LINEAR TIME ALGORITHMS FOR VISIBILITY AND SHORTEST PATH PROBLEMS INSIDE SIMPLE POLYGONS

Leo Guibas(1,2), *John Hershberger*(1), *Daniel Leven*(3), *Micha Sharir*(3),(4), *Robert E. Tarjan*(5),(6)

(1) Computer Science Department Stanford University

(2) DEC/SRC

(3) School of Mathematical Sciences, Tel-Aviv University

(4) Courant Institute of Mathematical Sciences New York University

(5) AT&T Bell Laboratories

(6) Department of Computer Science Princeton University

ABSTRACT

We present linear time algorithms for solving the following problems involving a simple planar polygon P: (i) Computing the collection of all shortest paths inside P from a given source vertex s to all the other vertices of P; (ii) Computing the subpolygon of P consisting of points that are visible from a segment within P; (iii) Preprocessing P so that for any query ray r emerging from some fixed edge e of P, we can find in logarithmic time the first intersection of r with the boundary of P; (iv) Preprocessing P so that for any query point x in P, we can find in logarithmic time the portion of the edge e that is visible from x; (v) Preprocessing P so that for any query point x inside P and direction u, we can find in logarithmic time the first point on the boundary of P hit by the ray at direction u from x; (vi) Calculating a hierarchical decomposition of P into smaller polygons by recursive polygon cutting, as in [Ch]. (vii) Calculating the (clockwise and counterclockwise) "convex ropes" (in the terminology of [PS]) from a fixed vertex s of P lying on its convex hull, to all other vertices of P. All these algorithms are based on a recent linear time algorithm of Tarjan and Van Wyk for triangulating a simple polygon, but use additional techniques to make all subsequent phases of these algorithms also linear.

Work on this paper by the fourth author has been supported by Office of Naval Research Grant N00014-82-K-0381, National Science Foundation Grant No. NSF-DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation, and by a Grant from the U.S-Israeli Binational Science Foundation.

1. Introduction

Recently Tarjan and Van Wyk [TV] have developed a linear-time algorithm for triangulating simple polygons, thereby improving the previous $O(n \log n)$ algorithm of [GJPT], and solving a major open problem in computational geometry. This result has extended in a significant way the list of problems already known to be solvable in linear time on simple polygons, which has included e.g. calculation of the convex hull of such a polygon [GY], [MA], calculation of the subpolygon of P visible from a given point [Le], [EA], and more. In addition, there are many problems known to be linear-time equivalent to the triangulation problem for simple polygons (for a list of these see e.g. [FM]) that are now therefore also solvable in linear time. Also, several other problems on simple polygons were given linear time solutions, provided that a triangulation of the given polygon is already available, and are thus now also solvable in linear time. These problems include calculation of the shortest path inside a simple polygon between two specified points [LP], preprocessing a simple polygon to support logarithmic-time point location queries [Ki], [EGS], stationing guards in simple art galleries [Fi], etc.

In this paper we continue the exploration for linear-time algorithms for simple polygons. We present several new such algorithms, which are all based on the availability of a triangulation of the given polygon, but exploit additional new techniques to achieve the linear-time goal.

Our new linear time algorithms solve the following problems for a given simple polygon P with n sides.

(1) Given a fixed source point X inside P, calculate the shortest paths inside P from X to all vertices of P (in fact our algorithm even provides a (linear time) preprocessing of P into a data-structure from which the length of the shortest path inside P from X to any desired target point Y can be found in time $O(\log n)$; the path itself can be found in time $O(\log n + k)$, where k is the number of segments along this path).

(2) Given a fixed edge e of P, calculate the subpolygon $Vis(P,e)$ consisting of all points in P visible from (some point on) e.

(3) Given e as above, preprocess P so that, given any query ray r emanating from e into P, the first point on the boundary of P hit by r can be found in $O(\log n)$ time.

(4) Given e as above, preprocess P so that, given any point X inside P, the subsegment of e visible from X can be computed in $O(\log n)$ time.

(5) Preprocess P so that, given any point X inside P and direction u, the first point $hit(X,u)$ on the boundary of P hit by the ray at direction u from X can be computed in $O(\log n)$ time.

(6) Calculate a hierarchical balanced decomposition tree of P by recursively cutting P along diagonals, as in [Ch].

(7) Given a vertex X of P lying on its convex hull, calculate for all other vertices Y of P the clockwise and counterclockwise *convex ropes* around P from X to Y, when such paths exist (these are polygonal paths in the exterior of P from X to Y that wrap around P, always turning in a clockwise (resp. counterclockwise) direction; cf. Section 5 for more detail).

Our results improve previous algorithms given for some of these problems (cf. [Ch], [CG], [PS]). Most of our algorithms are based on the solution to Problem (1), and exploit interesting relationships between visibility and shortest-path problems for a simple polygon. Our technique for solving Problem (1) extends the technique of Lee and Preparata [LP] for calculating the shortest path inside P between a single pair of points, and uses *finger trees*, a data-structure for efficient access to an ordered list when there is locality of reference (see Guibas, McCreight, Plass and Roberts [GMPR]) and Huddleston and Mehlhorn [HM]), to obtain an overall linear-time performance.

The paper is organized as follows. In Section 2 we present the linear-time solution to Problem (1). The visibility problems (2) - (4) are then solved in Section 3. The polygon cutting procedure and the shooting problem (Problems (5)-(6)) are solved in Section 4, and the convex rope algorithm for Problem (7) is presented in Section 5.

2. Calculating the Shortest Path Tree of a Simple Polygon

Let P be a simple polygon having n vertices, and let s be a given *source vertex* of P (actually our algorithm will also apply in case s is an arbitrary point interior to, or on the boundary of P). Denote, for each vertex v of P, the Euclidean shortest path from s to v inside P by $\pi(s,v)$. It is well known (see e.g. [LP]) that $\pi(s,v)$ is a polygonal path whose corners are vertices of P, and that $\bigcup_v \pi(s,v)$, taken over all vertices v of P, is a planar tree Q_{sp} (rooted at s), which we call the *shortest path tree* of P (with respect to s); this tree has altogether n nodes, namely the vertices of P, and its edges are straight segments connecting these nodes. Our goal is to calculate this tree in linear time.

Let G be a triangulation of the interior of P (which can be computed in $O(n)$ time, using Tarjan and Van Wyk's algorithm [TV]). The planar dual T of G (whose vertices are the triangles in G and whose edges join two such triangles if they share an edge) is a tree, each of whose vertices has degree at most 3. Thus, for each vertex t of P, there is a unique minimal path π in T from some triangle containing s to another triangle containing t, which induces an ordered sequence of diagonals $d_1, d_2, \ldots, d_l$ of P (to be more precise, d_1 should be chosen as the first diagonal between two adjacent triangles in π which does not terminate at s, and d_l should be chosen as the last such diagonal not not terminating at t; this takes care of situations in which s or t is a vertex of more than one triangle in G). Each diagonal d_i thus divides P into two parts containing s and t respectively and therefore $\pi(s,t)$ must intersect only diagonals d_i, and each of them exactly once.

Let $d = uw$ be a diagonal or an edge of P and let a be the least common ancestor of u and w in the shortest path tree Q_{sp}. It is shown in [LP] that $\pi(a,u), \pi(a,w)$ are both *outward-convex;* i.e. the convex hull of each of these subpaths lies outside the region bounded by $\pi(a,u)$, $\pi(a,w)$, and by the segment uw. Following [LP], we call the union $F = F_{uw} = \pi(a,u) \cup \pi(a,w)$ the *funnel* associated with $d=uw$, and a the *cusp* of that funnel. Suppose next that d is a diagonal of P used by G, and let Δuwx be the unique triangle in G having d as an edge, which does not intersect the area bounded between F and d. Then the shortest path from s to x must start with $\pi(s,a)$ and then either continue along the straight segment ax if this segment does not intersect F, or else proceed along either $\pi(a,u)$ or $\pi(a,w)$ to a vertex v such that vx is a tangent to F at v, and then continue along the straight segment vx (see Fig 2.1). These observations form the basis of the algorithm of Lee and Preparata [LP], and of ours.

We are now ready to describe our algorithm. We first triangulate P in $O(n)$ time (as in [TV]). As the algorithm comes to process a diagonal $d = uw$ of P, it maintains the current funnel $F = F_{uw}$ as a sorted list $[u_l, u_{l-1}, \ldots, a, w_1, \ldots, w_k]$, where $a = u_0 = w_0$ is the cusp of F, $\pi(a,u) = [u_0, \ldots, u_l]$, $\pi(a,w) = [w_0, \ldots, w_k]$, and $u_l = u$, $w_k = w$. This list is stored in a *finger tree* (cf. [GMPR], [HM]). This structure is essentially a search tree equipped with *fingers* (which, in our application, are always placed at the first element and at the last element of the tree). This structure supports searching for an element x in time $O(\log \delta)$, where δ is the distance from x to the nearest finger, and also supports operations that split the tree into two subtrees at an element x in amortized time $O(\log \delta)$, with δ as above. The algorithm also maintains a pointer $CUSP(F)$ to the cusp a of the present funnel.

The algorithm begins by placing s and an adjacent vertex v_1 in F, with $CUSP(F) = s$. It then proceeds recursively as follows.

ALGORITHM PATH(F)

Let u and w be the first and the last elements of F, and let $a = CUSP(F)$ (thus $F = \pi(a,u) \cup \pi(a,w)$). Let Δuwx be the unique triangle in the triangulation G of P that has uw as an edge and that has not yet been processed (cf. Fig. 2.1).

(a) Search F for an element v, for which vx is a tangent to F at v (if the straight line segment ax does not intersect F then $v = a$). This can be done by performing a binary search over the finger tree representing F. We then split $F \cup \{x\}$ into two new funnels $F_1 = [u, \ldots, v, x]$ and $F_2 = [x, v, \ldots, w]$. If v belongs to $\pi(a,u)$ then we set $CUSP(F_1) := v, CUSP(F_2) := a$. If, on the other hand, v belongs to $\pi(a,w)$ then $CUSP(F_1) := a$, $CUSP(F_2) := v$.

(b) Set $\pi(s,x) := \pi(s,v) \cup vx$ (Actually we just store a back pointer from x to v. The collection of all these pointers will constitute the required shortest path tree Q_{sp}).

(c) If the line segment ux is a diagonal of P then we call recursively PATH(F_1).

(d) If the line segment wx is a diagonal of P then we call recursively PATH(F_2).

The main difference between our algorithm and the algorithm of Lee and Preparata [LP] is in the techniques for representation, searching and splitting of funnels. Since in our case the algorithm may have to continue recursively with both funnels F_1, F_2, it thus requires a funnel searching and splitting strategy that uses finger trees (and is thus subtler than the simple linear list representation used in [LP]), to obtain the desired linear time complexity.

The correctness of our algorithm is a direct consequence of the correctness of the algorithm of Lee and Preparata. To bound the time required by the algorithm we argue as follows. The dual tree T of the triangulation G of P has $n-2$ nodes. Without loss of generality, suppose s lies in just one triangle τ_0 of T, which we take to be the root of T. (If s lies in several triangles of G, as may well be the case when s is a vertex of P, then at least one of them will be bounded by an edge of P incident to s, and we can start the algorithm from that triangle. It is easily checked that the algorithm will then propagate correctly the funnel structure to all the other triangles containing s; note that the funnels for (the edges of) these triangles are all trivial.) Each node of T has 0, 1, or 2 children. Clearly, our algorithm is essentially a depth-first traversal of T. With each node ζ of T we associate two parameters:

m_ζ - the size (i.e. number of edges) of the funnel F at the time ζ is being processed.

n_ζ - the number of edges of P that bound triangles in the subtree of T rooted at ζ.

When our algorithm processes the node ζ of T, it splits its funnel into two parts and then appends a new edge to both parts to form the funnels of the children ζ_1, ζ_2 of ζ in T. If the split parts of the funnel of ζ contain m_1 and $m_2 = m_\zeta - m_1$ edges respectively, then $m_{\zeta_1} = m_1 + 1, m_{\zeta_2} = m_2 + 1$, and the (amortized) cost of processing ζ using finger trees is $K(\zeta) = O(\min(\log m_{\zeta_1}, \log m_{\zeta_2}))$. Note also that if ζ has two children ζ_1, ζ_2 then $n_{\zeta_1} + n_{\zeta_2} = n_\zeta$ and $n_{\zeta_1}, n_{\zeta_2} \geq 1$. If ζ has just one child ζ' then $n_{\zeta'} = n_\zeta - 1$, and if ζ is a leaf then $n_\zeta = 2$.

We begin our analysis by showing that the "direct costs" $K(\zeta)$ at nodes $\zeta \in T$ that have just one child or are leaves, sum up to at most $O(n)$. This is argued in much the same way as in [LP], and we omit the argument in this version.

Next consider the total direct costs at nodes having two children. We claim that it is sufficient to consider only cases in which m_ζ grows exactly by 1 at each node of T having a single child, because these cases provide maximal growth of the function m down the tree T. Under this additional assumption, we prove:

Lemma 2.1: Let $\zeta \in T$, and let the leaves of the subtree T_ζ of T rooted at ζ be $\eta_1, \ldots, \eta_k$. We then have

$$\sum_{j=1}^{k} m_{\eta_j} = m_\zeta + |T_\zeta|$$

where $|T_\zeta|$ is the number of edges in T_ζ.

Proof: Omitted in this version.

Corollary: In the same notations we have

$$m_\zeta \leq \sum_{j=1}^{k} m_{\eta_j}$$

Using these notations, we define $M_\zeta = \sum_{j=1}^{k} m_{\eta_j} \geq m_\zeta$. For each $\zeta \in T$ let $C(\zeta)$ denote the total cost of processing nodes with two children in the subtree of T rooted at ζ. Then plainly $C(\zeta) = 0$ if ζ is a leaf; $C(\zeta) = C(\zeta')$ if ζ has just one child ζ'; and

$$C(\zeta) = C(\zeta_1) + C(\zeta_2) + O(\min(\log m_{\zeta_1}, \log m_{\zeta_2}))$$

if ζ has two children ζ_1, ζ_2. In solving these recurrence formulas, we can clearly assume without loss of generality that each node in T is either a leaf or has two children. Moreover, replacing m_ζ by M_ζ in these formulas, we obtain the recurrence formula $C(\zeta) = 0$ if ζ is a leaf, and

$$C(\zeta) = C(\zeta_1) + C(\zeta_2) + O(\min(\log M_{\zeta_1}, \log M_{\zeta_2}))$$

if ζ has two children ζ_1, ζ_2. But $M_\zeta = M_{\zeta_1} + M_{\zeta_2}$, if ζ has children ζ_1, ζ_2 and $M_\zeta \geq 1$ for all nodes ζ. Hence if $C^*(k)$ is the maximal cost $C(\zeta)$ for any node ζ with $M_\zeta = k$, then we obtain the familiar formula

$$C^*(m) = \max_{1 \leq k \leq m-1} \{C^*(k) + C^*(m-k) + O(\min(\log k, \log(m-k)))\}$$

whose solution is easily seen to be

$$C^*(m) = O(m).$$

Finally, by Lemma 2.1 we have for the root τ_0 of T

$$M_{\tau_0} = m_{\tau_0} + |T| = n - 1$$

Thus the total complexity of the algorithm is

$$O(n) + O(M_{\tau_0}) = O(n).$$

Summing up our analysis, we obtain

Theorem 2.1: The shortest paths inside a simple polygon P from a fixed source vertex to all the other vertices of P can all be calculated in linear time.

An extended algorithm

The algorithm described above can be extended to produce additional information regarding shortest paths from the source vertex s to arbitrary points inside P. We describe such an extension that produces in linear time a partitioning of P into $O(n)$ disjoint triangular regions, such that each region consists of all points X, the shortest paths to which all pass through the same sequence of vertices of P. To do this, we first prove the following lemma.

Lemma 2.2: For each edge e of P, let $\Phi(e)$ denote the region bounded by e and by the funnel F_e. Then

(a) Let x be a point inside such a region $\Phi(e)$. Let v be a vertex in the corresponding funnel such that vx is tangent to the funnel (in the terminology of the algorithm described above). Then the shortest path from s to x is the concatenation of the shortest path from s to v with the segment vx.

(b) The interiors of the regions $\Phi(e)$ are all disjoint, and the total number of edges along their boundaries is $O(n)$.

Proof: Easy, and omitted in this version. □

Let e be an edge of P, and let $\Phi(e)$ be the corresponding region of P. By extending each edge of the funnel F_e until it hits e, we obtain a partitioning of $\Phi(e)$ into disjoint triangles, such that each triangle has two vertices lying on e and its third vertex (called its *apex*) belongs to the funnel F_e. Moreover, it follows immediately from Lemma 2.2(a) that if $x \in \Phi(e)$ belongs to the triangle with apex q then qx is tangent to the funnel at q, and thus the shortest path from s to x is the concatenation of the shortest path from s to q and the segment qx.

Hence the collection of all triangles obtained this way for all regions $\Phi(e)$ yields a partitioning of P into disjoint triangles, whose total number, by Lemma 2.2(b), is $O(n)$. We can then use any one of the linear-time algorithms of [Ki] or of [EGS] to preprocess this partitioning into a data structure that supports $O(\log n)$-time point location queries. The preceding argument then implies that for each target point x in P we can find in $O(\log n)$ time the last vertex q of P on the shortest path from s to x. This easily implies

Theorem 2.2: Given a simple polygon P with n sides, and some vertex s of P, one can preprocess P in linear time, such that, for each query target point x in P, the length of the shortest path from s to x can be calculated in $O(\log n)$ time, and the path itself can be calculated in time $O(\log n + k)$, where k is the number of segments from which this path is composed.

3. Visibility Within a Simple Polygon

In this section we study a collection of problems involving visibility within a simple polygon. These problems have been studied in various recent papers [EA], [Le], [AT], [CG], [EG], [LL], [As], and a variety of algorithms have been developed to solve them. Some of the simpler problems already have linear time solutions, whereas others have been given $O(n \log n)$ solutions. Here we present linear-time solutions for all these problems, again using the linear-time triangulation algorithm of [TV], and an interesting relationship between visibility and shortest path problems.

Let P be a simple polygon with n sides. The problems studied in this section, and solved in linear time, are

I. Given a point x inside P, calculate the *visibility polygon* $Vis(P,x)$ consisting of all points $y \in P$ that are *visible* from x (i.e. such that the segment xy is fully contained within P). (This problem is simpler than the subsequent ones, and in fact there exist known linear-time algorithms for it [EA], [Le].)

II. Given a segment e inside P, calculate the (weak) visibility polygon $Vis(P,e)$ consisting of all $y \in P$ that are visible from some point on e. (An $O(n \log n)$ solution is given in [CG]; Avis and Toussaint [AT] present a linear time algorithm for determining whether $Vis(P,e) = P$.)

III. Given such a segment e, preprocess P so that for each query ray r emanating from some point on e into P, the first intersection of r with the boundary of P can be calculated in $O(\log n)$ time. (Again, an $O(n \log n)$ solution is given in [CG].)

IV. As in III, preprocess P so that for each query point $x \in P$, the subsegment of e visible from x can be calculated in $O(\log n)$ time. (An $O(n \log n)$ solution is given in [CG].)

Since Problem I is already known to be solvable in linear time, we begin our analysis in this version of the paper with Problem II. Let A, B be the endpoints of e. It is well known (and easily checked) that in this case $Vis(P,e)$ is a simple polygon whose vertices are either

(i) vertices of P visible from e; or

(ii) "shadows" cast on the boundary of P by rays that emanate from A or from B and pass through a vertex of P visible from that endpoint; or

(iii) "shadows" cast by rays r that emanate from some interior point on e and pass through two vertices x, y of P, such that the complement of P lies on one side of r in the vicinity of x, and on the other side of r in the vicinity of y.

Moreover, for vertices of $Vis(P,e)$ of type (ii), if they are shadows cast by a ray from A (resp. B) passing through some vertex x of P, then the complement of P in the vicinity of x must lie on the exterior of the convex angle between e and Ax (resp. Bx); cf. Fig. 3.1.

Let $e' = CD$ be another edge of P. Let $\pi(X,Y)$ denote the shortest path inside P between the two points X and Y. We say that $\pi(A,C)$ is *outward convex* if the convex angles formed by successive segments of this path with the directed line AB keep increasing. A symmetric definition of outward convexity applies to shortest paths from B.

Lemma 3.1: If e' contains a point visible from e then (up to exchanging C and D) the two paths $\pi(A,C)$ and $\pi(B,D)$ are both outward convex.

Proof: Omitted in this version. □

In this case we call the union of $\pi(A,C)$ and $\pi(B,D)$ the *hourglass* for the pair (e,e').

Apply the shortest-path algorithm of Section 2 to the two source vertices A and B, and also compute on the fly, for each vertex C of P, whether the paths $\pi(A,C)$, $\pi(B,C)$ are outward convex. Let $e' = CD$ be another edge of P. If the two paths $\pi(A,C)$ and $\pi(B,D)$ are not both outward convex, and also the two paths $\pi(A,D)$ and $\pi(B,C)$ are not both outward convex, then by Lemma 3.1 e' is not visible from e. Thus suppose without loss of generality that the two paths $\pi(A,C)$ and $\pi(B,D)$ are outward convex. It is easy to see that the shortest path $\pi(A,D)$ must then be the concatenation of three subpaths: a subpath $\pi(A,X)$ of $\pi(A,C)$ (where X is some point lying on $\pi(A,C)$); a straight segment XY, where Y lies on $\pi(B,D)$; and the subpath $\pi(Y,D)$ of $\pi(B,D)$. Moreover XY must be a common tangent to both paths $\pi(A,C)$ and $\pi(B,D)$ (see Fig. 3.2). The path $\pi(B,C)$ has a symmetric structure of the form $\pi(B,W) \,||\, WZ \,||\, \pi(Z,C)$, for appropriate points $W \in \pi(B,D)$, $Z \in \pi(A,C)$. It now follows that the subsegment of e' visible from e is that which is delimited by the intersections of e' with the two lines XY and WZ. Note also that, in the terminology of Section 2, when the shortest-path algorithm is run with A as the source, the funnel $F_{e'}$ associated with the segment e' has X as its cusp, and Y as an adjacent vertex. Similarly, when the algorithm runs with B as the source, W is the cusp of the funnel $F_{e'}$ and Z is an adjacent vertex in that funnel.

These observations suggest a straightforward method for calculating the points X, Y, W and Z. That is, as we execute the shortest-paths algorithm with A as a source, and reach an edge $e' = CD$ of P for which the path $\pi(A,C)$ is outward convex, we simply take X to be the cusp of the current funnel, and Y to be the next vertex on that funnel on its portion between X and D. The points W and Z are then found in a completely symmetric manner when running the shortest path algorithm with B as a source. Hence, to calculate $Vis(P,e)$ we simply have to traverse the boundary of P in, say clockwise order, collecting all visible subsegments along this boundary in the manner explained above, and replacing contiguous non-visible portions of the boundary by straight segments connecting visible vertices with their appropriate shadows. Thus we have

Theorem 3.1: The visibility polygon $Vis(P,e)$ of P with respect to an edge e can be calculated in linear time.

Remark: A similar connection between shortest paths and visibility inside a simple polygon, and also some of the technical tools developed above, have also been obtained independently by Toussaint [To].

Next consider Problem IV. Let $X \in P$ be an arbitrary point that is visible from some point $z \in e$. An immediate generalization of Lemma 3.1 implies that both paths $\pi(A,X)$, $\pi(B,X)$ must be outward convex. The converse statement is also true, as is easily checked. Moreover, let the last straight segment in $\pi(A,X)$ (resp. in $\pi(B,X)$) be CX (resp. DX) for some vertex C (resp. D) of P. Then the portion of e visible from X is delimited by the intersections of the lines CX, DX with e.

Thus to preprocess P as required in Problem IV, we simply execute the (extended) shortest-path algorithm of Section 2 twice, once with A as a source, and once with B as a source. This yields two partitionings Π_1, Π_2 of P into zones of influence by the vertices of P, which we then further preprocess into two corresponding data structures that support $O(\log n)$ point location queries. This can be done in linear time, using the techniques in [Ki] or [EGS]. In addition, we store at each vertex C of P indications whether the paths $\pi(A,C)$, $\pi(B,C)$ are outward convex, and (as usual) also pointers to the two fathers of C in the two shortest-path trees produced by the two runs of the algorithm.

Now let $X \in P$ be a given query point. First locate X in the two partitions Π_1 and Π_2, and obtain the two corresponding influencing vertices C, D of P. Then, in constant time, we can test whether the paths $\pi(A,X) = \pi(A,C) \parallel CX$, $\pi(B,X) = \pi(B,D) \parallel DX$ are both outward convex, and if so, find the intersections of the lines CX, DX with e, to obtain the desired subsegment of e that is visible from X.

Remark: Problem IV is stronger than the corresponding problem P3 studied in [CG], in that here the query point X can be any point inside P, whereas in [CG] it is required to lie on the boundary of P.

Next consider Problem III. Here we make use of the duality between rays emanating from e and points in the *two-sided plane* (2SP for short), as described in [GRS], [CG]. Following [CG], we will produce a partitioning Π of the 2SP into convex regions, each region containing the duals of all rays emanating from e and hitting the same edge of P; this same partitioning is obtained in [CG] in $O(n \log n)$ time, and we show here how to obtain it in linear time.

To this end we make use of the analysis of Problem II given above. Let $e' = CD$ be another edge of P that contains points visible from e. As above, we can then assume that the two paths $\pi(A,C)$, $\pi(B,D)$ are both outward convex. Let XY, WZ be the two common tangents to these paths, where $X,Z \in \pi(A,C)$ and $W,Y \in \pi(B,D)$. Let $R(e')$ be the region in Π corresponding to e'. Then clearly the boundary of $R(e')$ consists of points that are duals of rays r emanating from e and hitting points on e', such that r passes through a vertex of P (which can be either one of the endpoints A,B,C,D, or another vertex of P that r "grazes" on its way from e to e'. It is clear from the preceding analysis that such a vertex must lie on one of the paths $\pi(A,C)$, $\pi(B,D)$, or, more precisely, on one of their subpaths $\pi(X,Z)$, $\pi(W,Y)$.

Moreover, it is also easily checked that the vertices of $R(e')$ correspond either to rays that pass through two vertices of P that are adjacent in one of the subpaths $\pi(X,Z)$, $\pi(W,Y)$, or to the two extreme rays XY, WZ. It is also easy to establish adjacency of the vertices of $R(e')$ along its boundary. Specifically, two such vertices must correspond to two rays passing respectively through FG and GH, where F, G, H are three vertices of P that are either adjacent along one of the paths $\pi(X,Z)$, $\pi(W,Y)$, or are such that $F=X$, $G=Y$, and H is adjacent to X along $\pi(X,Z)$ or adjacent to Y along $\pi(W,Y)$, or, symmetrically $F=W$, $G=Z$, and H is adjacent to one of these points along either $\pi(X,Z)$ or $\pi(W,Y)$.

The preceding arguments also imply that the total number of vertices in Π is at most proportional to the sum of the sizes of the funnels for the edges of P, which are obtained during execution of the shortest path algorithm of Section 2. It is easy to check that this sum is linear in n, so that Π has only $O(n)$ vertices (cf. also [CG]).

It is also easy to calculate adjacency of regions in Π. Specifically, it suffices to consider adjacency of regions near a vertex τ of Π. By the preceding analysis, τ is the dual of a ray r emanating from e and passing through two vertices G, H of P. Then we can apply a simple local, though somewhat lengthy, case analysis (which requires only $O(1)$ time), to enumerate all possible pairs of edges e', e'' whose regions $R(e')$, $R(e'')$ in Π are adjacent near τ; generally, each of these edges will either lie adjacent to G or H, or contain one of the endpoints of r, if this endpoint is different from G and H. We leave details of this case analysis to the reader.

All these observations imply that Π can be calculated in linear time from the output of the executions of the shortest path algorithm of Section 2 with A and B as sources, which themselves take only linear time. Having calculated Π, we next apply to it one of the linear time preprocessing algorithms of [Ki] or of [EGS] for point location, thus obtaining a data structure from which the region of Π containing (the dual of) any query ray r, and thus also the edge of P first hit by r, can be found in $O(\log n)$ time.

4. Linear preprocessing for the shooting problem in a simple polygon

In this section we show how, given a simple polygon P, we can build in linear time and space a data structure that solves the *shooting problem* for P, as defined by Chazelle and Guibas [CG, Section 3]. This problem asks for preprocessing P so that, given any point X inside P and any direction u, we can quickly compute the point $hit(X,u)$ where the ray emanating from X in direction u hits the boundary of P for the first time. If the polygon P has n sides, the method presented in [CG] solves the shooting problem in $O(\log n)$ time per query, after builing a structure in $O(n \log n)$ time that requires $O(n)$ space. The contribution of this section is to show how to build exactly the same shooting structure used by [CG] in linear time.

We start with a balanced decomposition S of our simple polygon P, obtained by recursively subdividing the polygon according to Chazelle's [Ch] polygon cutting theorem. The method presented in [Ch] runs in $O(n \log n)$ time, but we are able to compute the same decomposition tree in linear time. To do this, we note that Chazelle's algorithm actually calculates a balanced decomposition of the dual tree T of a triangulation of P (see following subsection for a precise definition of this notion). Since now Tarjan and Van Wyk's algorithm produces such a triangulation in linear time, it remains to show that the following tree decomposition step can also be done in linear time.

Balanced decomposition of a binary tree in linear time

Let T be a given binary tree with n nodes. If an edge e of T is removed, then T is partitioned into two subtrees. If we now similarly partition each of these subtrees and continue doing this recursively, until the fragments left are single nodes, we obtain another tree structure, which is known as a *decomposition* of T. Such a decomposition is called *balanced* if there is a positive constant α such that each time time a subtree T' is partitioned by the removal of an edge, the two fragments obtained have each size at least $\alpha|T'|$, where $|T'|$ denotes the size of T'. We now make two remarks on such decompositions. First of all, it is clear that in a balanced decomposition the longest path of splittings is of length $O(\log n)$. Secondly, it is known that in any binary tree there is an edge whose removal leaves two components, each with at least $\lfloor (n+1)/3 \rfloor$ nodes. Such an edge is always adjacent to the *centroid* of the tree and can be found in linear time [Ch]. As a result, a balanced hierarchical decomposition of T with $\alpha = 1/3$ can be found in $O(n \log n)$ time total, by applying the centroid partition to each fragment recursively (this was essentially the approach used by Chazelle [Ch]).

In this subsection we will show how the centroid edge for partitioning each fragment can be computed at a cost of only $O(\log n)$ operations, after some appropriate data structures have been set up. The overall cost for obtaining the balanced hierarchical decomposition will then be reduced to $O(n)$. Our method makes use of an auxiliary ternary tree A to facilitate the splitting. The tree A has the same nodes as T, but while T can be arbitrary, A is balanced in a strong sense: it has at most $\frac{n}{2^h}$ nodes of height h. In fact A itself represents a decomposition of T: If all ancestors in A of a node v (but not v itself) are deleted from T, one of the resulting fragments of T will contain exactly the same nodes as the subtree of A rooted at v.

The auxiliary tree A can be defined by first labelling the nodes of T with integers that can be computed in a single post-order traversal of T. Let us denote by b_v the label of node v. If v is a leaf, then $b_v = 1$. Otherwise, if w and z denote the chlidren of v, the label b_v is computed as follows. Let i be the position of the leftmost carry in the computation of $b_w + b_z + 1$. If we view these labels as bitstrings, we can define b_v so that its portion to the left of position i (including that position) is equal to the corresponding portion of $b_w + b_z + 1$; to the right of position i all bits of b_v are 0. The (unique) node whose label is divisible by the highest power of two is taken to be the root of A. Removal of this node generates at most three subtrees in T, which recursively define (at most) three subtrees in A of the root node. The computation of the above labels and the entire construction takes linear time. Details will be given in the full version of this paper.

Once we have the auxiliary tree A, we show that we can find the desired centroid edge for splitting T in logarithmic time (essentially by searching through A for that splitting edge). Moreover, in the same time bound, we can break A into two auxiliary trees, one for each of the two resulting fragments of T. By continuing this process all the way down to the leaves we obtain the desired balanced hierarchical decomposition S of T. In order to analyze the cost of this method more carefully, we note that auxiliary tree nodes cannot increase in height as the decomposition proceeds, and that the total number of nodes in A of height k is at most $\frac{n}{2^k}$. if the root of A has height h, then The cost of splitting an auxiliary tree whose root has height h is $O(h)$, and furthermore no node can appear as the root of such an auxilary tree more than $O(h)$ times total. Therefore the whole decomposition takes time

$$O(\sum_{k=0}^{\lfloor \log n \rfloor} k^2 \frac{n}{2^k}) = O(n) .$$

Remark: An alternative technique for obtaining such a balanced tree decomposition is obtained by using a simplified version of the dynamic tree data structure (as described in [Ta, Ch. 5]), which can also support logarithmic-cost tree-splitting operations. However the method sketched above is somewhat simpler and more direct.

The shooting problem

The balanced decomposition S of T as obtained above is most usefully thought of as a balanced tree. The leaves of that tree correspond to the triangles of an underlying triangulation T of P, while the internal nodes correspond to the diagonals in T. The root of S represents a diagonal d that partitions P into two subpolygons P_1 and P_2. Each of these subpolygons is in turn partitioned by a diagonal, and so on till we just have the triangles in T. Because S is a balanced decomposition, the maximum depth of any leaf is $O(\log n)$. We give each diagonal d of the triangulation an integer *label* $\lambda(d)$, which represents its level or depth in the tree structure. By convention the diagonal corresponding to the root s of the tree S has label 1; the children of the root have label 2, and so on.

The key idea for solving the shooting problem is to store, for certain pairs of diagonals (d_1,d_2), a representation of all lines that cut d_1 and d_2 but do not intersect the portion of P between d_1 and d_2. This set of lines is compactly represented by the hourglass for the pair (d_1,d_2), as discussed in Section 3. We will denote by Λ the list of all such pairs. It is easiest to construct these auxiliary structures from the bottom up.

Consider the following process, which we term the *merging process:* Start with the underlying triangulation T; its triangles can be considered as the leaves of the balanced tree S referred to above. For each triangle, at least two of whose sides are diagonals, add all pairs of bounding diagonals to the list Λ of pairs to be considered. Now remove all diagonals of the highest label. This creates new regions by merging pairs of old regions (triangles). It cannot happen that three or more regions get merged into one, since diagonals with the same label are never adjacent in T. If R_1, R_2 are two regions being merged, then add to the list Λ all pairs (d_1,d_2), where d_1 (resp. d_2) is a remaining diagonal bounding R_1 (resp. R_2). We continue this process, at each stage removing all diagonals of the next label and adding to the list Λ all new pairs of diagonals bounding one of the newly formed regions. Because of the structure of the balanced decomposition of P, it will never be the case that two diagonals with the same label become adjacent. Thus at each stage only pairs of regions get merged. Furthermore, any region that ever arises in this process will have at most a logarithmic number of diagonals on its boundary, as no two of them can have the same label.

In S, the balanced decomposition tree of P, each pair of diagonals (d_1,d_2) produced by the above process corresponds to an (ancestor, descendant) pair of nodes. This can be most easily seen by imagining the time-reversal of the merging process. A leaf of the current decomposition is expanded by introducing a new highest numbered diagonal g. Inductively we assume that the region corresponding to this leaf is a descendant of all diagonals bounding it. The introduction of the new diagonal clearly preserves this invariant. Furthermore, the new pairs of the form (e,g) that must now be added to Λ are clearly (ancestor, descendant) pairs. This proves our claim. From now on, whenever we write a pair (d_1,d_2) of diagonals in Λ, we will follow the convention that the first element is the ancestor and the second the descendant.

Let S^* denote the decomposition tree after the addition of all edges corresponding to the diagonal pairs in Λ (all edges of S naturally are represented in Λ). Let e be a particular diagonal of the decomposition which occurs with label $\lambda(e) = \mathrm{depth}(e)$. How many pairs of the form (e,g) can be in Λ (recall that g must be a descendant of e)? Note that g is uniquely determined by its level and the side of e it lies on - it is the "nearest" diagonal to e of the right level and on the appropriate side. Thus no more than $2(\tau(e) - \lambda(e))$ such pairs can exist in Λ, where $\tau(e)$ is the maximum depth in S of any node in the subtree rooted at e. Let $\mu(e)$ denote the total number of pairs in Λ that arise out of the subtree of S rooted at e. Then the above remarks prove that $\mu(e) \leq 2 \sum (\tau(g) - \lambda(g))$, where the sum is taken over all descendants g of e, including e itself. If s, l and r denote respectively the root of S and the root's left and right children, then we can write

$$\mu(s) = \mu(l) + \mu(r) + O(\log n) ,$$

where the last term is the contribution of s to Λ. Since S is balanced, there is some constant α, $0<\alpha<1/2$, such that the subtrees of each node x of S are in size at least the fraction α of the whole tree rooted at x. By standard techniques it then follows that the above recurrence has a solution of the form $\mu(s) = O(|S|) = O(n)$. This proves that Λ, and therefore S^*, has linear size.

As we remarked at the beginning, our aim is to associate with each pair (e,g) in Λ a visibility structure representing all lines cutting diagonals e and g, but not the portion of the polygon P between these diagonals. Such lines are constrained to avoid the two inwards concave chains defined by the hourglass illustrated in Figure 4.1. These chains are the convex hulls of the two polygonal paths joining e and g along P.

Suppose that we are in the midst of the diagonal-removing process and are currently working on diagonal f, whose removal merges two regions, one containing the diagonal e and the other the diagonal g. Suppose also that we have already computed the hourglasses for the pairs (e,f) and (f,g). Then in order to compute the hourglass for (e,g) it suffices to compute the outer common tangents of the corresponding pairs of concave chains in the hourglasses for (e,f) and (f,g). This is illustrated in figure 4.1. Note that since only two new edges are needed to form the new hourglass from the old ones, the total number of edges in all the hourglasses will be proportional to the number of diagonal pairs in Λ, i.e. it will be $O(n)$.

Recall our convention to write each diagonal pair (e,g) in Λ so that e is the ancestor and g the descendant. The size of the hourglass of (e,g) is bounded by the size $w(e,g)$ of the portion of P between e and g. Now we claim that $\log w(e,g) = O(\tau(e) - \lambda(e))$, for in a balanced decomposition the height of each subtree is logarithmic in its size. So the cost of computing the common tangents needed in the construction of the hourglass of e and g is $O(\tau(e) - \lambda(e))$. Therefore, if $\kappa(e)$ denotes the total cost of these common tangent computations for all pairs of diagonals in the subtree rooted at e, we have $\kappa(e) = O(\sum (\tau(g) - \lambda(g))^2)$, where the sum ranges over all descendants g of e (including e). By arguments entirely analogous to those used above we can write a recurrence of the form

$$\kappa(s) = \kappa(l) + \kappa(r) + O(\log^2 n)$$

for the total cost of computing the common tangents and conclude that it too is $O(n)$.

We will be traversing the search structure S^* from the top down in order to solve the shooting problem. The details are exactly as in [CG]. Because of this, we allocate each hourglass edge to the *highest* hourglass in S^* that has it as an edge - in other words to the last one formed in the above merging process. This means that at each stage, as we compute the hourglass of (e,g) from the hourglasses of (e,f) and (f,g), we need first to find the common tangents discussed above, and then split the old hourglasses where the common tangents touch them. The extremal pieces get joined by the tangent to form the new hourglass, while the inner pieces are left with the old hourglasses. See figuree 4.1 for an illustration. This splitting and joining can be implemented in the same order of magnitude time as the common tangent computation, if a balanced tree structure is used to represent the hourglasses. The details are straighforward and therefore omitted. This implies that the entire search structure S^* can be computed in linear time and that it occupies linear space. We have therefore shown that:

Theorem 4.1: Given a simple polygon P of n sides, it is possible in linear time and space to construct an auxiliary structure that allows us to solve the shooting problem for P in time $O(\log n)$ per query. These bounds are optimal.

Note: We have been able to avoid the use of finger trees in the above construction because we started out with a balanced decomposition. We could in fact have obtained a linear shooting structure in linear time from *any* decomposition S of P, whether balanced or not. This requires the use of finger trees in a manner analogous to that of Section 3. However the resulting shooting structure S^* can no longer guarantee logarithmic search time, as there is no logarithmic bound on its depth.

5. The Convex Rope Algorithm

As an additional application of the shortest path algorithm of Section 2, we consider the following *Convex Rope problem,* as posed by Peshkin and Sanderson [PS]: Let P be a simple polygon, and let s be a vertex of P lying on its convex hull. Let v be another vertex of P. The *clockwise convex rope* from s to v is the shortest polygonal path $(s = p_0, \ldots, p_m = v)$ starting at s and ending at v that does not enter the interior of P and that is clockwise convex, in the sense that the directed segment $p_i p_{i+1}$ lies to the right of the directed segment $p_{i-1} p_i$, for $i = 1, \ldots, m-1$. The counterclockwise convex rope from s to v is defined in a symmetric manner (see Fig. 5.1). Not all the vertices of P necessarily admit convex ropes from s. Those vertices v that do admit both clockwise and counterclockwise ropes from any such s are precisely those that are "visible from infinity" (cf. [PS]); calculation of these convex ropes is required in [PS] to plan reachable grasps of P by a simple robot arm. In [PS], an $O(n^2)$ algorithm is presented. Using the shortest path algorithm of Section 2, we obtain an improved algorithm running in linear time.

The convex rope problem can be solved as follows. Compute first the convex hull of P in linear time (cf. [PS], [GY]). The clockwise (resp. counterclockwise) convex rope from s to any vertex v on the convex hull can simply be calculated by moving along the convex hull in a clockwise (resp. counterclockwise) direction from s to v. For each vertex v not on the convex hull, v lies inside a simple polygon Q (a "bay" of P) bounded by some subsequence of the sides of P and by an edge of the convex hull that is not a side of P (see Fig 5.1; note also that the collection of all these bays can be found in linear time).

Let v_1, v_2 be the endpoints of this edge such that v_1 is reached first from s when moving along the convex hull in a clockwise direction. The clockwise (resp. counterclockwise) convex rope from s to v is then the clockwise convex rope from s to v_1 (resp. the counterclockwise convex rope from s to v_2) followed by the shortest path from v_1 to v (resp. from v_2 to v) within Q, provided that this shortest path is clockwise (resp. counterclockwise) convex (otherwise the required convex rope does not exist). Hence we can use the shortest path tree algorithm of Section 2 to calculate the shortest paths from v_1 and from v_2 to all the vertices of Q (and also to check whether these paths are convex in the required directions) in time $O(|Q|)$. Since the sum of the sizes of all the "bays" Q of P is $O(n)$, it follows that we can solve the convex rope problem in $O(n)$ time.

6. Conclusion

We have presented a collection of linear time algorithms for solving a variety of shortest paths and visibility problems inside a simple polygon, exploiting interesting relationships between these two types of problems. Our work has enriched the collection of problems solvable in linear time for simple polygons, but there are quite likely many additional problems for which linear time solutions can be developed. For example, Suri [Su] has recently extended our technique to solve in linear time the k-visibility problem, in which, given a simple polygon P and an edge e of P, we wish to partition P into disjoint subparts $P_1, P_2, \cdots$ such that P_1 contains all points in P directly visible from some point on e, P_2 contains all points in P visible from some point in P_1 but not from e, and so on.

Acknowledgement

The authors wish to thank Chris Van Wyk for useful discussions concerning the problems studied in this paper.

References:

[As] T. Asano, Efficient algorithms for finding the visibility polygon for a polygonal region with holes, manuscript, University of California at Berkeley.

[AT] D. Avis and G.T. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, *IEEE Trans. on Computers,* C-30 (1981) pp. 910-914.

[Ch] B. Chazelle, A theorem on polygon cutting with applications, *Proc. 23th IEEE Symp. on Foundations of Computer Science,* 1982, pp. 339-349.

[CG] B. Chazelle and L. Guibas, Visibility and intersection problems in plane geometry, *Proc. ACM Symposium on Computational Geometry,* 1985, pp. 135-146.

[EGS] H. Edelsbrunner, L. Guibas and J. Stolfi, Optimal point location in monotone subdivisions, DEC/SRC Tech. Rept. 2, 1984.

[EG] H.A. El Gindy, An efficient algorithm for computing the weak visibility polygon from an edge in simple polygons, manuscript, McGill University, 1984.

[EA] H.A. El Gindy and D. Avis, A linear algorithm for computing the visibility polygon from a point, *J. Algorithms* 2 (1981) pp. 186-197.

[Fi] S. Fisk, A short proof of Chvatal's watchman theorem, *J. Comb. Theory,* Ser. B, 24 (1978), p. 374.

[FM] A. Fournier and D.Y. Montuno, Triangulating simple polygons and equivalent problems, *ACM Trans. on Graphics* 3 (1984) (2) pp. 153-174.

[GJPT] M.R. Garey, D.S. Johnson, F.P. Preparata and R.E. Tarjan, Triangulating a simple polygon, *Information Processing Letters,* Vol. 7, 4, 1978, pp. 175-179.

[GMPR] L. Guibas, E. McCreight, M. Plass and J. Roberts, A new representation for linear lists, *Proc. 9th ACM Symp. on Theory of Computing,* 1977, pp. 49-60.

[GRS] L. Guibas, L. Ramshaw and J. Stolfi, A kinetic framework for computational geometry, *Proc. 24th IEEE Symp. on Foundations of Computer Science,* 1983, pp. 100-111.

[GY] R.L. Graham and F.F. Yao, Finding the convex hull of a simple polygon, *J. of Algorithms.*

[GK] D.H. Greene and D.E. Knuth, *Mathematics for the analysis of algorithms,* Birkhauser, Boston 1982.

[Hi] P.T. Highman, The ears of a polygon, *Inf. Proc. Letters* 15 (1982) pp. 196-198.

[HM] S. Huddleston and K. Mehlhorn, A new data structure for representing sorted lists, *Acta Informatica* 17 (1982) pp. 157-184.

[Ki] D. Kirkpatrick, Optimal search in planar subdivisions, *SIAM J. Computing,* 12 (1983) pp. 28-35.

[Le] D.T. Lee, Visibility of a simple polygon, *Comp. Vision, Graphics and Image Processing* 22 (1983) pp. 207-221.

[LL] D.T. Lee and A. Lin, Computing the visibility polygon from an edge, manuscript, Northwestern University, 1984.

[LP] D.T. Lee and F.P. Preparata, Euclidean shortest paths in the presence of rectilinear barriers, *Networks* Vol 14, 3, 1984, pp. 393-410.

[MA] D. McCallum and D. Avis, A linear algorithm for finding the convex hull of a simple polygon, *Inf. Proc. Letters* 9 (1979) pp. 201-206.

[PS] M.A. Peshkin and A.C. Sanderson, Reachable grasps on a polygon: The convex rope algorithm, *Technical Report CMU-RI-TR-85-6,* Carnegie-Mellon University, 1985 (to appear in *IEEE J. Robotics and Automation* 2 (1) (1986)).

[PS2] F.P. Preparata and K.J. Supowit, Testing a simple polygon for monotonicity, *Inf. Proc. Letters* 12 (1981) pp. 161-164.

[Su] S. Suri, Finding minimum link paths inside a simple polygon, to appear in *Comp. Vision, Graphics and Image Processing,* 1986.

[Ta] R.E. Tarjan, *Data Structures and Network Algorithms,* CBMS-NSF Regional Conference Series in Applied Mathematics, SIAM, Philadelphia 1983.

[TV] R.E. Tarjan and C. Van Wyk, A linear time algorithm for triangulating simple polygons, manuscript, October 1985.

[To] G. Toussaint, Shortest path solves edge-to-edge visibility in a polygon, Tech. Rept. SOCS-85.19, McGill University, 1985.

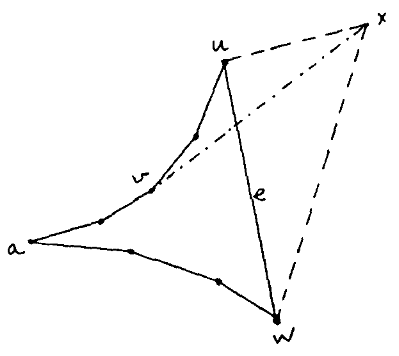

Fig. 2.1

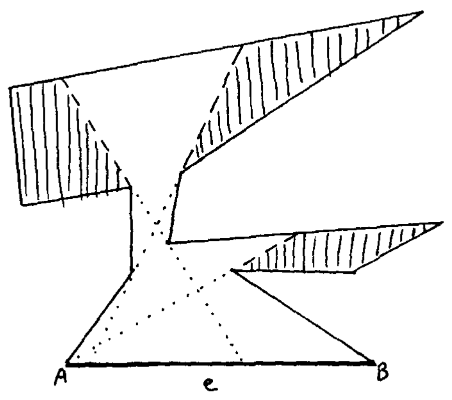

Fig. 3.1

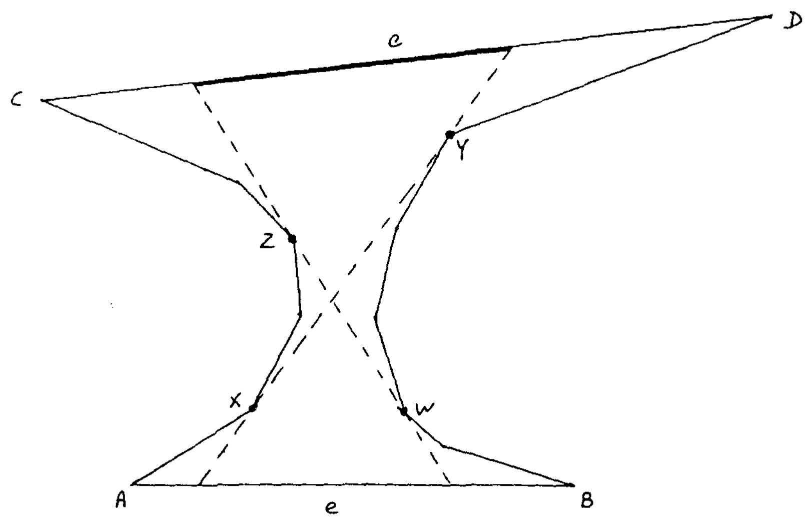

Fig. 3.2

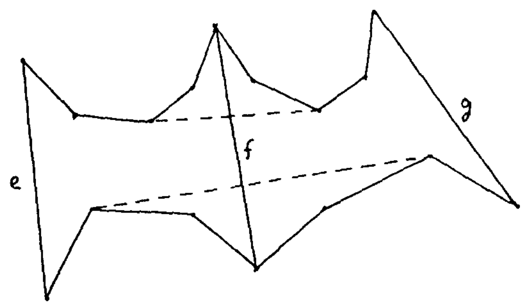

Fig. 4.1

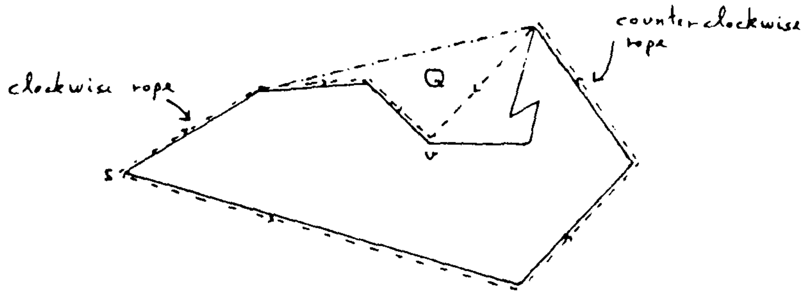

Fig. 5.1