A Linear Time Algorithm for Minimum Link Paths inside a Simple Polygon

SUBHASH SURI

Department of Electrical Engineering and Computer Science, The Johns Hopkins University, Baltimore, Maryland 21218

Received August 12, 1985; revised February 25, 1986

1. INTRODUCTION

This paper addresses the following problem. Suppose a communication channel is to be set up between two points inside a simple polygon whose sides are opaque to the transmission. The transmission originates at a point \( s \), called source, and is to be received at a point \( d \), called destination. Clearly, a direct communication is impossible unless the points \( s \) and \( d \) are in sight of each other inside the polygon. Therefore, some repeaters (mirrors, for instance, if the transmission is optical in nature) are to be installed inside the polygon to make the communication feasible. What is the minimum number of repeaters required? With this motivation, a formal statement of the problem can be given as follows. Let \( P \) be a simple polygon. Let \( s \) and \( d \) be two designated points in \( P \). Find a polygonal path between \( s \) and \( d \) that is internal to \( P \) and has minimum number of vertices possible. If the path is given by the set of vertices \( \{ s = x_0, x_1, \ldots, x_k = d \} \), then each segment \( x_i x_{i+1}, 0 \leq i \leq k - 1 \), is called a link, and the entire path is said to consist of \( k \) links. The related problem of finding the Euclidean minimum length path between two points internal to a polygon was studied by Lee and Preparata [7]. The problem of minimum link path was originally posed by Toussaint. In this paper, we present an \( O(n^2) \) time algorithm for computing a path with minimum number of links possible, where \( n \) is the number of vertices of the polygon \( P \). Our algorithm makes use of the recently discovered linear time triangulation algorithm of Tarjan and Van Wyk [12], and linear time edge-visibility algorithm due to Guibas et al. [5].

In Section 2, we introduce our key lemmas that lead to an efficient algorithm for a minimum link path. An algorithm for finding a minimum link path is presented in Section 3. This algorithm has worst-case time complexity of \( O(n^2) \), which is improved to \( O(n) \) in Section 4. Finally, Section 5 concludes with few remarks and directions for further work on this problem.

2. PRELIMINARIES

We assume that the polygon \( P \) is given as a counterclockwise sequence of its vertices. The symbol \( P \) is used to denote the boundary of the polygon as well as the region of the plane enclosed by it. We begin by describing the notion of visibility polygons. A visibility polygon can be viewed as the region in \( P \) that is lit from a light source placed somewhere in \( P \). The light source can be either a point or a line segment (usually referred to as an edge). We use the symbol \( V \) to denote the boundary of the visibility polygon, from either a point source or an edge source. For

---

1El Gindy [4] has independently obtained an algorithm for this problem that runs in \( O(n \log n) \) time.
a point $x \in P$, the visibility polygon from $x$, denoted by $V(x)$, is the set of points in $P$ visible from $x$, i.e.,

$$V(x) = \{ z \in P | xz \cap P = xz \},$$

where $xz$ denotes the line segment connecting $x$ and $z$. For a line segment $e$ inside $P$, the visibility polygon from $e$, denoted by $V(e)$, is the set of points in $P$ visible from $e$, i.e.,

$$V(e) = \{ z \in P | \exists y \in e \text{ s.t. } zy \cap P = zy \}.$$

A chord is a line segment that lies entirely interior to $P$ and has its endpoints on the boundary. A chord $ab$ divides $P$ into two subpolygons, $P_a$ and $P_b$, such that $P_a$ and $P_b$ have their common intersection along $ab$ only. More precisely, let a polygon be represented as a counterclockwise sequence of its vertices. Let $P = (v_1, v_2, \ldots, v_n)$. Let $a$ and $b$ lie on the edges $v_i v_{i+1}$ and $v_j v_{j+1}$, $i < j$, respectively.

The two subpolygons resulting from cutting $P$ along $ab$ are given as $P_a = (b, v_i, v_{i+1}, \ldots, v_j, a)$ and $P_b = (a, v_{j+1}, v_{j+2}, \ldots, v_n, b)$. Two line segments $t_i$ and $t_j$ are said to overlap if and only if their common intersection is a non-degenerate line segment, i.e., is not a point. The following lemma characterizes the edges that make up $V(e)$. The proof is rather elementary and is omitted here.

**Lemma 1.** Let $e$ be a line segment in $P$ such that vertices of $P$ together with the endpoints of $e$ are in general position. Each edge of $V(e)$ either

(a) overlaps with an edge of $P$, or

(b) is a chord with a reflex vertex of $P$ at one endpoint.

Let $T$ be a triangulation of $P$, where a triangulation of a simple polygon is defined as follows, in the manner of Melhorn [8]:

A triangulation of a vertex set $\{v_1, v_2, \ldots, v_n\}$ is a maximal set of nonintersecting straight line segments between points in this set. A triangulation of polygon $P$ is a triangulation of its vertex set such that all the triangulating edges lie in $P$.

Let $G$ be the dual graph of this triangulation. $G$ is obtained by assigning a vertex to each face of $T$, and joining two vertices $t_i$ and $t_j$ if and only if the triangles $t_i$ and $t_j$ in the triangulated polygon share a common side. Thus, a triangle $t_i$ of $T$ has its corresponding node in the dual graph $G$ labeled $t_i$. The graph $G$ obtained in this way is a tree, whose nodes have a maximum degree of three. Let $t_i$ and $t_j$ be the triangles in $T$ containing $s$ and $d$, respectively. Let $X_{t_i}$ be the path in $G$ from node $t_i$ to node $t_d$. Since $G$ is a tree, this path is unique. Let the nodes in the path $X_{t_i}$ be indexed in increasing order according to their appearance along the path. Then the path $X_{t_i}$ can be given as an ordered list of nodes $\{t_i, t_{i+1}, \ldots, t_{n-i}, t_d\} = t_i$. The triangles $t_i$ through $t_d$ indexed in this manner define a "forward" direction of visibility in $P$. Our computation of the visibility polygons will progress in this direction. Let $T_{t_i}$ denote the ordered list of triangles $\{t_i, t_{i+1}, \ldots, t_d\}$. We assume throughout that $G$ is rooted at $t_i = t_d$.

Our interest in $T_{t_i}$ is motivated due to the following reasons. We show that any minimum link path in $P$ from $s$ to $d$ intersects every triangle of $T_{t_i}$ in the order of increasing indices. For each triangle $t_i \in T_{t_i}$ we compute the minimum number of links in a path from $s$ to a point in $t_i$. Since $d \in t_d$, these computations will ultimately lead to the minimum number of links required between $s$ and $d$. By storing certain boundaries of the visibility polygons computed in the procedure, we will also reconstruct the path from $s$ to $d$ with a minimum number of links.

First we show that certain portions of $P$ that are inessential for a minimum link path between $s$ and $d$ can be removed to simplify our discussion, and possibly gain some efficiency in practice. Let $x_1, y_1$, and $z_1$ (resp. $x_2, y_2$, and $z_2$) be the vertices of $t_i$ (resp. $t_j$) such that $y_1 z_1 (y_2 z_2)$ is shared between $t_i$ and $t_j$ (resp. $t_{j-1}$ and $t_j$). Let $P(x_1, y_1)$ and $P(x_2, z_2)$ be the polygons not containing $t_i$ that result from cutting $P$ along $x_1 y_1$ and $x_2 z_2$, respectively. Similarly for polygons $P(x_1 z_1)$ and $P(x_2 z_1)$. The following lemma is quite straightforward.

**Lemma 2.** The polygon $P$ with $P(x_1, y_1), P(x_1 z_1), P(x_2 y_2)$, and $P(x_2 z_2)$ removed is sufficient for computing a minimum link path between $s$ and $d$.

The proof of Lemma 2 is based on the fact that $P$ is a simple polygon, and none of the removed sections contains either $s$ or $d$. We omit this simple proof.

For the remainder of our discussion, we assume that our polygon $P$ has been trimmed in the manner of Lemma 2. The following corollary follows immediately from Lemma 2.

**Corollary 1.** In the dual graph $G$, $t_i$ is a leaf.

A chord $ab$ is said to intersect a triangle $t_i \in T_{t_i}$ if and only if $(a, b) \cap t_i \neq \emptyset$, where $(a, b)$ is the open line segment from $a$ to $b$ and $t_i$ denotes the boundary and the interior of the triangle $t_i$. For the remaining discussion, we will focus our attention on only those chords that intersect at least one triangle of $T_{t_i}$. Let $e$ be a chord of $P$. Define $t_{i, \max} \in T_{t_i}$ to be the highest indexed triangle intersected by $e$. Let $P_e$ and $P_{t_{i, \max}}$ be two subpolygons that result from cutting $P$ along $e$. Assume without loss of generality that $P_e$ contains all the triangles $t_{i, \max} + 1, t_{i, \max} + 2, \ldots, t_k$. The visibility polygon $V(e)$ is computed inside $P_{t_{i, \max}}$ only.

Let $e$ be a chord of $P$, and $V(e)$ the visibility polygon of $e$. Let $t_i \in T_{t_i}$, $1 \leq i \leq k$, be the highest indexed triangle whose interior is (at least partially) visible from $e$. Let $p, q$, and $r$ be the three vertices of $t_i$. Assume that $gr$ is shared between $t_i$ and $t_{i+1}$. An edge $ab$ of $V(e)$ is defined to be the window of $e$, $w(e)$, if

(a) $ab$ is a chord of $P$,

(b) $ab$ intersects $t_i$ and $t_{i+1}$

(c) if $ab$ intersects the boundary of $t_i$ in $a_i$ and $b_i$, then the interior of the quadrilateral $a_i b_i a b$ is not visible from $e$.

Intuitively the window, $w(e)$, represents the farthest along $T_{t_i}$ that is visible from $e$ in $P$. Starting from the source $s$, our algorithm will compute the visibility polygons only for the successive windows. The following lemma establishes that the window is well defined for any chord of $P$.

**Lemma 3.** Let $e$ be a chord of $P$. Either there exists an unique $w(e)$, or the destination lies in $V(e)$.
Lemma 4. Let $e$ be a chord and $w(e)$ its window. Let $t_{w, \text{max}}$ be the highest indexed triangle of $T_{1,k}$ intersected by $e$. Let $t_{w, \text{min}}$ be the highest indexed triangle of $T_{1,k}$ intersected by $w(e)$. Let $y \in t_{w, \text{max}} \cap t_{w, \text{min}}$. Let $x \in e$ be a point that is visible from $y$:

(a) The line segment $xy$ intersects all triangles $t_i \in T_{1,k}$, $e \text{ max} \leq i \leq w \text{ max}$. 

(b) Any other triangle of $T$ that is intersected by $xy$ is also intersected by $e$.

Proof. (a) Let $p$, $q$, and $r$ be the vertices of $t_{w, \text{max}}$. Assume that $qr$ is shared between $t_{w, \text{max}}$ and $t_{w, \text{max}+1}$. Clearly, $qr \cap e = \emptyset$. Let $P_1$ be the subpolygon that results from cutting $P$ along $qr$ and contains $e$. Let $P_2$ be the other subpolygon. If $y \in t_{w, \text{max}+1}$, the claim follows vacuously. Otherwise, since $y \in P_2$ and $x \in P_1$, the segment $xy$ must intersect $qr$ and $t_{w, \text{max}}$. Let $xy$ intersect $qr$ in $x'$. Since $x' \in t_{w, \text{max}}$ and $y \in t_{w, \text{max}}$, and $e \text{ max} < w \text{ max}$, $xy$ must cross a sequence of triangles, namely, $t_{w, \text{max}} = t_i, t_j, \ldots, t_l = t_{w, \text{max}}$ such that two adjacent triangles share a common side. This sequence, therefore, represents a path from $t_{w, \text{max}}$ to $t_{w, \text{max}}$ in $G$. But, since $G$ is a tree, this path is unique. The sequence $t_i$ through $t_l$ must, therefore, coincide with the sequence of triangles in $T_{1,k}$ indexed between $e \text{ max}$ and $w \text{ max}$. It follows that $xy$ intersects all the triangles of $T_{1,k}$ indexed between $e \text{ max}$ and $w \text{ max}$. The claim is thus proved.

(b) It is sufficient to show that any triangle of $T$ that is intersected by $xx'$ is also intersected by $e$. If $x$ lies in any triangle of $T_{1,k}$, i.e., $x \in t_i, t_i \in T_{1,k}$ and $e \text{ min} \leq i \leq e \text{ max}$, the argument presented in (a) can be repeated to show that $xx'$ intersects only the triangles of $T_{1,k}$ indexed from $i$ through $e \text{ max}$. Otherwise proceed as follows. Let $p$, $q$, and $r$ be the vertices of $t_{w, \text{max}}$ where $qr$ is shared between $t_{w, \text{max}}$ and $t_{w, \text{max}+1}$. Let $e$ intersect $pq$ and $pr$ at $a_i$ and $b_i$, respectively (see Fig. 2). Let $xx'$ intersect $pr$ at $b_i$. Since the triangle $\Delta a_i b_i r_i$ is internal to $V(e)$, it cannot have holes. Hence, there are no vertices of $P$ in the interior of triangle $\Delta a_i b_i r_i$, $xx'$, therefore, intersects exactly those triangles that are intersected by $xx'$, and the claim follows. ∎

Now we want to establish that there are no gaps in the visibility polygon $V(e)$ with respect to the set $T_{1,k}$, i.e., $V(e)$ intersects all the triangles of $T_{1,k}$ that lie between $e$ and $w(e)$.
COROLLARY 2. \( V(e) \) intersects every triangle \( t_i \subseteq T_{i,k} \), for every \( l \leq i \leq w, \max \).

The following lemma establishes a property of \( V(e) \) that is crucial for an efficient computation of a minimum link path.

**Lemma 5.** Let \( e_1, e_2, \ldots, e_k \) be a sequence of chords such that

(a) \( e_1 \) intersects \( t_{i_1} = t_i \),
(b) \( e_j = w(e_{j-1}) \), \( 2 \leq j \leq i \), and
(c) \( d \not\in \bigcup_{j=1}^{i} V(e_j) \).

Let \( x \) be a point in \( \bigcup_{j=1}^{i} V(e_j) \). Then any path in \( P \) from \( x \) to \( d \) intersects \( e_i \).

**Proof.** Consider the two subpolygons \( P_1 \) and \( P_2 \) that result from cutting \( P \) along the chord \( e_i \). Assume without loss of generality that \( x \in P_1 \). We first establish that \( d
\in P_2 \). All triangles \( t_u \subseteq T_{i,k} \), \( 1 \leq u \leq e_{\min} - 1 \), are completely contained in \( P_1 \), where \( e_{\min} \) is the lowest indexed triangle of \( T_{i,k} \) intersected by \( e_i \). In addition, \( P_1 \) also contains all the triangles \( t_u \subseteq T_{i,k} \), \( e_{\min} \leq u \leq e_{\max} \), where \( e_{\max} \) is the highest indexed triangle of \( T_{i,k} \) intersected by \( e_i \). \( P_2 \) intersects no other triangle of \( T_{i,k} \) besides these. Hence, if \( d \in P_1 \), then \( k \leq e_{\max} \).

Since \( e_j = w(e_{j-1}) \) for \( 2 \leq j \leq i \), Lemma 4 implies that \( \bigcup_{j=1}^{i} V(e_j) \) intersects all triangles \( t_u \subseteq T_{i,k} \), \( 1 \leq u \leq e_{\max} \). Let \( e_{\min} \leq u \leq i - 1 \), be the first chord such that \( V(e_j) \) intersects \( t_u \). Since \( d \not\in V(e_j) \), \( d \) must lie in that part of \( t_u \) that is hidden from \( e_j \) by a chord. Since \( k \) is the highest indexed triangle of \( T_{i,k} \), this chord must be \( w(e_j) \). If \( u \leq (i - 1) \), then it is easily seen that \( d \not\in V(e_{j+1}) \), thus contradicting the hypothesis on \( d \). If \( u = (i - 1) \), \( d \) clearly must lie on that side of \( w(e_j) = e_i \) that goes into \( P_2 \) after splitting \( P \) along \( e_i \). Therefore, \( d \not\in P_1 \).

Clearly, the common intersection of \( P_1 \) and \( P_2 \) lies only on \( e_i \). Since \( x \in P_1 \) and \( d \in P_2 \), any path from \( x \) to \( d \) must intersect \( e_i \). This establishes the claim of the lemma.

**Note.** Both \( x \) and \( d \) can be regarded as degenerate cases of a chord.

3. **ALGORITHM FOR MINIMUM LINK PATH**

Our algorithm computes the visibility polygons for successive windows, starting from \( s \). Let \( s = e_1 \) be the first chord. Compute \( V(e_1) \). If \( d \in V(e_1) \), we have a one-link path, i.e., \( x = s \) and \( d \) can be joined by a straight line inside \( P \), and we are done. Otherwise, by Lemma 3, \( w(e_1) \) exists. Also, Lemma 5 says that a path from \( s \) to \( d \) intersects \( w(e_1) \). Let \( e_2 = w(e_1) \), and compute \( V(e_2) \), and so on until we find a window \( e_i = w(e_{i-1}) \) such that \( d \in V(e_i) \). The minimum link path from \( s \) to \( d \) can then be found by constructing a sequence of point \( d_{i+1} = d, d_i, d_{i-1}, \ldots, d_1 \), where \( d_{i+1} = d \) and \( d_j, 1 \leq j \leq i \) is a point on \( e_j \) that can see \( d_{j+1} \). Since, \( d_i = s \), the path has \( i \) links. Thus, the algorithm for finding a minimum link path can be given as follows:

**Algorithm A.** An algorithm for finding a minimum link path.

1. \( w(e_0) = s \); \hspace{1cm} (*Initialization*)
2. \( i \leftarrow 0; \)
3. **repeat**
4. \( i \leftarrow i + 1; \)
5. \( e_i \leftarrow w(e_{i-1}); \)
6. Compute \( V(e_i) \)
7. until \( d \in V(e_i) \)
8. \( d_{i+1} \leftarrow d; \)
9. for \( j = i \) down to 1 do
10. \( d_j \leftarrow a \) point on \( e_j \) that can see \( d_{j+1}; \)
11. **output minimum link path** = \( (s = d_i, d_{i-1}, \ldots, d_1, d_{i+1} = d) \); **end (Algorithm A)**

**Correctness and Time Complexity of Algorithm A.** The correctness of Algorithm A can be easily established as follows. Let \( e_1, e_2, \ldots, e_k \) be the set of chords output by Algorithm A such that

(a) \( e_1 = s \),
(b) \( e_j = w(e_{j-1}) \), \( 1 < j \leq k \),
(c) \( d \not\in V(e_j) \) for \( d \not\in V(e_j) \).

Then we claim that a minimum link path \( (MLP) \) between \( s \) and \( d \) requires \( k \) links. Assume for a contradiction that \( MLP \colon (s = e_1, e_2, \ldots, e_k = d) \) is a minimum link path such that \( i < k \). Clearly, \( e_i \in V(e_j) \), since \( e_j = e_i \). Lemma 5 dictates that MLP intersects \( e_2 \), and it follows that \( e_j \in V(e_i) \). The vertices of MLP and windows of Algorithm A can thus be matched one to one, which contradicts the assumption that \( i < k \). It is therefore established that Algorithm A does indeed output a minimum link path from \( s \) to \( d \).

The time complexity analysis can be done in the following manner. Since a minimum link path from \( s \) to \( d \) can have \( O(n) \) links, the **repeat** loop can be executed \( O(n) \) times. Computation of \( V(e_i) \) is in fact computing visibility polygon from an edge, and can be done in \( O(n) \) time using very recent results due to Guibas et al. [5]. The property \( d \in V(e_i) \) can be checked in time proportional to the size of \( V(e_i) \), which is \( O(n) \). That the actual construction of the path (lines 8-11) takes only \( O(n) \) time can be established as follows. We preprocess each visibility polygon \( V(e_i) \) using the methods of Guibas et al. [5] so that given a point \( x \) on the boundary of \( V(e_i) \) a point \( y \) such that \( xy \) lies in \( V(e_i) \) can be located in \( O(\log n) \), where \( n \) is the size of \( V(e_i) \) (see [5] for details). Hence, if the path has \( k \) links, then the total cost of path reconstruction is \( O(k \Sigma_{i=1}^{k} (\log n_i)) \), where \( \Sigma_{i=1}^{k} n_i = n \). Clearly, \( O(\Sigma_{i=1}^{k} (\log n_i)) = O(n) \). It follows, therefore, that the time complexity of Algorithm A is \( O(n^2) \). It is noteworthy that if we spend \( O(n) \) time for computation of \( V(e_i) \) each time, by considering the entire polygon \( P \), Algorithm A can actually take \( O(n^2) \) time: Fig. 3 shows a polygon that forces the minimum link path to have \( n/2 \) links.

In the following section we show how the time complexity of Algorithm A can be improved to \( O(n) \). This improvement in the running time is achieved by computing \( V(e_i) \)'s more efficiently. Instead of spending \( O(n) \) time for computing each \( V(e_i) \), we spend only \( O(n) \) time, where \( n_i \) is the total number of triangles of \( T \) that are intersected by \( V(e_i) \). Since \( \Sigma_{i=1}^{k} n_i = O(n) \), even though the **repeat** loop of Algorithm A can be executed \( O(n) \) times, the computation of all \( V(e_i)'s \) together will require only \( O(\Sigma_{i=1}^{k} n_i) = O(n) \) total time. The technique for computing \( V(e_i)'s \) in \( O(n) \) time is detailed in the following section.
4. EFFICIENT COMPUTATION OF $V(e)$

In this section, we substantiate our claim that the visibility polygon $V(e)$ can be computed in $O(n)$, where $n$ is the number of triangles of $T$ that are intersected by $V(e)$. The basic idea is as follows. Let $e$ be a chord of $P$ for which we want to compute $V(e)$. Let $W(e)$ be the window of $e$. Let $t_{min}$ be the lowest indexed triangle intersected by $e$. Let $t_{max}$ be the highest indexed triangle intersected by $w(e)$. Lemma 4 implies that $V(e)$ intersects all triangles $t_i \in T_{i,k}$, $e_{min} \leq w_{max}$. To be more precise, let $y \in w(e)$ and $t_{max}$ be a point. Let $x \in e$, be a point that is visible from $y$. Lemma 4 proves that the segment $xy$ intersects only those triangles of $T$ that are either intersected by $e$, or are triangles of $T_{i,k}$ indexed between $t_{min}$ and $w_{max}$. Therefore, we can perform a binary search exclusively on the triangles of $T_{i,k}$ in the “forward” direction to locate $t_{max}$. The binary search starts by computing the visibility region from $e$ inside a small set of triangles and doubles the number of triangles under consideration at each step. These notions are made more precise in the following.

Consider the dual graph of the triangulation, $G$. Let $x$ and $y$ be two nodes in $G$. There is a unique path in $G$ from $x$ to $y$. The union of triangles $x$ and $y$ along with all the triangles that lie along the path from $x$ to $y$ describe a polygonal region in $P$. We use this duality between length of a path in $G$ and size of the corresponding polygonal region in $P$ to successively double the size of the polygonal region considered for computing the visibility polygon from $e$. By size of a polygonal region $P$, we mean the number of vertices of $P$.

Let $e$ be a chord of $P$ for which we need to compute the visibility polygon $V(e)$. Let $e_{max} - e_{min} + 1 = m$. Let $m = m_1 + m_2$ be the total number of triangles of $T$ that are intersected by $e$. First $m_1$ triangles are in $T_{i,k}$ indexed between $e_{min}$ and $e_{max}$, and remaining $m_2$, namely, $(t_1, t_2, \ldots, t_{m_2})$ are in $T - T_{i,k}$. Let $P_1$ be the subpolygon that results from cutting $P$ along $e$ and contains all the triangles $(t_1, t_2, \ldots, t_{m_2})$.

We start by computing the visibility region from $e$ inside $P_1 \cap (t_{min} + 1 \leq \ldots \leq t_{max} + 1 \ldots t_{max})$. This region is trivially entirely visible from $e$. At next step we double the size of the polygonal region under consideration by including $m_2$ more triangles of $T_{i,k}$ starting from $t_{min}$. The visibility computation is done over again at each new step. In general, let $P^j$ be the polygonal region under consideration at the $j$th step. Let $|P^j|$ denote the number of triangles in $P^j$. Let $t_{max}$ be the highest indexed triangle of $T_{i,k}$ that is included in $P^j$. The polygonal region considered for the $(j + 1)$th step is $P^j \cup \{t_{min}, \ldots, t_{max}, p^j\}$. We call this procedure of successively increasing the size of a polygonal region by a factor of 2, “doubling.” The doubling is discontinued if no point of the common diagonal of the $i_{max}$-th and $(i_{max} + 1)$-th triangles is visible from $e$, in which case $w_{max}$ must lie between $e_{min}$ and $t_{max}$. In addition, $w_{max} \geq i_{max}$ for any $1 < i_j$; otherwise the doubling must have stopped at the $j$th step. Hence we conclude that if doubling stops at $j$th step, then $e_{min} \leq w_{max} \leq j_{max}$, and $V(e)$ intersects at least half the triangles of $P$.

Now we are ready to compute the actual visibility polygon $V(e)$. The reason for this final computation is the following. The visibility region computed at the $j$th step of doubling may be a proper subset of $V(e)$. This is due to the fact that all the triangles of the set $T - T_{i,k}$ have been completely ignored. Therefore, the part of $V(e)$ that lies in the triangles of $T$, has not yet been computed. However, $t_{max}$ is correctly determined. In order to compute $V(e)$, therefore, the final computation considers the following triangles

(a) all triangles of $P$,
(b) all triangles $t_i$ such that in $G$ the least common ancestor of $t_i$ with $t_{max}$ is indexed between $e_{min}$ and $j_{max}$.

Since $G$ is rooted at $t_i$, the least common ancestor is well defined. The refined algorithm for computing $V(e)$ can now be given as

**Algorithm B**

An algorithm for computing $V(e)$ efficiently.

1. $P' \leftarrow$ union of all the triangles that are intersected by $e$;
2. $V' \leftarrow$ region of $P'$ visible from $e$;
3. $j \leftarrow 1$;
4. repeat
5. $j \leftarrow j + 1$;
6. $P' \leftarrow P' - t_j$ U $\{[P' - t_j]$ new triangles of $T_{i,k}\}$
7. $V' \leftarrow$ region of $P'$ visible from $e$;
8. $t_{max} \leftarrow$ highest indexed triangle in $P'$;
9. until $P' \cap \{\text{common diagonal of} \ t_{max} \text{and} \ t_{max} + 1\} = \emptyset$;
10. $P_{final} \leftarrow P' \cup \{\text{all triangles} \ t_i \in T - T_{i,k} \text{whose least common ancestor with} \ t_{max} \text{in} G \text{is indexed between} \ e_{min} \text{and} \ j_{max}\}$
11. $V(e) \leftarrow$ region of $P_{final}$ visible from $e$;

end (Algorithm B)

**Proof of Correctness of Algorithm B**

We show that Algorithm B correctly computes $V(e)$. First we show that $t_{max}$ is correctly determined. Let $t_{max}$ be the highest indexed triangle intersected by $w(e)$. Lemma 4 proves that for any point $y \in w(e)$ and a point $x \in e$, that is visible from $y$, the line segment $xy$ intersects only those triangles of $T$ that are either intersected by $e$, or are triangles of $T_{i,k}$ indexed between $e_{min}$ and $w_{max}$. Since $P'$ considers all triangles of $T_{i,k}$ indexed between $e_{min}$ and $j_{max}$, where $j_{max} > w_{max}$ along with $t_j$ through $t_{max}$, which are the triangles intersected by $e$, of the set of $T_{i,k}$, $w_{max}$ is correctly determined. Next, if $V(e)$ intersects any triangle of $t_i \in T - T_{i,k}$ then the least common ancestor of $t_i$ and $t_{max}$ in $G$ must be indexed between $e_{min}$ and $j_{max}$ (see Fig. 4). This can be established in the following manner. We can assume without loss of generality that $j_{max} \leq k$. Let $t_u \in T - T_{i,k}$ be a triangle that is intersected by $V(e)$. Let the lowest common ancestor of $t_u$ and $t_i$ in $G$ be $t_v$, $1 \leq v \leq k$. We
We have considered the problem of finding a minimum link path between two designated points inside a simple polygon. We first describe an algorithm that runs in $O(n^3)$ time. Later, we improve the running time of this algorithm to $O(n)$. Using a similar approach we can also preprocess the polygon $P$ in $O(n)$ time and space to obtain a data structure for minimum link path queries. This data structure can

(1) output the minimum number of links needed between a fixed source $s$ and an arbitrary destination $d$ in $O(\log n)$ time,

(2) construct the actual path from $s$ to $d$ in time $O((\log n + k)$, where $k$ is the number of links in the path.

The details of this preprocessing will be presented in [10].

The general problem of finding minimum link paths between two points in the presence of polygonal obstacles seems more difficult. Recently Suri and O’Rourke [9] showed that the boundary of an edge-visibility polygon in the presence of polygonal holes may have $O(n^2)$ vertices in the worst case. Similar worst-case lower bounds may apply to the minimum link path problem when the holes are present.

Recently Toussaint [7] suggested a new measure of the shape complexity of a simple polygon. Let $d(x, y)$, the distance between $x$ and $y$, be defined as the minimum number of links in a polygonal path between $x$ and $y$. This metric, called the minimum link metric, suggests a new measure for the internal diameter of a polygon. Define the link-diameter of a polygon $P$ to be the largest distance between two points of $P$ in the minimum link metric. Our algorithm for minimum link path between two points can be used to give an $O(n^2)$ algorithm for this problem.

However, Suri [10] has recently discovered a more efficient algorithm that has running time of $O(n \log n)$. He also presents a simpler approximation algorithm with running time $O(n \log n)$, whose output is guaranteed to be within two of the link-diameter.

ACKNOWLEDGMENTS

I wish to thank Professor Joseph O’Rourke with whom I have had fruitful discussions on this problem and whose guidance provided constant encouragement. This work was supported by NSF Grant DCR83-51468.

REFERENCES

8. K. Mehlhorn, "Multi-dimensional Searching and Computational Geometry," EATCS Monographs on
A Vectorizer and Feature Extractor for Document Recognition

THEO PAVLIDIS

AT & T Bell Labs, Murray Hill, New Jersey 07974

Received September 25, 1985; accepted March 11, 1986

This paper has two parts. The first part (Sects. 1–5) describes a thinning algorithm that operates directly on the run length encoding of a bilevel image. Besides finding strokes that approximate the dark regions of the image, the algorithm determines other features that are useful for character recognition: arcs, holes, endpoints, etc. The second part (Sect. 6) describes the use of the results of the thinning algorithm in a simple demonstration system for document recognition. © 1986 Academic Press, Inc.

1. INTRODUCTION

The conversion of raster images into vector form is of interest in many applications ranging from cartography to character recognition. A significant part of the problem is the identification of linear or curved strokes from the binary data. Some authors start with a pixel-based thinning algorithm [18, 7, 14, 10] and then identify line or curve segments. Others extract the information by a decomposition technique [8, 3, 2]. Combinations of the two approaches as well as alternative techniques have also been investigated [1, 5, 12, 15, 11].

This paper extends the results of [15] to achieve a vectorization algorithm that operates directly on the run length encoding of a bilevel image. When applied to pages of text, the algorithm is faster by a factor of about three than the hybrid algorithm of [15], and faster by at least an order of magnitude than the pixel-based thinning algorithm of [14]. This speed-up is not surprising because the algorithm forms large groups of pixels and then operates on them rather than the individual pixels. If we have an \( N \times N \) picture, then a pixel-based thinning algorithm has complexity of at least \( N^2 \). In contrast, the complexity of the current algorithm is approximately of the order of the number of run length code segments, except for the initial cost of finding the run length encoding. However, computational speed is not the only criterion to judge a vectorization algorithm. The quality of the results is also important. The raster images in question are usually the results of digitization of documents that contain a mixture of text and graphics. The desired end result of the processing is a list of vectors and character codes. Therefore, vectorization for such applications must produce lines and curves that meet two objectives: (a) They are close to the original lines and curves for the graphics part. (Or at least they are similar to the lines that a human observer might draw in order to vectorize a blob.) (b) They are useful as features for the recognition of characters for the text part. Readers of the paper may judge for themselves whether we have achieved the first objective by looking at the various examples. In order to facilitate recognition the

\*Present address: Dept. of Electrical Engineering, SUNY, Stony Brook, NY 11794.

\*This is of order \( N^2 \) but is at a low coefficient. Some digitizers have hardware producing that code so the test is hidden from the vectorization algorithm.