a circular list of vertices and intervening edges as $v_0e_1v_1e_2 \cdots e_{n-1}v_{n-1}e_0v_0$, where $e_i = (v_{i-1}, v_i)$. We also impose a direction upon each edge

This work was supported by the National Science Foundation under Grant MCS 76-17321 and by the Joint Services Electronics Program under Contract DAAB-07-72-C-0259

Authors' present addresses D T Lee, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201, F.P Preparata, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL 61801

[1] All indices are taken modulo n

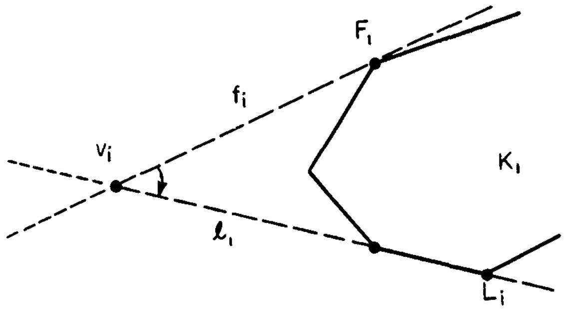

FIG 1 Illustration of the definition of F_i and L_i

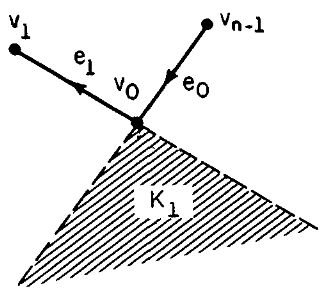

FIG 2 Illustration of polygon K_1

such that the interior of the polygon lies to the left of the edge, or, equivalently, the boundary of P is directed counterclockwise. A vertex v_i is called *reflex* if v_{i+1} lies to the right of the line containing e_i and directed like e_i, that is, if the interior angle at v_i is larger than 180°; a vertex is called *convex* otherwise. We also assume that the interior angle at a convex vertex v_i be strictly smaller than 180°, since the elimination of straight-angle vertices does not change P and can be done by a preliminary scan of the boundary of P in time $O(n)$. It is obvious that the kernel of P, being the intersection of half-planes, is a convex polygon $K(P)$ and is bounded by at most n edges. Thus if the kernel is nonempty, the output will also be represented by the sequence of vertices and edges of the polygon $K(P)$.

2. *The Algorithm*

The algorithm we shall outline scans in order the vertices of P and construct a sequence of convex polygons $K_1, K_2, \ldots, K_{n-1}$. Each of these polygons may or may not be bounded. We shall later show (Lemma 1) that K_i is the common intersection of the half-planes lying to the left of the directed edges $e_0, e_1, \ldots, e_i$. This result has the obvious consequences that $K_{n-1} = K(P)$ and that $K_1 \supseteq K_2 \supseteq \cdots \supseteq K_i$; the latter implies that there is some $r > 1$ such that K_i is unbounded or bounded depending upon whether $i < r$ or $i \geq r$, respectively.

Notationally, if points w_i and w_{i+1} belong to the line containing the edge e_{s_i} of P, then $w_i e_{s_i} w_{i+1}$ denotes the segment between w_i and w_{i+1} with the same direction as e_{s_i}. When a polygon K_i is unbounded, two of its edges are half-lines; so, $\Lambda e w$ denotes a half-line terminating at point w and directed like edge e, while $we\Lambda$ denotes the complementary half-line.

During the processing, the boundary of K is maintained as a doubly linked list of vertices and intervening edges. This list will be either linear or circular, depending upon whether K_i is unbounded or bounded, respectively. In the first case, the first and last item of the list will be called the *list head* and *list tail*, respectively.

Among the vertices of K_i we distinguish two vertices F_i and L_i, defined as follows. Consider the two lines of support[2] of K_i through vertex v_i of P. Let f_i and l_i be the two half-lines of these lines which contain the points of support, named so that the clockwise angle from f_i to l_i in the plane wedge containing K_i is no greater than π (Figure 1). Vertex F_i is the point common to f_i and K_i which is farthest from v_i; L_i is similarly defined. These two vertices play a crucial role in the construction of K_{i+1} from K_i.

If P has no reflex vertex, then P is convex and trivially $K(P) = P$. Thus let v_0 be a reflex vertex of P. We can now describe the kernel algorithm.

Initial Step We set K_1 equal to the intersection of the half-planes lying to the left of edges e_0 and e_1, i e $K_1 \leftarrow \Lambda e_1 v_0 e_0 \Lambda$ (Figure 2) $F_1 \leftarrow$ point at infinity of $\Lambda e_1 v_0$, $L_1 \leftarrow$ point at infinity of $v_0 e_0 \Lambda$

---

[2] Recall that l is a line of support of a polygon P if l has at least one point in common with P and the interior of P entirely lies on one side of l

General Step We distinguish several cases We assume that the vertices of K_i be numbered consecutively as w_1, w_2, , counterclockwise

(1) *Vertex v_i is reflex* (see Figures 3(a, b))

(1 1) *F_i lies on or to the right of* $\Lambda e_{i+1}v_{i+1}$ (Figure 3(a)) We scan the boundary of K_i *counterclockwise from F_i* until either we reach a unique edge $(w_{t-1}w_t)$ of K_i intersecting $\Lambda e_{i+1}v_{i+1}$ or we reach L_i without finding such an edge In the latter case, we terminate the algorithm ($K(P) = \varnothing$) In the former case, we take the following actions

(i) We find the intersection w' of $(w_{t-1}w_t)$ and $\Lambda e_{i+1}v_{i+1}$

(ii) We scan the boundary of K_i *clockwise from F_i*, until either we reach an edge $(w_{s-1}w_s)$ intersecting $\Lambda e_{i+1}v_{i+1}$ at a point w'' (this is guaranteed if K_i is bounded) or (only when K_i is unbounded) we reach the list head without finding such an edge In the first case, letting $K_i = \alpha w_s \cdots w_{t-1}\beta$ (where α and β are sequences of alternating edges and vertices), we set $K_{i+1} \leftarrow \alpha w'' e_{i+1} w' \beta$, in the second case ($K_i$ is unbounded) we must test whether K_{i+1} is bounded or unbounded If the slope of $\Lambda e_{i+1}v_{i+1}$ is comprised between the slopes of the initial and final half-lines of K_i, then $K_{i+1} \leftarrow \Lambda e_{i+1} w' \beta$ is also unbounded Otherwise we begin scanning the boundary of K_i *clockwise from the list tail* until an edge $(w_{r-1}w_r)$ is found which intersects $\Lambda e_{i+1}v_{i+1}$ at a point w'', letting $K_i = \gamma w_{t-1}\delta w_r \eta$ we set $K_{i+1} \leftarrow \delta w'' e_{i+1} w'$ and the list *becomes circular*

The selection of F_{i+1} is done as follows If $\Lambda e_{i+1}v_{i+1}$ has just one intersection with K_i, then $F_{i+1} \leftarrow$ (point at infinity of $\Lambda e_{i+1}v_{i+1}$), otherwise $F_{i+1} \leftarrow w''$ To determine L_{i+1}, we scan K_i counterclockwise from L_i until either a vertex w_u of K_i is found such that w_{u+1} lies to the left of $v_{i+1}(v_{i+1}w_u)\Lambda$, or the list of K_i is exhausted without finding such vertex In the first case $L_{i+1} \leftarrow w_u$, in the other case (which may happen only when K_i is unbounded) $L_{i+1} \leftarrow L_i$

(1 2) *F_i lies to the left of* $\Lambda e_{i+1}v_{i+1}$ (Figure 3(b)) In this case $K_{i+1} \leftarrow K_i$, but F_i and L_i must be updated To determine F_{i+1}, we scan K_i *counterclockwise from F_i* until we find a vertex w_t of K_i such that w_{t+1} lies to the right of $v_{i+1}(v_{i+1}w_t)\Lambda$, we then set $F_{i+1} \leftarrow w_t$ The determination of L_{i+1} is the same as in case (1 1)

(2) *Vertex v_i is convex* (see Figures 4(a, b))

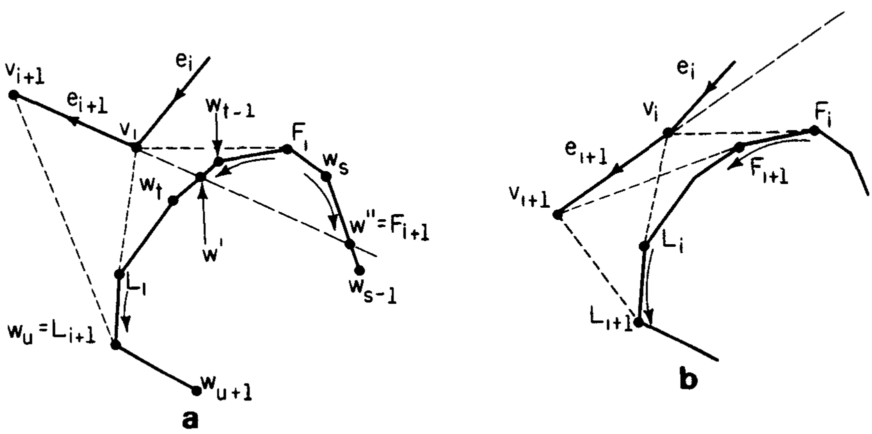

FIG 3 General step when v_i is reflex

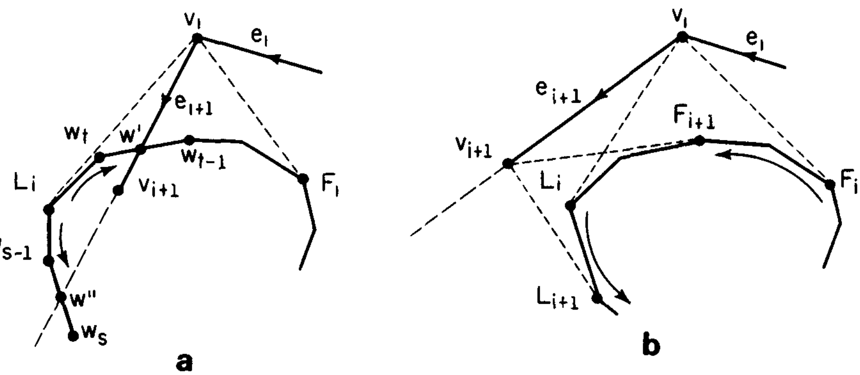

FIG 4 General step when v_i is convex

(2.1) *L_i lies on or to the right of $v_ie_{i+1}\Lambda$* (Figure 4(a)). We scan the boundary of K_i *clockwise from* L_i until either we reach a unique edge $(w_{t-1}w_t)$ intersecting $v_ie_{i+1}\Lambda$ or we reach F_i without finding such an edge. In the latter case, we terminate the algorithm ($K(P) = \varnothing$). In the former case, we take the following actions:

(i) We find the intersection w' of $(w_{t-1}w_t)$ and $v_ie_{i+1}\Lambda$.

(ii) We scan the boundary of K_i *counterclockwise from* L_i until either we reach an edge $(w_{s-1}w_s)$ intersecting $v_ie_{i+1}\Lambda$ at point w'' (guaranteed if K_i is bounded) or (only when K_i is unbounded) we reach the list tail without finding such an edge. Letting $K_i = \alpha w_t \cdots w_{s-1}\beta$, in the first case we let $K_{i+1} \leftarrow \alpha w' e_{i+1} w'' \beta$; in the second case ($K_i$ is unbounded) we must test whether K_{i+1} is bounded or unbounded. If the slope of $v_ie_{i+1}\Lambda$ is comprised between the slopes of the initial and final half-lines of K_i, then $K_{i+1} \leftarrow \alpha w' e_{i+1}\Lambda$ is also unbounded. Otherwise we begin scanning the boundary of K_i *counterclockwise from the list head* until an edge $(w_{r-1}w_r)$ is found which intersects $v_ie_{i+1}\Lambda$ at a point w''; letting $K_i = \gamma w_{r-1}\delta w_t\eta$ we set $K_{i+1} \leftarrow \delta w' e_{i+1} w''$ and the list *becomes circular*.

The selections of F_{i+1} and L_{i+1} depend upon the position of v_{i+1} on the half-line $v_ie_{i+1}\Lambda$ and upon whether $v_ie_{i+1}\Lambda$ has one or two intersections with K_i. We distinguish these two cases.

(2.1.1) $v_ie_{i+1}\Lambda$ intersects K_i in w' and w''. If $v_{i+1} \in [v_ie_{i+1}w']$ then F_{i+1} is selected as in case (1.2). Otherwise F_{i+1} is set to w'. If $v_{i+1} \in [v_ie_{i+1}w'']$ then L_{i+1} is set to w''. Otherwise L_{i+1} is selected as in case (1.1) except that we scan K_{i+1} *counterclockwise from* w''.

(2.1.2) $v_ie_{i+1}\Lambda$ intersects K_i in just w'. If $v_{i+1} \in [v_ie_{i+1}w']$, F_{i+1} is selected as in case (1.2), otherwise $F_{i+1} \leftarrow w'$. L_{i+1} is set to the point at infinity of $v_ie_{i+1}\Lambda$.

(2.2) *L_i lies to the left of $v_ie_{i+1}\Lambda$* (Figure 4(b)). In this case $K_{i+1} \leftarrow K_i$. F_{i+1} is determined as in (1.2). If K_i is bounded then L_{i+1} is determined as in case (1.1), otherwise $L_{i+1} \leftarrow L_i$.

The correctness of the algorithm is asserted by the following lemma, where we let H_j denote the half-plane lying to the left of line $\Lambda e_j\Lambda$.

LEMMA 1. *The polygon K_{i+1} is the intersection of $H_0, H_1, \ldots, H_{i+1}$ for $i = 0, 1, \ldots, n-2$.*

PROOF. By induction. Notice that K_1 is by definition the intersection of H_0 and H_1 (initial step of the algorithm). Assume inductively that $K_i = H_0 \cap H_1 \cap \cdots \cap H_i$. Then in all cases contemplated in the general step we constructively intersect K_i and H_{i+1}, thereby establishing the claim. □

While Lemma 1 guarantees that the algorithm correctly constructs $K(P)$, a minor but important modification of the general step is needed in order to achieve efficiency. In fact, there could be polygons P, with $K(P) = \varnothing$, for which time $O(n^2)$ could be used before termination. This can be avoided by an additional test based on the following properties of kernels.

LEMMA 2. *Let P be a simple polygon and suppose that $K(P) \neq \varnothing$. For any points $p \in K(P)$ and u on the boundary of P, the segment (pu) is contained in P.*

PROOF. Let u belong to edge $e_j = (v_{j-1}v_j)$ of P, and consider the triangle $(pv_{j-1}v_j)$ (Figure 5). Assume the segment (pu) is not contained in P, and let q be a point of (pu) external to P. Then there are two edges, e_s and e_r, of the boundary of P which cross (pu) on opposite sides of q. Since $p \in K(P)$, no edge of P crosses either (pv_{j-1}) or (pv_j) except possibly at v_{j-1} or v_j, respectively. Since the boundary of P is a single cycle, e_s and e_r belong to a chain C of edges between v_j and v_{j-1}. But Ce_j is closed; hence it coincides with the boundary of P

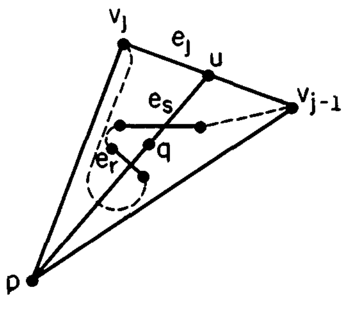

FIG. 5. If $p \in K(P)$ no point of (pu) is external to P

since P is simple; moreover, $P \subset (pv_{j-1}v_j)$. Also, by the simplicity of P, both (pv_j) and (pv_{j-1}) cannot belong to Ce_j; hence, at least one of them is external to P, a contradiction. □

Now consider the two lines of support of K_i through vertex v_0 of P (Figure 6). Let f and l be the two half-lines of these lines containing the points of support, named so that the clockwise angle from f to l is convex. Also let f^* be the segment between v_0 and the point of support farthest from v_0, and let $\bar{f}$ be the complementary half-line of f; l^* and $\bar{l}$ are similarly defined.

THEOREM 1. *Suppose that K_{i+1} is nonempty and that e_{i+1} crosses either l^* or $\bar{f}$,* with v_{i+1} *in the convex wedge delimited by $\bar{f}$ and l (crosshatched in Figure 6); then $K(P) = \varnothing$.*

PROOF. Suppose that e_{i+1} crosses l^* (Figure 6(a)); then we claim that the boundary of P separates K_{i+1} from v_0. Indeed, this is obvious if e_{i+1} crosses both f^* and l^*. If not (i.e. v_i is in the wedge bounded by f and l), the boundary of P cannot cross l^* more than once; otherwise K_{i+1} would lie *on the right* of some edge $\Lambda e_s \Lambda$ $(s < i)$ and would therefore be empty. Suppose now $K(P) \neq \varnothing$ and let p be a point in $K(P)$, obviously $p \in K_{i+1}$, and the segment (pv_0) is entirely contained in P. But (pv_0) crosses the boundary of P, whence a contradiction and $K(P) = \varnothing$.

Assume now that e_{i+1} crosses $\bar{f}$ (Figure 6(b)); then we claim that the boundary of P cuts the convex wedge delimited by $\bar{l}$ and $\bar{f}$. Indeed, this is obvious if e_{i+1} crosses both $\bar{f}$ and $\bar{l}$; if not (i.e. v_i is inside the wedge bounded by $\bar{f}$ and $\bar{l}$), the boundary of P cannot cross $\bar{f}$ more than once, by the same argument given above. Suppose now $K(P) \neq \varnothing$, with $p \in K(P) \subseteq K_{i+1}$; then the half-line $p(pv_0)\Lambda$ reaches the boundary of P within the above wedge in a point u. But, by Lemma 2, the segment (pu) must be contained in P; however, since it crosses its boundary at v_0, we have a contradiction and $K(P) = \varnothing$. □

Therefore we shall modify the general step of the algorithm by adding the following additional operations (test and update): (1) Before determining K_{i+1}, F_{i+1}, and L_{i+1}: If e_{i+1} crosses either l^* or $\bar{f}$, with v_{i+1} in the convex wedge delimited by $\bar{f}$ and l, terminate the algorithm with $K(P) = \varnothing$. (2) In cases (1.1) and (2.1), after determining K_{i+1}, F_{i+1}, and L_{i+1}: Let F^*, L^* be points of support on f and l, respectively. Initially, since K_1 is $\Lambda e_1 v_0 e_0 \Lambda$, F^* and L^* are set to points at infinity of $\Lambda e_1 v_0$ and $v_0 e_0 \Lambda$, respectively. If in obtaining K_{i+1} from K_i, the vertices F^* and/or L^* are deleted, we update them accordingly as follows (of course only the required updates are performed): (i) v_i *is reflex.* $F^* \leftarrow w'$, $L^* \leftarrow w''$ if v_0 lies to the left of $\Lambda e_{i+1} v_{i+1}$ and $F^* \leftarrow w''$, $L^* \leftarrow w'$ otherwise. (ii) v_i *is convex.* $F^* \leftarrow w''$, $L^* \leftarrow w'$ if v_0 lies to the left of $v_i e_{i+1} \Lambda$ and $F^* \leftarrow w'$, $L^* \leftarrow w''$ otherwise. Note that w', w'' are determined in cases (1.1) and (2.1), and that if w'' does not exist, its place is taken by the point at infinity of the half-line being considered.

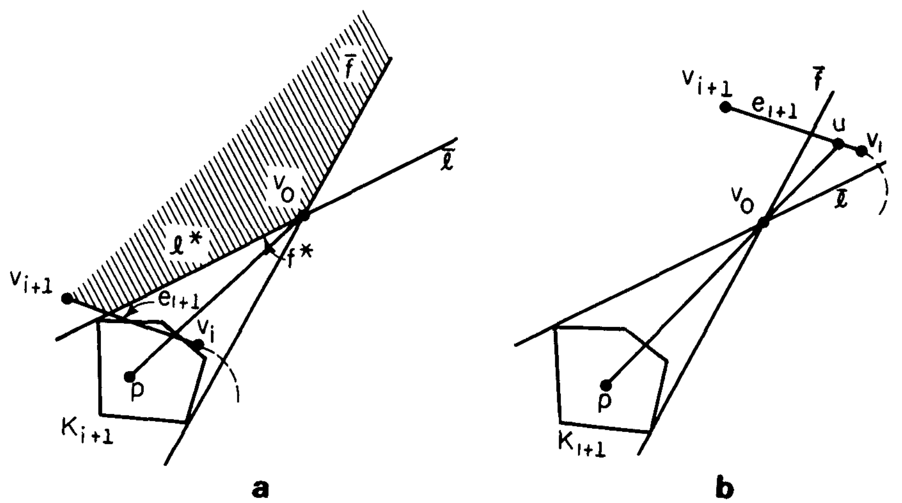

FIG 6 Illustrations for the proof of Theorem 1

We say that K_s, for $s = i, \dots, n-1$, is *vacuous* if the above test fails when processing edge e_i. We have the following corollary:

COROLLARY 1. *Suppose K_i is not vacuous and let p be any point in K_i. Also let α_j be the interior angle at p in the triangle $(pv_{j-1}v_j)$, positive if (pv_j) follows (pv_{j-1}) counterclockwise, for $j = 1, \dots, i$. Then we claim that $\sum_{j=1}^{i} \alpha_j < 3\pi$.*

PROOF. Suppose that $\sum_{j=1}^{i}\alpha_j \geq 3\pi$. This means that the boundary of P, starting at v_0, wraps around $p \in K_i$ as shown either in Figure 7(a) or in Figure 7(b). In both cases, K_i is bounded. In the first case the boundary of P crosses l in at least two points, each on opposite sides of the point(s) of support; in the second case, the boundary of P makes a full turn around v_0 and must therefore cross $\bar{f}$. In either case, the additional test described above will fail, contrary to the hypothesis that K_i is not vacuous. □

3. *Performance Analysis*

It is convenient to analyze separately the two basic types of actions performed by the kernel algorithm. The first concerns updating the kernel, by intersecting K_i with $\Lambda e_{i+1}\Lambda$ to obtain K_{i+1}; the second concerns updating F_i and L_i and consists of counterclockwise or *forward* scans of K_i to obtain the new vertices of support (note however that in some cases, as (1.1) and (2.1), the update of K_i implicitly yields updates for one or the other of the support vertices).

We begin by considering intersection updates. In case (1.1) (when the algorithm does not terminate), we scan K_i starting from F_i both clockwise and counterclockwise (this scan also finds F_{i+1}). Let ν_i be the total number of edges visited before finding the two intersections w' and w''. This process actually removes $\nu_i - 2$ edges from K_i (those comprised between w_s and w_{t-1} in Figure 3(a)), and since each of the removed edges is collinear with a distinct edge of P, we have $\sum(\nu_i - 2) \leq n$. Thus $\sum\nu_i$, the total number of vertices visited by the algorithm in handling case (1.1), is bounded above by $3n$, i.e. it is $O(n)$. The same argument, with insignificant modifications, can be made for case (2.1).

Next, we consider those updates of the support vertices F and L which are not implicitly accomplished in the intersection process. These updates occur for L in all cases (1.1), (1.2), (2.1), and (2.2), and for F in cases (1.2) and (2.2). Note that in all of these cases the vertices of support *advance* on the boundary of K_i. Let us consider, for example, the update of L in case (1.1); the other cases can be treated analogously. Consider the set of edges of K_{i+1} which the algorithm visits before determining L_{i+1}; the visit to the edge immediately following L_{i+1} is referred to as an *overshoot*. It is immediately realized that in handling case (1.1) the number of overshoots is globally $O(n)$, since there is at most one overshoot per vertex of P. Next, we claim that, ignoring overshoots, any edge is visited at most twice. In fact, assume that, when processing v_i, an edge is being visited for the third time. Because

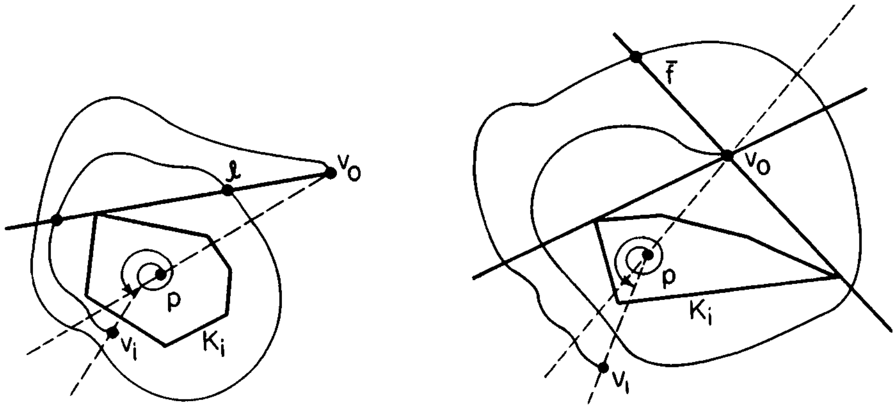

FIG 7 Illustrations for the proof of Corollary 1

of the forward scan feature, this implies that the boundary of P wraps around K_i at least twice, i.e. there is some point $q \in K_i$ for which the construction of Corollary 1 yields $\sum \alpha_j \geq 4\pi$, contrary to Corollary 1.

Thus the work performed in handling case (1.1)—as well as cases (1.2), (2.1), and (2.2)—is $O(n)$. Finally, the updates of F^* and L^* are all accomplished implicitly in finding w' and w''. Therefore, we conclude that the entire algorithm runs in time proportional to the number of vertices of P and is optimal to within a constant factor.

ACKNOWLEDGMENT. The authors thank their referees for extremely pertinent and valuable suggestions which have substantially improved the quality of this paper.

REFERENCES

1 SHAMOS, M I, AND HOEY, D Geometric intersection problems 17th Annual Symp on Foundations of Computer Science, Houston, Tex , Oct 1976, pp 208-215 (IEEE)

RECEIVED MAY 1977, REVISED JANUARY 1978