# Minimizing Turns for Discrete Movement in the Interior of a Polygon

JOHN H. REIF AND JAMES A. STORER

*Abstract*—**The problem of movement in two-dimensional Euclidean space that is bounded by a (not necessarily convex) polygon is considered. Movement is restricted to be along straight line segments, and the objective is to minimize the number of bends or "turns" in a path. Most past work on this problem has addressed the movement between a source point and a destination point. An $O(n * \log(n))$ time algorithm is presented for computing a data structure that represents the minimal-turn paths from a source point to all other points in the polygon. An advantage of this algorithm is that it uses relatively simple data structures and is practical to implement. Another advantage is that it is easily generalized to accommodate the movement of a disk of radius $r > 0$.**

## I. Introduction

A VERY SIMPLE model of robot motion is one where the robot is capable of moving at a constant speed in a straight line and must stop and rotate to change directions. Under such a model, it is natural to consider the problem of finding a *minimal-turn path* between two points in Euclidean space which avoids a set of (not necessarily convex) polyhedral obstacles; Fig. 1 depicts such a situation. Another application of this problem arises in systems that communicate by light microwave, where a device of some sort is required to turn corners. A third application is the transportation of objects in space where once an object is moving, it requires little or no additional energy to continue moving but requires energy to stop or turn (e.g., see Niedringhaus [13]). A fourth application of minimal-turn computations is the computation of various types of visibility information.

In this paper we restrict our attention to the special case of two-dimensional Euclidean space where the obstacle space consists of the interior of an arbitrary polygon. Actually, all of our results apply to "generalized polygons," as depicted in Fig. 2, where one-dimensional obstacles are allowed.[1] However, to simplify the notation needed, we will not consider this point further. Finding minimal-turn paths is trivial for convex polygons where all points are visible from all others and becomes increasingly more difficult with polygons that are "more nonconvex"; for example, parts of a polygon could be "spirals" like the polygon shown in Fig. 3.

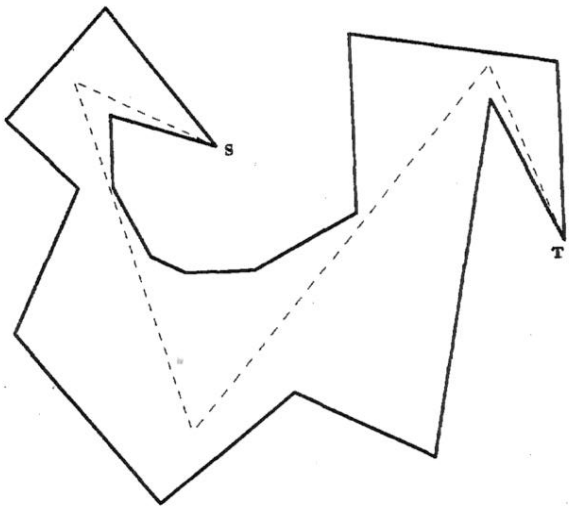



Fig. 1. Minimal-turn path.

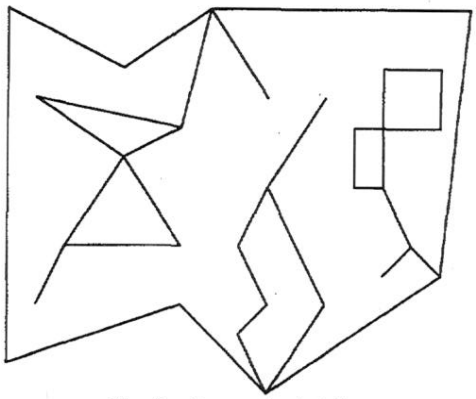


Fig. 2. Degenerate obstacles.

The *single-source single-destination* minimal-turn path problem for a polygon is to find a path between two given points in the polygon that avoids crossing any polygon edges and has a minimal number of turns among all such paths. In this paper we consider the more general *single-source multiple-destination* minimal-turn problem, which we shall refer to as simply the single-source problem. Here, we are given a source vertex $s$, and the task is to partition the polygon into regions whose points have equal minimal-turn paths back to the source. One of the advantages of considering the single-source multiple-destination problem instead of the single-source single-destination problem is that a series of queries can be answered efficiently. For example, consider an automated

supported in part by the Office of Naval Research under Grant N000-14-80-C-0547 and in part by the National Science Foundation under Grant DCR-8403244.

J. H. Reif was with the Aiken Computation Laboratory, Harvard University, Cambridge, MA 02138. He is now with Duke University, Durham, NC, USA.

J. A. Storer is with the Computer Science Department, Brandeis University, Waltham, MA 02254, USA.

IEEE Log Number 8715289.

[1] For a formal definition of generalized polygons see Reif and Storer [16].

**Preprint of paper appearing in IEEE Journal of Robotics and Automation, Vol. 3, No. 3, June 1987, pp. 182-193.**

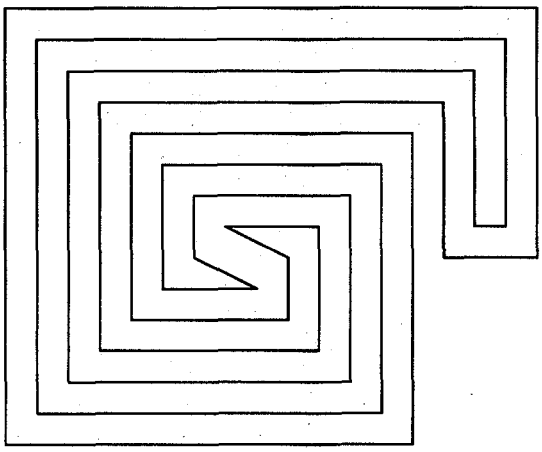


Fig. 3. Spiral polygon.

warehouse where a robot server must repeatedly proceed from a source point (the service window) to various points in the warehouse (to retrieve objects).

We let $n$ denote the number of polygon edges. The main result of this paper is an $O(n * \log(n))$ time algorithm which takes as input a polygon and produces as output a data structure that is a triangulation of the polygon with the following properties.

1) This data structure uses only $O(n)$ space.
2) All points within the interior of a given triangle require the same number of turns to reach to the source; the triangle is labeled with this number. All points on a given edge of a given triangle, excluding its endpoints, require the same number of turns to reach to the source; the number of turns for a point on an edge (this point may be an end point of the edge) to reach the source can be determined in $O(1)$ time.
3) Given a point $p$, the triangle that contains it can be determined in $O(\log(n))$ time.
4) For any point $p$ in the polygon, once it is determined in which triangle $p$ lies, a minimal-turn path between the source and $p$ can be output in time proportional to the number of turns.

It is important to note that once the data structure has been constructed, a sequence of queries about the distance of points from the source can be answered efficiently; that is, for a vertex $v$ it is possible in $O(1)$ time to determine the minimal number of turns for a path between $s$ and $v$ and for each point $p$ of the plane that is not a vertex, if it is known in which triangle $p$ lies, then the minimal number of turns for a path between $s$ and $p$ can be determined in $O(1)$ time. If it is not known in which triangle $p$ lies, then this can first be determined in $O(\log(n))$ time with standard point location techniques (e.g., Sarnak and Tarjan [17]).

In the last section we note how our algorithm can be generalized to produce a data structure for the minimal-turn movement of a disk; this gives rise to minimal-turn movement computations for arbitrary objects, to within the accuracy obtainable by enclosing the object with the smallest possible disk.

Reif and Storer [16] have considered the problem of computing paths of minimal length in two- and three-dimensional Euclidean space. In addition, [16] contains further references to a host of other work on this problem (the reader should also see the recent work of Mitchell *et al.* [28]. Although some of the tools developed for the shortest path problem are applicable to the one addressed here, the problems are quite different. The difficulty with minimizing turns is that unlike a minimum-length path, a minimum-turn path does not necessarily turn at obstacle vertices and is not in general unique.

Another area of research in computational geometry that is relevant to turn minimization is visibility; for example (as discussed at the end of Section V), the algorithm to be presented can easily be used to find all points in the polygon that are visible from a given point or visible from at least one point on a given line segment. For past work on visibility, see Guibas *et al.* [9], Asano *et al.* [1], Chazelle and Guibas [5], Suri and O'Rourke [23], and Avis and Toussaint [2]. As far as results that directly pertain to minimizing turns, recently and independent of the work presented here, ElGindy [7] and Suri [20] have discovered $O(n \log(n))$ algorithms for the less general two-point problem. Later Suri [21][2] claims that the bound of [20] for the two-point problem can be improved to linear by employing a linear time polygon triangulation algorithm claimed by Tarjan and Van Wyk [26] and a linear time edge-visibility algorithm claimed by Guibas *et al.* [9]. However, the polygon triangulation algorithm of Tarjan and Van Wyk [26] has recently been shown not to run in linear time.[3]

The primary advantage of the algorithm to be presented here over independent work mentioned earlier is that we address the more general single-source problem and that our algorithm uses simple data structures with practical implementations. It is also easily generalized to the movement of a disk of radius $r > 0$.

For other past work pertaining to minimizing turns, the reader may refer to a variety of work in VLSI layout theory. For example, Storer [24] and Tamassia [25] consider the problem of embedding graphs on the planar grid so as to minimize the number of bends introduced on edges, and Pinter [14] considers the problem of minimizing jogs when routing edges across a channel.

Section II contains basic definitions. Section III defines the notion of a funnel, which is the basis of our algorithm. Section IV presents our algorithm for a restricted class of polygons called *serpentine* polygons. Serpentine polygons are ones which have a triangulation such that the triangles can be visited by a simple path; that is, the polygon can be viewed as having a "snake" shape. Serpentine polygons are more easily handled because there are no "choices" when walking through a triangulation of the polygon. In Section IV we also make the simplifying assumption that the source point is a vertex of the polygon that has no edges of the triangulation incident to it (i.e., the source is one of the ends of the

[2] This has since appeared as Suri [22].

[3] However, it does appear that it runs in time $O(n \log \log(n))$. We have been informed by a referee of this paper that a forthcoming technical report of Suri employs this bound.

"snake"). Section V presents the algorithm for arbitrary polygons with an arbitrary location of the source point. Finally, Section VI notes how our algorithm can be generalized to handle the movement of a disk.

For all sections to follow we assume the conventional log-cost RAM model of computation augmented with the following:

1) both integer and real variables;
2) arithmetic operations of the form $x + y$, $x - y$, $x * y$, $x/y$ and tests of the form $x \geq 0$; each charged unit time cost;
3) the square root operation, which is charged unit cost.

## II. Definitions

This section contains the basic definitions necessary to state formally the single-source minimum-turn problem for a polygon. To simplify notation, we consider the plane to be the subset of three-dimensional space given by $\{(x, y, z): z = 0\}$; this allows us to adopt (0, 0, 1) as a reference point sitting above the plane and assume that the notions of clockwise and counter-clockwise are defined relative to this reference point.

*Definition 1:* A *straight-line planar map* is a finite connected undirected graph $G$ that has been embedded on the plane via the following data structure. Associated with each vertex $v$ of $G$ is a distinct pair of real numbers $i \geq 0$ and $j \geq 0$ called the *coordinates* of $v$ along with an *adjacency list* of the edges incident to $v$. The *embedding* of a vertex $v$ of $G$ is the point in the plane located at the coordinates of $v$. Associated with each edge $(v, w)$ of $G$ is a straight-line segment $\alpha(v, w)$ which connects the embedding of $v$ and the embedding of $w$, called the embedding of the edge $(v, w)$. It must be that the following conditions hold.

1) The embeddings of the two edges cannot intersect except possibly at their endpoints.
2) If the embeddings of two edges $(u, v)$ and $(v, w)$ are colinear, then vertex $v$ has degree $\geq 3$.
3) A clockwise ordering of the embedded edges incident to an embedded vertex $v$, forms a cyclic permutation of the adjacency list for the vertex $v$.

The *faces* of a planar map $G$ are the open sets formed by the connected regions of the plane when the embedding of $G$ is removed. The face with infinite area is called the *external* face, all other faces are *internal* faces.

Given a planar map $G$, a point of the two-dimensional Euclidean plane is said to be *a point of* $G$ if it is not contained in the external face of $G$.

A *polygon* is a straight-line planar map such that each vertex has degree exactly 2. Given two points $p$ and $q$ of a polygon, a path between $p$ and $q$ is a finite sequence of straight line segments that connects $p$ and $q$ and does not intersect the exterior face of the polygon.

A *border path* is a path such that its endpoints are vertices of the polygon and its line segments are edges of the polygon.

A few notes concerning the above definition are in order.

- From this point on, to simplify the presentation, we shall usually ignore the distinction between a vertex in $G$ and its embedding as a point in the plane or an edge in $G$ and its embedding in the plane as a line segment.
- We assume the reader to be familiar with basic properties of planar maps. In particular, every planar map has exactly one exterior face and the number of edges and faces is bounded by a constant times the number of vertices.[4]
- Our notion of a polygon is often called a *simple polygon* in the literature.
- As a consequence of the above definition, when we talk about a "point of a polygon," that includes points in the internal face.
- We shall frequently ignore the distinction between the polygon itself and the polygon together with its internal face.
- We have defined a path so as to allow motion of a point to be in contact with the polygon edges. Motion of a point that avoids the polygon edges by some tolerance $\epsilon > 0$ can be modeled by moving a disk of diameter $\epsilon$ (see Section VI).

*Definition 2:* The *number of turns* on a path between $p$ and $q$ is one less than the number of line segments on the path. A minimal-turn path between $p$ and $q$ is a path between $p$ and $q$ that has no more turns than any other path between $p$ and $q$.

The following lemma guarantees that a minimal-turn path always exists and provides worst-case bounds on the number of turns.

*Lemma 1:* For any polygon $P$ with $n$ edges and any two points $p$ and $q$ in $P$, there always exists a path between $p$ and $q$ with at most $\lfloor n/2 \rfloor - 1$ turns. Furthermore, this bound is tight; that is, for any integer $n \geq 3$, there are polygons of $n$ edges which have a pair of points[5] such that any path between them requires $\lfloor n/2 \rfloor - 1$ turns.

*Proof:* Let us first show that $\lfloor n/2 \rfloor - 1$ is an upper bound. We can assume that the polygon has at least four edges (since any two points in a triangle can be connected with zero turns). It is easy to verify that any point in a polygon must be visible from at least three vertices of $P$. Given this, let $p_1$, $p_2$, and $p_3$ be three vertices of $P$ that are visible from $p$, and let $q_1$, $q_2$, and $q_3$ be three vertices of $P$ that are visible from $q$. If any of $p_1$, $p_2$, and $p_3$ are the same as one of $q_1$, $q_2$, and $q_3$, then the minimal turn distance between $p$ and $q$ is at most 1, and we are done (since we are assuming the polygon to have at least four edges). Otherwise, there exist two distinct border paths $X$ and $Y$ of $P$ that include exactly one of the vertices $p_1$, $p_2$, and $p_3$ and exactly one of the vertices $q_1$, $q_2$, and $q_3$. Then one of $X$ and $Y$ has at most $\lfloor (n - 6)/2 \rfloor$ vertices that are not endpoints, and thus we can form a path from $p$ to $q$ that has at most $\lfloor (n - 6)/2 \rfloor + 2 = \lfloor n/2 \rfloor - 1$ turns.

To show that $\lfloor n/2 \rfloor - 1$ can be achieved infinitely often, we can consider the class of "zig-zag" polygons; Fig. 4 shows zig-zag polygons of order 1, 2, 3, and 4. There are many ways

[4] For any planar map $G$ with vertex set $V$ and edge set $E$, it is always true that $|V| \leq |E| \leq 3|V| - 6$ and the number of faces in $|E| - |V| + 2$. See, for example, Berge [3].

[5] Actually, infinitely many polygons of $n$ edges with infinitely many pairs of points, since for the construction to be presented, a given polygon and a given pair can always be perturbed slightly.

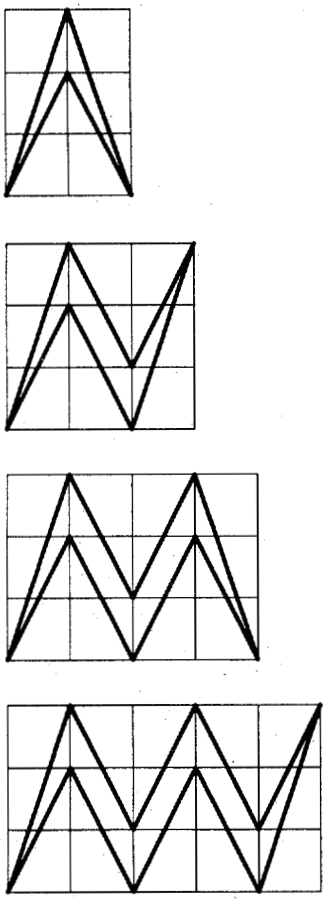


Fig. 4. Zig-zag polygons.

to define a zig-zag polygon of order $k$; one is by appropriately connecting vertices on the 3 by $k + 1$ planar mesh (as was done for Fig. 4). We will not bother with a formal definition here since it should be clear to the reader from inspection of Fig. 4 what a zig-zag polygon of order $k$ is, that it has $2(k + 1)$ vertices, that a path between the ends of a zig-zag of order $k$ requires $k$ turns, and that $\lfloor (2(k + 1))2 \rfloor - 1 = k$. From this it follows that for even $n \geq 3$ there are polygons of $n$ edges that achieve the bound of $\lfloor n/2 \rfloor - 1$. For odd $n > 3$ ($n = 3$ is trivial), we can simply add a "dummy" node to a zig-zag of order $[(n - 1)/2] - 1$ (by deforming slightly one of the edges into two edges).

*Definition 3:* The two-dimensional single-source single-destination minimal-turn path problem for a polygon is

input a polygon $P$, a *source point* $s$, and a *destination point* $t$ ($s$ and $t$ are points of $P$);
output a minimal-turn path from $s$ to $t$.

The previous definition is a special case of the problem addressed in this paper, which we now define with the following two definitions.

*Definition 4:* A *triangulation of a polygon* $P$ is a straight-line planar map $P^+$ such that the following conditions hold.

1) The edges of $P$ are a subset of the edges of $P^+$.
2) The external face of $P^+$ is identical to the external face of $P$.
3) All internal faces of $P^+$ are triangles; that is, they are bounded by exactly three edges.

A triangulation of a polygon is *chordal* if in addition the following condition holds.

4) The set of vertices of $P^+$ is the same as the set of vertices of $P$.

The *border* of $P^+$ is $P$. All edges of $P^+$ that are in $P$ are called *border* edges and all other edges of $P^+$ are called *triangle* edges; for chordal triangulations, triangle edges are also referred to as *chords*.

It should be noted that, with the above definition, straight-line planar maps such as Fig. 5(a) are not legal triangulations. Fig. 5(b) shows a nonchordal triangulation of the same polygon as Fig. 5(a) and Fig. 5(c) shows a chordal triangulation. It is common in the literature for the term "triangulation" to mean what we call "chordal triangulation"; we have defined the more general notion because although a chordal triangulation of the polygon suffices for the input to our algorithm, it will in general produce a nonchordal triangulation as output.

*Definition 5:* The two-dimensional single-source multiple-destination minimal-turn path problem for a polygon, which we henceforth refer to as simply the two-dimensional single-source problem, is

input a polygon $P$ of $n$ edges and a source point $s$ ($s$ is a point of $P$);
output a data structure representing a straight-line planar map $P^+$ with the following properties:

1) $P^+$ is a triangulation of $P$;
2) $P^+$ has size $O(n)$;
3) all points within a given triangle of $P^+$ require the same number of turns to reach to the source; the triangle is labeled with this number; all points on a given edge of a given triangle, excluding its endpoints, require the same number of turns to reach to the source; the number of turns for a point on an edge (this point may be an end point of the edge) to reach the source can be determined in $O(1)$ time;
4) Given a point $p$, the triangle that contains it can be determined in $O(\log (n))$ time;
5) For any point $p$ in the polygon, once it is determined in which triangle $p$ lies, a minimal-turn path between the source and $p$ can be output in time proportional to the number of turns.

As mentioned in the Introduction, it will be convenient to first develop an algorithm for the single-source problem that works for a restricted class of polygons given by the following definition.

*Definition 6:* A triangle of a triangulated polygon is *serpentine* if it contains at least one border edge. A triangulation of a polygon is *serpentine* if it is chordal and all triangles are serpentine. A polygon is serpentine if it has at least one serpentine triangulation.

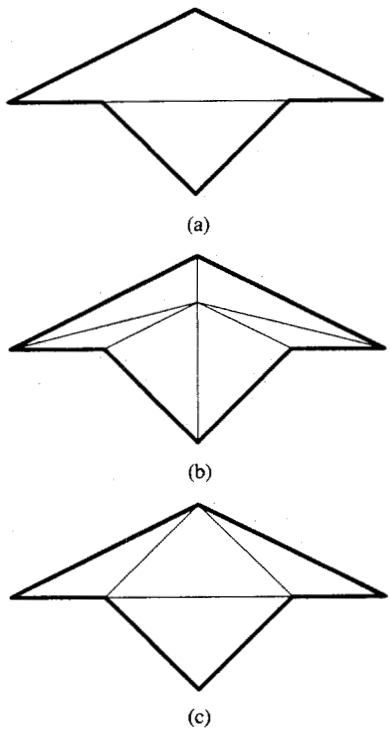



Fig. 5. (a) Illegal triangulation. (b) Legal triangulation. (c) Chordal triangulation.

The polygons of Figs. 1, 3, 4, and 5 are examples of serpentine polygons (although the triangulations shown in Fig. 5(b) and Fig. 5(c) are not serpentine). Fig. 6 shows a serpentine triangulation (the thick edges are the polygon and the thin edges are the chords). Fig. 7 shows a nonserpentine polygon; the thin edges show the only possible triangulation.

The motivation for the term "serpentine" comes from consideration of the geometric dual[6] of the serpentine triangulation. If we ignore all edges incident to the external face, then the dual is a simple path. That is, visiting all triangles of a serpentine triangulation requires no "decisions."

## III. Funnels

In this section we define the notion of a *funnel*, which is fundamental to our single-source algorithm. To motivate the material of this section, let us digress briefly to describe the "basic" idea behind the single-source algorithm to be presented in the following two sections. If one imagines a "cone of visibility" coming from the source, it will encompass some of the polygon and chop off other parts. The boundary that separates a chopped-off part from the rest of the polygon is a straight line. One can now imagine a "generalized cone of visibility" that comes from such a straight line and further cuts up the polygon; generalized cones of visibility can then come from these boundaries to chop the polygon up further, and so on. That is, the idea is to partition the polygon into regions of visibility. The cost 0 region consists of all of those points visible from the source, the cost 1 regions consist of points that are visible from the lines used to partition the cost 0 region,

[6] The graph with one vertex for each face and an edge between every pair of vertices that correspond to adjacent faces.

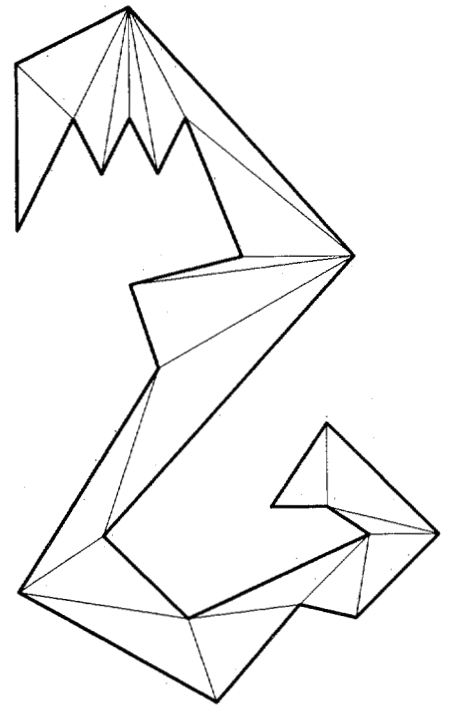


Fig. 6. Serpentine triangulation.

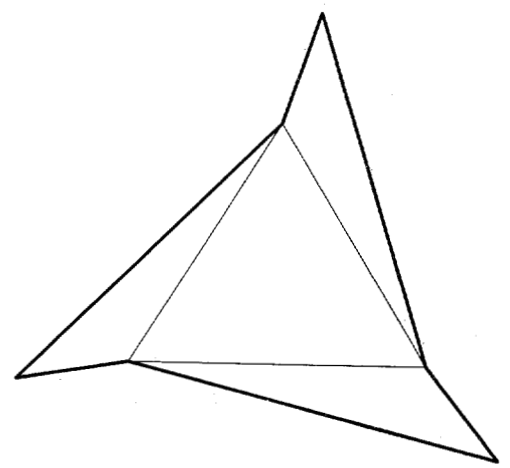


Fig. 7. Nonserpentine polygon with its only triangulation.

etc.; Fig. 8 depicts this. Essentially, these generalized cones of visibility are what we will define to be funnels.

Before formally defining a funnel, we first present a definition and a lemma. The proof of this lemma is straightforward (and well known) but is included here to motivate the definition of a funnel further.

*Definition 7:* A *straight-line* border path is a border path such that all of its line segments are colinear.

*Lemma 2:* Let $B$ be a straight-line border path of a polygon $P$. Let $S$ be the set of points of $P$ that are visible from at least one point of $B$, and let $T$ be the set of points of $P$ that are not visible from $B$. Then there is a set of straight-line segments $X$ such that the removal of the points of $X$ from $P$ (and the internal face of $P$) partitions $P$ into a number of disjoint connected sets $P_1 \cdots P_k$ such that $S$ is the union of $P_1$ and $X$, and $T$ is the union of $P_2$ through $P_k$. Furthermore, let $p$ be any point of $T$. Then there is a point $q$ on $B$ such that a minimal-

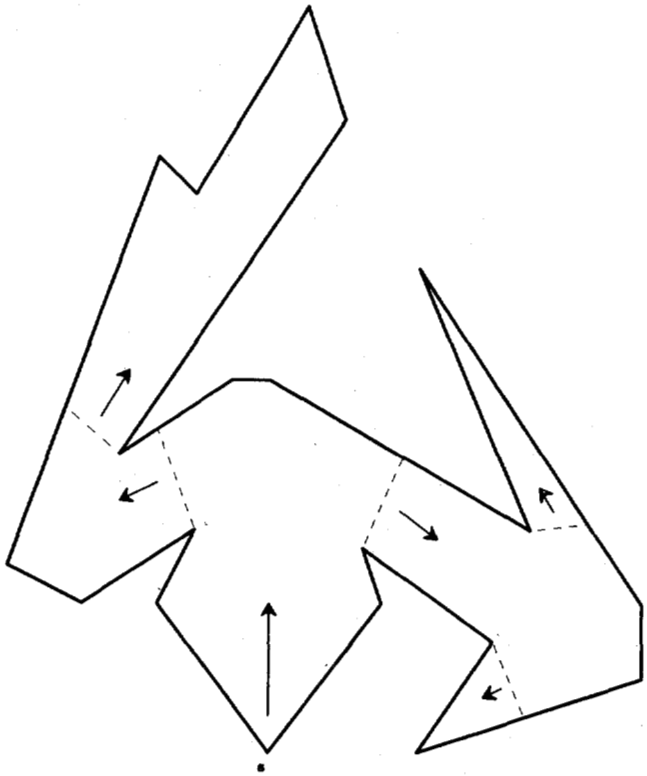


Fig. 8. Basic idea: partitioning into regions of visibility.

turn path exists from $p$ to $q$ that makes its last turn at one of the points of $X$.

*Proof:* Asserting the existence of the set of line segments $X$ is equivalent to asserting that the border of the set of points of $P$ that are visible from at least one point of $B$ forms a polygon (with an interior that is a subset of the interior of $P$); this is easily shown with standard techniques (e.g., Valentine [27]). As for the second claim of the lemma, we can first observe that any minimal-turn path, call it $Z$, from $p$ to any point $q$ of $B$ must turn at some point in $S$; to see this, suppose that this was not the case, and let $a$ be the last turn before $q$ (or $p$ if the path from $p$ to $q$ has no turns). Then $a$ is a point in $T$ that is visible from $B$, a contradiction to the definition of $T$. We can next observe that $Z$ must have exactly one turn in $S$, since to have more than one turn in $S$ would clearly contradict the fact that the path has a minimal number of turns. We can now replace this turn with a different turn as follows. Let $b$ be the intersection of $Z$ with $X$. Then there must be a point $c$ of $B$ such that the line segment $(b, c)$ is in $P$ (since $b$ is in $S$). However, this implies that the path consisting of the portion of $Z$ between $p$ and $x$ together with $(b, c)$ must have the same number of turns as $Z$; this path turns on $X$, as claimed by the lemma.

*Definition 8:* A *funnel* is a triple $F = (P, B, \alpha)$ where $P$ is a polygon that is triangularized by chords, $B$, the *base* of the funnel, is a straight-line border path of $P$, and $\alpha$, the *cost* of the funnel, is a nonnegative integer.

Fig. 9(a) and (b) are two examples of funnels; both are incomplete in that not all of the polygon is shown. For both figures, the thick edges are polygon edges, thin lines are triangle edges, the dashed line from point $a$ to point $b$ is the

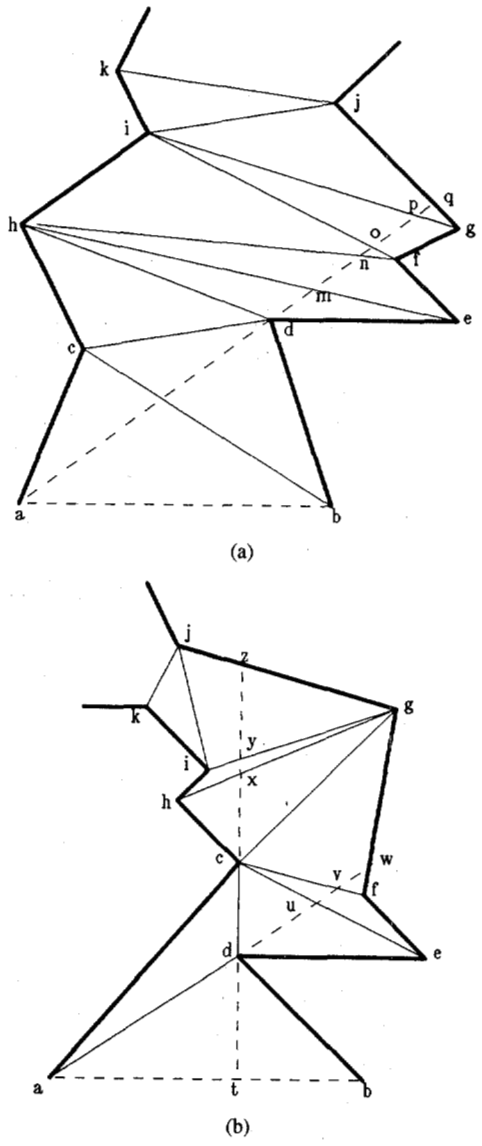



Fig. 9. Funnels.

funnel base. We now shall discuss the significance of the other dashed lines. Consider first Fig. 9(a). All of the polygon that is shown in Fig. 9(a) is visible from the funnel base except for the portion of the polygon that is cut off by the dashed line segment from point $d$ to point $q$. Furthermore, a new funnel can be formed from this cut off portion of the polygon by adding the line segments $(m, f)$ and $(o, g)$ to triangulate it fully (we could just as well use segments $(n, e)$ and $(p, f)$), taking the segment from $d$ to $q$ as the base, and assigning this funnel cost 1 greater than the cost of the original funnel. The motivation for doing this is that the new funnel base going from $d$ to $q$ corresponds to partition lines like those given in Lemma 2, and indicated in Fig. 8. Similarly, we can consider Fig. 9(b). Here, all of the polygon that is shown is visible from the funnel base except for the portions of the polygon that are cut off by the dashed line segment from $c$ to $z$ and the dashed line segment from $d$ to $w$, and similar to Fig. 9(a), we can form new funnels from these cuts.

Intuitively, the dashed line segment from $d$ to $q$ in Fig. 9(a) and the dashed line segments from $c$ to $z$ and $d$ to $w$ in Fig.

9(b) bound the area that is visible from the funnel base; note that Fig. 9(a) has only one side since the other side of the portion of the polygon that is shown is completely visible. However, in general, the visible region can extend arbitrarily far into the polygon and can cut off many disjoint pieces of the polygon. For example, any number of things could happen in the part of the polygon that is not shown in Fig. 9(a). Hence our formal definition of a *funnel side* will include only what is essentially the beginning of the cuts; as we move through the polygon, what is currently the funnel side will be constantly adjusted. To compute the funnel sides, we shall use the notion of a "funnel top."

*Definition 9:* Let $F = (P, B, \alpha)$ be a funnel, and let $a$ and $b$ be the endpoints of $B$. If there is a chord of $P$ which does not share a point with $B$ but is an edge of a triangle that does share a point with $B$, then $F$ is said to be *open* and this chord is called a *top* of $F$; otherwise $F$ is said to be closed.

It is possible for a funnel to have more than one top. However, the following definition and lemma give a sufficient condition for a funnel to have a unique top. This lemma will be employed by the algorithm to be presented in the next section.

*Definition 10:* Let $P$ be a polygon. A *cut* of $P$ is a straight-line segment $C$ such that its endpoints are distinct and are either vertices or lie on edges of $P$ and all other points of $C$ lie in the interior of $P$. If $P$ is triangulated and $C$ is a cut of $P$, $C$ is nontrivial if the two polygons formed from $P$ by $C$[7] each contain all of the interior of at least one triangle of the triangulation of $P$.

*Lemma 3.2:* Let $P$ be a polygon given with a serpentine triangulation. Let $C$ be a nontrivial cut of $P$. Let $Q$ be either of the two polygons obtained by cutting $P$ along $C$. Then $Q$ can be augmented with additional chords to form a serpentine triangulation. Furthermore, if $\alpha$ is an arbitrary integer, then the funnel $(Q, C, \alpha)$ has a unique top.

*Proof:* First, let us verify that $Q$ can be augmented with chords to form a serpentine triangulation. Consider a particular triangle of $P$ that is cut by $C$. If this cut forms two triangles, then we are done since they both have an edge on $C$ (which is now a border path of $Q$). The other possibility is that the triangle is divided into one triangle and one quadrilateral. At least one of the edges of this quadrilateral lies on the border of $P$ and at least one lies on $C$; call these two edges $A$ and $B$. If $A$ and $B$ intersect, then the unique chord of the quadrilateral that has as one endpoint this intersection will suffice. Otherwise, either of the two possible chords will suffice.

For the second half of the lemma, consider the geometric dual of $P$ less the edges incident to the external face; call this $D$. Suppose that $Q$ had two (or more) tops. Then on one side of $C$ we have the two tops and on the other we must have at least one triangle (since $C$ is nontrivial). However, this implies that $D$ cannot be a simple path, contradicting the fact that the triangulation is serpentine.

Given the notion of a funnel top, we are now ready formally to define the funnel sides. Note that the above definition states how the funnel top is initially defined. In general, our algorithm will start with this initial value of the funnel top and then move it from chord to chord through the polygon, at each state applying the following definition to compute the current value of the funnel sides. In reading the following definition, it may be useful to refer to Fig. 10.

*Definition 11:* Suppose that a funnel $F = (P, B, \alpha)$ is open and has a unique top. Let $a$ and $b$ be the endpoints of $B$. Let $T = (c, d)$ be the funnel top where $c$ is the endpoint of $T$ that is connected to $a$ via a border path that does not pass through $d$ (and $d$ is the endpoint of $T$ that is connected to $b$ via a border path that does not pass through $c$). Let $L$ be the subsegment of $(a, b)$ that is visible from $(c, d)$ where $x$ is the end point of $L$ that is closest to $a$ and $y$ is the other end point of $L$ (the one closest to $b$). The *sides* of $F$ are the segments $(x, d)$ and $(y, c)$.

Some terms that relate to funnel sides are the following. Let $(u, v)$ be a funnel side where $u$ is a point of $B$ and $v$ is an endpoint of $T$; let $w$ denote the other endpoint of $T$. The *funnel boundary* corresponding to $(u, v)$ is the border path of $P$ that connects $w$ to the base and does not pass through $v$. The *critical vertex* for $(u, v)$ is the first vertex (that is not $v$) of the funnel boundary corresponding to $(u, v)$ that is intersected when traversing this side from $v$ to $u$ (it may be that the critical point is $u$, in which case $u$ must be an endpoint of $B$).

In Fig. 10, $(c, d)$ is the funnel top, $(x, d)$ is a funnel side with critical vertex $j$ (where $(a, g, h, i, j, c)$ is the associated funnel boundary), and $(y, c)$ is a funnel side with critical vertex $k$ (where $(b, l, k, d)$ is the associated funnel boundary). For an additional illustration of the funnel top and sides, we can refer to Fig. 9. In Fig. 9(a), $(c, d)$ is the funnel top, $(a, d)$ is a side with critical vertex $a$ (where $(a, c)$ is the associated funnel boundary), and $(b, c)$ is a side with critical vertex $b$ (where $(b, d)$ is the associated funnel boundary). In Fig. 9(b), $(c, d)$ is the funnel top, $(a, d)$ is a side with critical vertex $a$ (where $(a, c)$ is the associated funnel boundary), and $(t, c)$ is a side with critical vertex $d$ (where $(b, d)$ is the associated funnel boundary).

The following definition describes how funnel sides are extended to "chop off" parts of the polygon that are not visible from the funnel base.

*Definition 12:* Let $F = (P, B, c)$, be an open funnel that has a unique top. Let $X$ and $Y$ be its sides. Let $\hat{X}$ be the infinite line that contains $X$ as a subsegment. The *bypass base* corresponding to $X$ is the subsegment of $\hat{X}$ that intersects $P$ (and its interior faces) but does not intersect $X$ except at one of its end points. The bypass base corresponding to $Y$ is similarly defined.

As an example of funnel bypass bases, in Fig. 9(a) the only bypass base is the path $(d, m, n, o, p, q)$ and in Fig. 9(b) the bypass bases are the paths $(d, u, v, w)$ and $(c, x, y, z)$.

The following definition provides the terms needed to describe how the funnel top is moved.

*Definition 13:* Let $F = (P, B, \alpha)$ be an open funnel with a unique top such that its bypass bases have already been computed; that is, the edges of the bypass bases (if any) have been inserted, additional chords have been supplied where necessary (with the construction of Lemma 3), and the edges of the bypass bases are now treated as border edges for the

[7] A cut always partitions the polygon into two polygons.

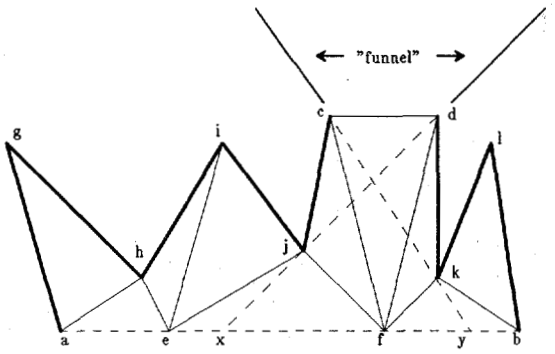



Fig. 10. Computation of funnel sides.

purposes of this definition. The *leading triangle* of $F$ is that triangle of $P$ that has as one of its edges the top of $F$ and has a vertex that is not in the region bounded by the base of $F$ and the top of $F$; this vertex is called the *leading vertex* of $F$. The *leading edges* of $F$ are those edges of the leading triangle of $F$ that are not the top of $F$ and not a border edge.

For examples of leading triangles and edges, in Fig. 9(a), the leading triangle is $(c, d, h)$ and the leading edge is $(d, h)$. In Fig. 9(b), the leading triangle is $(c, d, u)$ and the leading edge is $(c, u)$.

For the above definition, it is important that for serpentine triangulations there is at most one leading edge. For closed funnels, the notions of funnel sides, bypass bases, leading triangle, and leading edge will not be needed.

## IV. Serpentine Polygons

In this section, we present an algorithm for the single-source problem for the special case that we have a serpentine triangulation of the polygon and that the source is a vertex of degree 2 in this triangulation (i.e., it is at one of the two ends of the snake). In addition, we shall not bother until the next section to augment the data structure so that minimal-turn paths can be output in time proportional to the number of turns; the algorithm to be presented in this section will only label triangles with the cost of such a path. As illustrated in Fig. 8, our approach is to start with an initial funnel consisting of the entire triangulated polygon, the source vertex as the base (a degenerate instance of a funnel base is a single point), and cost 0, and then expand outward, cutting the polygon along visibility lines to form new funnel bases (the funnel bypass bases).

Algorithm 1 is a high-level description of how to label the internal faces correctly in linear time, when a serpentine triangulation is given as input. Note that labels on the faces are all that is needed since given a point on an edge, its minimal-turn distance back to the source is simply the minimum of the labels on the two or three (in the case of a vertex) faces that are adjacent to it.

*Algorithm 1—Simplified Single-Source Algorithm:*

I) Receive as input a triangulated polygon $P$ together with a point $s$.
II) Enqueue the funnel $(P, s, 0)$.
III) WHILE (queue not empty) DO BEGIN
  A) dequeue a funnel $F = (P, B, \alpha)$;
  B) label with $\alpha$ all triangles of $P$ that intersect $B$;
  C) compute the funnel top $T$, leading vertex $v$, and sides $X$ and $Y$;
  D) WHILE (not all of $F$ is labeled) DO BEGIN
    1) compute the bypass bases $B_1$ and $B_2$ and (by cutting along these bases) remove from $F$ the subregions $P_1$ and $P_2$; in the process of doing this, add chords to triangulate any four- or five-sided faces that are formed;
    2) enqueue the funnels $(P_1, B_1, \alpha + 1)$ and $(P_2, B_2, \alpha + 1)$;
    3) label with $\alpha$ the leading triangle of $F$;
    4) update $T$ to be the leading edge of $F$, update the appropriate one of $X$ or $Y$ to have endpoint $v$, and update $v$ to be the new leading vertex;
  END.
IV) Augment the triangulation with a point location data structure.

For the purposes of this section, we only wish to assert that if Step I does receive a serpentine triangulation, then the algorithm will work correctly; in the next section, Step I will invoke a standard polygon triangulation algorithm to produce a (not necessarily serpentine) triangulation of the polygon. Step II places the initial funnel in a queue. Step III repeatedly removes a funnel from the queue (Step III.A) and as bypass bases are discovered, Step III will place new funnels in the queue; the algorithm terminates when there are no more funnels to expand and the graph has been completely labeled. Step III.B labels all triangles that are directly visible from the base; for example in Fig. 9(a), triangles $(a, b, c)$ and $(b, c, d)$ would be labeled, in Fig. 9(b), triangles $(a, c, d)$ and $(a, b, d)$ would be labeled, in Fig. 10, all 10 of the triangles shown would be labeled. Step III.C computes the initial funnel top, sides, and leading vertex; this is done by a simple linear search which can be charged to the triangles that are labeled in Step III.[8] Step III.D.1 walks in a straight line one triangle at a time to perform a "cut and paste" operation that removes a piece of the funnel so that a new funnel can be formed; in this process, it may be necessary to subdivide some four- or five-sided regions. For example, in Fig. 9(a), we proceed from point $d$ to point $m$ with no additional edges needed, but when proceeding from point $m$ to point $n$, we must either add the edge $(n, e)$ or $(m, f)$ (it does not matter which). By Lemma 3 we can always add such edges in a way so as to maintain the serpentine triangulation.[9] Step III.D.2 enqueues the two new funnels that have been removed from $F$; note that one or both of the bypass funnels may not exist (e.g., Fig. 9(a) has only one bypass base). Steps III.D.3 and III.D.4 are what actually cause progress to be made in the portion of the funnel that is visible from the base. Since all points of the leading triangle are clearly visible from the funnel base, Step III.D.3 labels the

[8] In fact, in practice, these two steps would be combined.

[9] Note that in practice, fewer additional edges may have to be added (by a constant factor) by first constructing both bypass bases and then fixing five-sided regions with two edges (rather than adding single edges to four-sided regions after placing the first bypass base and then do this again after the second bypass is added).

leading triangle with cost $\alpha$. Step III.D.4 then makes the new funnel top of $F$ be the leading edge of $F$, promotes the appropriate funnel side past the leading triangle (the old leading vertex becomes the new endpoint of the appropriate funnel side), and updates the leading vertex. In the process of promoting the funnel side, it may be that the associated critical point will change. The new critical point is computed by moving it towards the funnel top along the convex hull of the funnel boundary until the appropriate relationship exists between the side and the edge of the convex hull that is incident to the critical point. Since this process of walking up the convex hull always goes in the same direction (towards the top), the cost of traversing these edges can be charged to the edges themselves.

A detail left out of the foregoing discussion is how the convex hulls of funnel boundaries are maintained. This is done as follows. Each time a new edge is added to one of the funnel boundaries, it can be spliced into the convex hull by repeatedly "short-cutting"; that is, if $(a, b)$ is the new edge, and the existing convex hull (when listed starting at the funnel top) is $(b, c, d, \cdots)$, then if the angle $(a, b, c)$ is not convex, delete edge $(b, c)$ and replace edge $(a, b)$ with edge $(a, c)$. Although a number of short-cutting operations may be necessary to splice in a new edge, the cost of each operation can be charged to the edge that is deleted.

Step IV can employ a standard algorithm to augment a triangulation for point location. Such an algorithm for point location in a triangulated plane is given by Sarnak and Tarjan [17].[10]

Fig. 11 shows a sample output of Algorithm 1; solid lines are polygon edges, dashed lines are bypass bases, thin lines are original chords, and dotted lines are new triangle edges that were added to subdivide four- or five-sided regions that induced in the splitting process.

Correctness of Algorithm 1 follows from the discussion of the preceding section. The following theorem addresses the running time of the algorithm.

*Theorem 1:* Algorithm 1 can be implemented to run in $O(n)$ time, where $n$ is the number of edges in the input polygon.[11]

*Proof:* We assume that Step I can read in the triangulated polygon in linear time. Step II is clearly $O(1)$. Step IV has already been discussed. We now focus attention on Step III.

A key fact is that each of the triangles can be subdivided by at most two funnel bases. To see this, first observe that the minimal-turn distance from any two points in the interior of a given triangle back to the source can differ by at most 1; this is because if we ever found a difference of two, we could improve the cost of one of the points with a path that first visited the other. Given this, it follows that the triangle has one

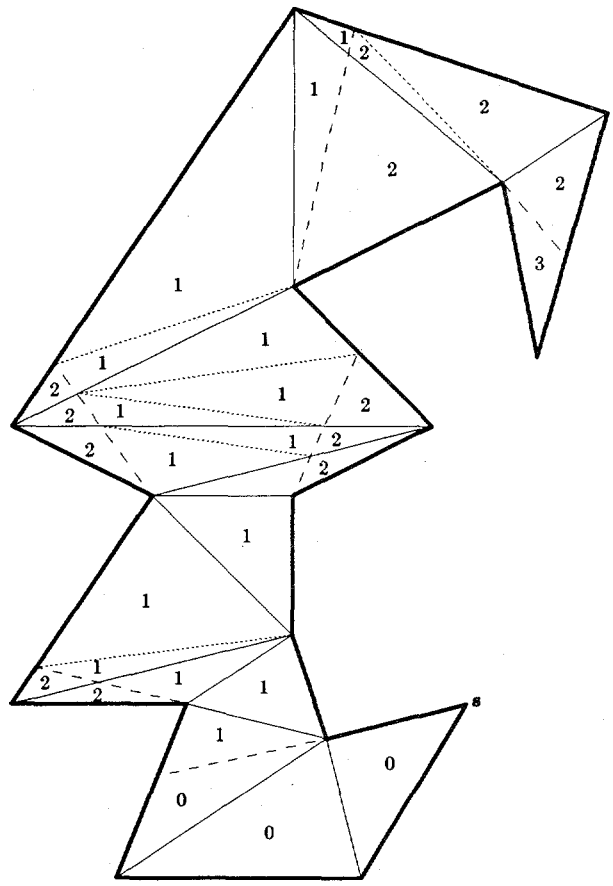



Fig. 11. Sample output.

region of a given cost and at most two other regions of one more than the cost, from which it follows that the triangle can be subdivided by at most two funnel bases (to produce at most three subregions which are either three-, four-, or five-sided). Furthermore, since the funnel bases cannot cross in a yet-to-be-visited triangle, it is not hard to check all possibilities exhaustively to see that the only possible combinations for number of sides in the subdivision of a triangle are (3, 3), (3, 4), (3, 3, 4), (3, 3, 5), and (3, 4, 4); hence each of the original triangles given as input in Step I can be divided into at most five subtriangles. Thus the size of the output of Algorithm 1 is linear in the size of its input.

The analysis is now relatively straightforward; we charge time to the triangles and verify that no triangle is charged more than $O(1)$ time.[12] This is clearly true for Steps III.A, III.B, and III.C; we are left with verifying that the WHILE loop of Step III.D can be so charged. All operations are trivial except for maintaining and traversing the convex hulls of funnel boundaries (which has already been discussed) and traversing the bypass bases; each bypass base is traversed only once to form a new funnel and this cost can be charged to the triangles bordering the new funnel base.

[10] For other work on the subject see Kirkpatrick [10], Preparata [15], Garey *et al.* [8], Lipton and Tarjan [12], Lee and Preparata [11], Dobkin and Lipton [6], and Shamos [18], [19]. Chazelle [4] considers systolic architectures for doing this.

[11] The total number of edges in a chordal triangulation is linear in the number of edges in the polygon, so it does not matter which quantity $n$ denotes.

[12] Whether it is the original triangles or the triangles of the output that are charged is not important since the size of the input and output to Algorithm 1 are linearly related. In addition, for ease of discussion, some of the analysis preceding this discussion has charged cost to edges rather than triangles (since each triangle has only three edges).

## V. Arbitrary Polygons

We now show how Algorithm 1 can be generalized to handle nonserpentine triangulations, minimal path back-pointers, and arbitrary locations for the source point.

*Lemma 4:* Algorithm 1, can be modified to accommodate nonserpentine polygons in $O(n * \log(n))$ time.

*Proof:* We can replace Step I by a call to a standard $O(n * \log(n))$ algorithm to compute a (not necessarily serpentine) triangulation. We must now address two issues: multiple funnel tops and multiple leading edges. Let us first consider multiple funnel tops.

Once we have a funnel with a unique top, Algorithm 1 will maintain a unique top by splitting the funnel into more than one funnel when multiple leading edges are encountered; this will be discussed shortly. However, when a new funnel base is formed and it is time to initialize the funnel top, there may be more than one. In this case, we can simply treat each such top as a separate problem; that is, consider the other funnel tops as border edges for the purposes of defining the sides associated with the top in question. The time to initialize the set of funnels corresponding to a given base is still linear in the number of triangles that intersect the base.

Let us now consider how to handle multiple leading edges. The algorithm encounters no problems until we arrive at the situation in Step III.D where the leading triangle, call it $LT$, is nonserpentine (composed of three chords). One of these chords, call it $T$, is the top of the funnel; call the other two chords $U$ and $V$. We are now faced with the new situation that there are two ways "out of" the leading triangle. If the leading triangle is cut by one of the funnel bases, then the funnels that are cut off by the bypass bases as well as the remaining funnel can be processed separately. Thus we can limit our attention to the case where the bypass bases do not intersect the leading triangle (except at the endpoints of the funnel top). However, now we can make the following key observation.

> All points of the leading triangle are visible from the funnel base. Hence for any path $X$ from a point $p$ to the source $s$ that passes through $LT$, there must be another path from $p$ to $s$ that passes through $LT$ at most once[13] and has no more turns than $X$.

The above observation implies that we do not have to worry about paths that start from $p$, enter $LT$ through $U$, leave through $V$, enter again through $V$, leave through $T$, and go on to the source. Similarly, we do not have to worry about paths that start from $p$, enter $LT$ through $V$, leave through $U$, enter again through $U$, leave through $T$, and go on to the source.

Thus we can split into two problems by first proceeding as though $U$ is a border edge and then proceeding as though $V$ is a border edge. Unfortunately, the straightforward implementation of this approach can cause edges of the polygon to be repeatedly searched (to find the critical point in question) and cause the algorithm to be quadratic in the worst case.[14] To remedy this problem, a balanced tree data structure (e.g., 2-3 tree or AVL tree) can be used to store the convex hulls of the funnel boundaries; this replaces the linear search for the critical point by a logarithmic one.[15]

*Lemma 5:* Algorithm 1, as modified by Lemma 4, can be modified with only a constant increase in running time to accommodate an arbitrary location for the source point.

*Proof:* Suppose that the source point $s$ in a triangulated polygon $P$ is located in a triangle defined by the points $a$, $b$, $c$; it may be that $s$ is a point of one of the edges (or two edges in the case that $s$ is $a$, $b$, or $c$) of the triangle. Then we can proceed as follows.

1) Label the triangle $(a, b, c)$ with cost 0.
2) Cut along the edges $(a, b)$, $(b, c)$, and $(a, c)$ from $P$ to split $P$ into three pieces $P_{ab}$, $P_{bc}$, and $P_{ac}$.
3) Let $\hat{P}_{ab}$ be $P_{ab}$ together with the edges $(s, a)$ and $(s, b)$. Note that if $s$ is a point of $(a, b)$, then the triangle $(s, a, b)$ will be degenerate, but this will not cause a problem. We can similarly define $\hat{P}_{bc}$ and $\hat{P}_{ac}$.
4) Now run Algorithm 1 (as modified for nonserpentine polygons) on each of $\hat{P}_{ab}$, $\hat{P}_{bc}$, and $\hat{P}_{ac}$.

*Lemma 6:* Algorithm 1, as modified by Lemmas 4 and 5, can be modified with only a constant increase in running time to include back pointers in the data structure that allow one to trace out a minimal-turn path from a given point to the source in time proportional to the number of turns.

*Proof:* We can simply place in each triangle a back pointer to the current funnel base.

*Theorem 2:* Algorithm 1 can be implemented to run in $O(n * \log(n))$ time (where $n$ is the number of edges in the input polygon) for arbitrary polygons, with an arbitrary location of the source vertex. In addition, the algorithm can be easily modified to include back pointers that can be used to output minimal-turn paths in time proportional to the number of turns.

*Proof:* This follows from the above lemmas.

Before leaving this section, it is worth noting that the output of Algorithm 1 can be viewed as a partitioning of the polygon into regions consisting of points that have equal minimal-turn distance back to the source. For example, Fig. 11 is partitioned into one region of cost 0, one region of cost 1, four regions of cost 2, and one region of cost 3. From this partitioning, useful visibility information directly follows. For example, the region of cost 0 is all of the points that are visible from the source. Additional visibility information may be obtained by starting Algorithm 1 with different initial conditions. For example, to obtain visibility information for a given line segment, extended it to form a chord that cuts the original polygon into two polygons, treat this chord as a funnel base with cost 0, and run Algorithm 1 on both polygons. The two regions of cost 0 that are computed are the set of all points that

[13] That is, the intersection of $X$ and the border of $LT$ contains at most two points.

[14] In fact, this problem existed with a preliminary version of this paper and we would like to thank one of the referees of this paper for pointing this out.

[15] That is, since angles of the edges in the convex hull are monotonically increasing, this ordering can be used as the basis for the search for the critical point.

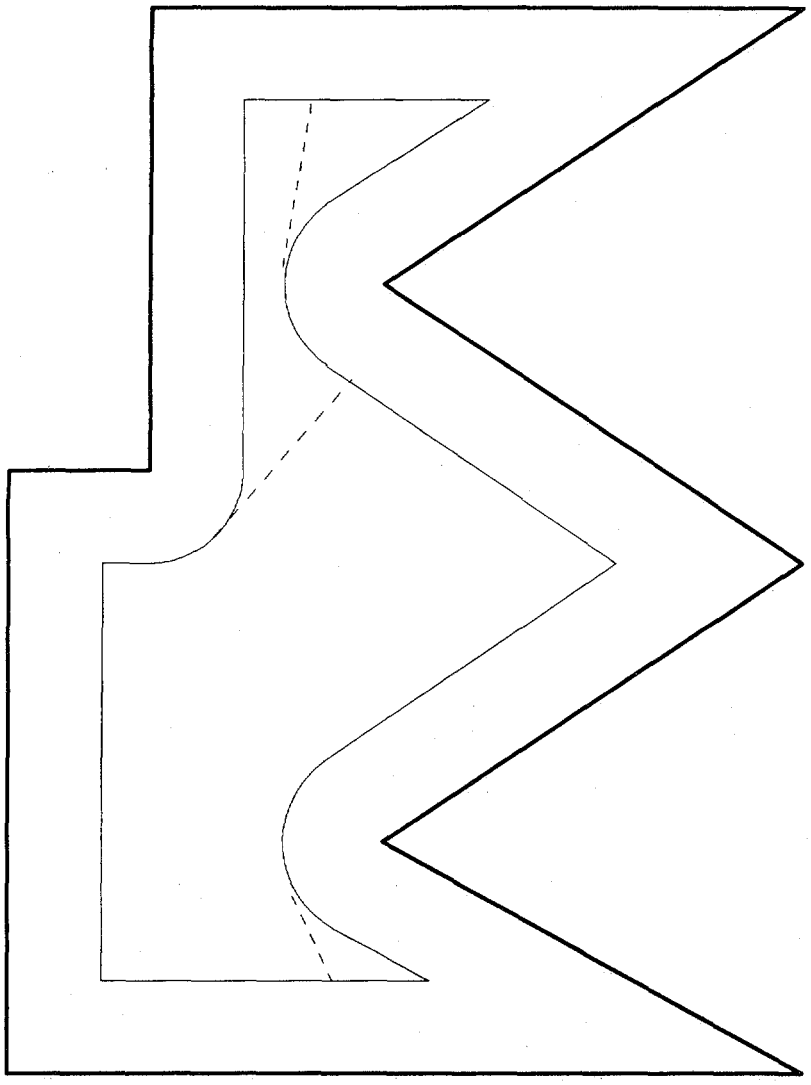


Fig. 12. Movement of disk.

are visible from at least one of the points on the line segment. Regions of cost greater than 0 represent "higher degrees" of visibility from the line segment.

## VI. Movement of a Disk

In this section we note that Algorithm 1 (with the modifications of the last section) can be easily generalized to the movement of a disk of radius $r \geq 0$. The key idea is to add an extra step between Steps I and II of Algorithm 1 to "pad" the polygon by convolving it with the disk. In Reif and Storer [16] it is shown how to do this in linear time (even if the polygon has "holes") if the Voronoi diagram is provided; however, for simple polygons, this construction can easily be modified to use a chordal triangulation instead. Fig. 12 shows a padded polygon. Given the padded polygon, Algorithm 1 can now proceed with straightforward modifications to allow funnel edges to be formed from tangents; for example, the dashed lines in Fig. 12 are the bypass funnel bases. One technical point does remain: The "triangulation" produced by the algorithm does not consist entirely of straight-line segments, and thus cannot be fed directly into a standard point location construction employed in Step IV. However, prior to Step IV, we can enclose triangles with curved edges by larger straight-edged ones. Then, after the algorithm of Step IV has been run, point location can be done in two steps. First use the data structure produced by Step IV to locate the enclosing triangle and then test a quadratic inequality to determine in which subregion of the triangle the point lies.

## References

[1] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai, "Visibility-polygon search and Euclidean shortest paths," in *Proc. 26th Annu. IEEE Symp. Foundations of Computer Science*, Portland, OR, 1985, pp. 155–164.

[2] D. Avis and G. T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," School of Computer Science, McGill Univ., Tech. Rep. 80.2, 1980.

[3] C. Berge, *Graphs and Hypergraphs*. New York: North-Holland, 1976.

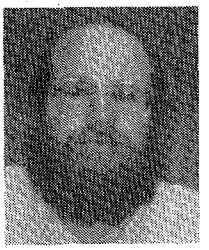
[4] B. Chazelle, "Computational geometry on a systolic chip," *IEEE Trans. Computers*, vol. C-33, pp. 774–785, 1984.

[5] B. Chazelle and L. J. Guibas, "Visibility and intersection problems in plane geometry," in *Proc. First Annu. ACM Symp. Computational Geometry*, Baltimore, MD, 1985, pp. 135–145.

[6] D. P. Dobkin and R. J. Lipton, "Multidimensional searching problems," *SIAM J. Computing*, vol. 5, pp. 181–186, 1976.

[7] H. ElGindy, "Hierarchical decomposition of polygons with applications," Ph.D. dissertation, School of Computer Science, McGill Univ., 1985.

[8] M. R. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, "Triangulating a simple polygon," *IPL*, vol. 7, pp. 175–179, 1978.
[9] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan, "Linear time algorithms for visibility and shortest path problems inside a simple polygon," presented at the 2nd Annu. ACM Symp. *Computational Geometry*, Yorktown Heights, NY, 1985.
[10] D. G. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM J. Computing*, vol. 12, pp. 28–35, 1983.
[11] D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," *SIAM J. Computing*, vol. 6, pp. 594–606, 1977.
[12] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem," in *Proc. 18th IEEE Symp. Foundations of Computer Science*, 1977, pp. 162–170.
[13] W. P. Niedringhaus, "Scheduling without queueing: The space factory problem," Dep. of Elec. Eng. and Comput. Sci., Princeton Univ., Princeton, NJ, Tech. Rep. 253, 1979.
[14] R. Y. Pinter, "On routing two-point nets across a channel," Mass. Inst. Technol., Cambridge, VLSI Memo 82-99, 1982.
[15] F. P. Preparata, "A new approach to planar point location," *SIAM J. Computing*, vol. 10, 1981.
[16] J. Reif and J. A. Storer, "Shortest paths in Euclidean space with polyhedral obstacles," Comput. Sci. Dep., Brandeis Univ., Waltham, MA, Tech. Rep. CS-85-121, 1985, to appear in *J. Ass. Comput. Mach.*
[17] Sarnak and Tarjan, "Planar point location using persistent search trees," *CACM*, vol. 29, pp. 669–679, 1986.
[18] M. I. Shamos, "Geometric complexity," in *Proc. 7th ACM Symp. Theory of Computing*, Albuquerque, NM, pp. 224–233, 1975.
[19] M. I. Shamos, "Computational geometry," Ph.D. dissertation, Comput. Sci. Dep., Yale Univ., New Haven, CT, 1978.
[20] S. Suri, "Finding minimum link paths inside a simple polygon," Dep. Elec. Eng. and Comput. Sci., Johns Hopkins Univ., Baltimore, MD, Tech. Rep. JHU/EECS-85/11, 1985.
[21] ——, "Finding minimum link paths inside a simple polygon," Report JHU/EECS-85/11, Dep. Elec. Eng. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, Tech. Rep. JHU/EECS-85/11, (rev.), 1985.
[22] ——, "A linear time algorithm for minimum link path in a simple polygon," *Comput. Vision, Graphics, Image Processing*, pp. 99–110, July 1986.
[23] S. Suri and J. O'Rourke, "Worst-case optimal algorithms for constructing visibility polygons with holes," JHU/EECS-85/12, Dep. of Elec. Eng. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, Tech. Rep. JHU/EECS-85/12, 1985.
[24] J. A. Storer, "On minimal node-cost planar embeddings," *Networks*, vol. 14, pp. 181–212, 1984.
[25] R. Tamassia, "On embedding a graph in the grid with the minimum number of bends," to appear in *SIAM J. Comput.*, 1986.
[26] R. E. Targan and C. J. Van Wyk, "Linear-time algorithm for triangulating simple polygons," presented at the 18th Annu. ACM Symp. on the Theory of Computing, Berkeley, CA, 1985.
[27] F. A. Valentine, "Minimal sets of visibility," in *Proc. Amer. Math. Soc.*, vol. 4, pp. 917–921, 1953.
[28] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou, "The discrete geodesic problem," (draft) to appear in *SIAM J. Computing*, 1986.



**John H. Reif** received the B.S. degree (magna cum laude) in applied mathematics and computer science from Tufts University, Medford, MA, and the M.S. and Ph.D. degrees from Harvard University, Cambridge, MA, in 1973, 1974, and 1977, respectively.

He taught for two years at Rochester University and then moved back to Harvard University in 1979 as Assistant Professor of Computer Science. He became Associate Professor at Harvard University in 1983. He took a sabbatical at MIT in fall 1985 and visited the Mathematical Sciences Institute at Berkeley, CA in Spring 1986. In summer, 1986 he took his current position as Professor of Computer Science at Duke University, Durham, NC. His current research contributions, in addition to robotic movement planning, include parallel and randomized algorithms.

**James A. Storer** received the B.A. degree from Cornell University, Ithaca, NY, in 1975, and the M.A. and Ph.D. degrees from Princeton University, Princeton, NJ, in 1977 and 1979, respectively.

From 1979 to 1981 he worked as a Researcher for the Bell Laboratories facility located in Murray Hill, NJ. He is currently a member of the faculty of the Computer Science Department at Brandeis University, Waltham, MA. He has recently completed a book on data compression (Computer Science Press). He is also a data compression consultant and a cofounder of Data Compression Corporation, a company that produces hardware for on-line data compression (patent pending). His current research interests include design and analysis of algorithms, parallel computation, data compression, computational geometry, robotics, and VLSI design and layout.

Dr. Storer is a member of the IEEE Computer Society and the Association for Computing Machinery.