Optimal Linear-Time Algorithm for the Shortest Illuminating Line Segment in a Polygon

(Extended Abstract)

Gautam Das *† Giri Narasimhan†

Abstract

Given a simple polygon, we present an optimal linear-time algorithm that computes the *shortest illuminating line segment*, if one exists; else it reports that none exists. This solves an intriguing open problem by improving the $O(n \log n)$-time algorithm [Ke87] for computing such a segment.

1 Introduction

In this paper we present an optimal linear-time algorithm to do the following: given a simple polygon with n vertices, it computes in $O(n)$ time the shortest illuminating line segment inside the polygon, if one exists; otherwise it reports that no such segment exists. In other words, it computes the shortest line segment within a polygon that needs to be illuminated in order for the interior of the polygon to be lit. The algorithm solves an outstanding open problem by improving the $O(n \log n)$-time algorithm presented by Ke [Ke87] for this problem.

The concept of *weak visibility* in polygons was introduced by Avis and Toussaint [AT]. Since then it has received much attention from researchers [AT, BMT, SS, TL, DHN1, DHN2, Che, DC, IK, Ke87]; also see the survey article by J. O'Rourke [O'R]. Two sets of points are *weakly visible* from each other if *every* point in either set is visible from *some* point in the other set. Thus, our algorithm computes the shortest line segment that is weakly visible from a given simple polygon.

*Supported in part by NSF Grant CCR-930-6822

†Math Sciences Dept., Memphis State University, Memphis, TN 38152. *e-mail*: dasg/giri@next1.msci.memst.edu

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association of Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

10th Computational Geometry 94-6/94 Stony Brook, NY, USA
© 1994 ACM 0-89791-648-4/94/0006..$3.50

An interesting related result due to Bhattacharya and Mukhopadhyay [BM] is a linear-time algorithm to compute a *single* weakly-visible line segment in a polygon. Although a linear-time algorithm was presented for this problem in [DHN2], the strength of their result lies in the fact that their algorithm does not use the linear-time triangulation algorithm [Cha], or the linear-time algorithm to compute shortest paths in a triangulated polygon [LP, GHLST]. The present algorithm, in contrast, uses both the triangulation as well as the shortest path algorithms, but manages to compute more; it computes the *shortest* weakly-visible segment in a polygon. Another related result is an optimal linear-time algorithm due to Bhattacharya et al. [BMT] for computing the shortest line segment from which the exterior of a simple polygon is weakly visible. It may be noted that we are concerned with internal weak visibility.

Besides resolving a long-standing open problem, our paper is also interesting because of the techniques used. The results in this paper build on some of our previous work on optimal linear-time algorithms for weak-visibility problems in polygons. The linear-time algorithms for computing *all LR-visible pairs of points* [DHN1] and for computing *all weakly-visible chords* [DHN2] output a mass of information related to visibility within a polygon. Our present algorithm shows how to exploit this wealth of information to answer more interesting questions related to weak visibility in polygons. We achieve our results by studying the structure of *minimal* weakly-visible segments and identifying the bounding chords for such segments. As described later, one of the by-products of our algorithm in this paper is a linear-time algorithm to generate all minimal weakly-visible segments. These techniques were also used in [DHN2] to obtain a linear-time recognition of L_2*-convexity* of simple polygons.

The shortest illuminating line segment in a poly-

gon can also be thought of as the shortest straight line path that a watchman could patrol along in order to watch over a polygonal art gallery. There have been a number of papers on the *shortest watchman tour* problem [CN, PV]. Our algorithm finds the shortest straight-line watchman tour, if one exists.

2 Notation

We define notation for this paper. A *polygonal chain* is a concatenation of line segments. The endpoints of the segments are called *vertices*, and the segments themselves are *edges*. If the segments intersect only at the endpoints of adjacent segments, then the chain is *simple*, and if a polygonal chain is closed we call it a *polygon*. In this paper, we deal with a simple polygon P of n vertices, and its interior, $int(P)$. The segment between two points x and y is denoted $\overline{xy}$, and $int(\overline{xy}) = \overline{xy} \setminus \{x, y\}$. Two points $x, y \in P$ are *visible* (or *co-visible*) if $\overline{xy} \subset P \cup int(P)$. Two sets of points A and B are said to be *weakly visible* from each other if *every* point in A is visible from *some* point of B, and vice versa. A polygon is said to be L_2–convex if for every pair of points in the polygon, there exists another point from which the first two are visible. The (Euclidean) distance between two points x and y is denoted $dist(x, y)$. We assume that the input is in general position, which means that no three vertices are collinear, and no three lines defined by edges intersect in a common point.

If x and y are points of P, then $P_{CW}(x, y)$ ($P_{CCW}(x, y)$) is the subchain obtained by traversing P clockwise (counterclockwise) from x to y. The subchains $P_{CW}(x, y)$ and $P_{CCW}(x, y)$ includes their endpoints x and y. A polygon P is said to be *LR-visible* with respect to a pair of points x and y, if $P_{CW}(x, y)$ is weakly visible from $P_{CCW}(x, y)$.

We let $\vec{r}(x, \alpha)$ represent the ray rooted at x in direction α. For a vertex x of P, let x^+ be the vertex adjacent to x in the clockwise direction, and x^- the vertex adjacent in the counterclockwise direction. Informally, the *ray shot* from a point $x \in P$ in direction α consists of "shooting" a "bullet" from x in direction α which travels until it hits a point of P. The first point where this shot hits P is called the *hit point* of the ray shot. The line segment from a shooting point x to its hit point x' is called a *chord* of the polygon. A *weakly-visible chord* is a chord that is weakly visible from the rest of the polygon. A *weakly-visible segment* is simply any line segment in $P \cup int(P)$ that is weakly visible from the rest of the polygon.

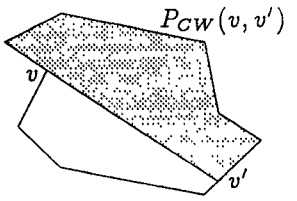

Figure 1: A clockwise component and its C-polygon

Each reflex vertex v defines two special ray shots as follows. If α is the direction from v^- to v, we let $\vec{r}_{CW}(v) = \vec{r}(v, \alpha)$ represent the *clockwise ray shot* from v. If v' is the hit point of the clockwise ray shot, then the subchain $P_{CW}(v, v')$ is the *clockwise component* of v (see Fig. 1). Counterclockwise ray shots and components are defined in the same way. A component is *redundant* if it is a superset of another component.

The clockwise component of v also defines a subpolygon called the *clockwise C-polygon* of v, which is the subpolygon of P bounded by the polygonal chain $P_{CW}(v, v')$ and the chord $\overline{vv'}$. The clockwise C-polygon of v is shown as a shaded region in Fig. 1. The counterclockwise C-polygons are defined in a similar fashion. We define the *envelope* of a C-polygon to be the boundary (in the interior of P) of the C-polygon of v. Note that the envelope of a C-polygon is simply the straight-line segment (chord) consisting of the ray shot from v. Later, we will refer to the *envelope* of an intersection of a set of C-polygons, which is the boundary (inside of P) of the intersection of the set of C-polygons.

3 Preliminaries

In this section we describe some of the geometric properties of a weakly-visible line segment.

It was noted in [IK] that the family of non-redundant components completely determines LR-visibility of P, since a pair of points s and t admits LR-visibility if and only if each non-redundant component of P contains either s or t. A similar result from [DHN2] states that the family of non-redundant components also determines all weakly-visible chords, since a chord $\overline{st}$ is a weakly-visible chord if and only if each non-redundant component of P contains either s or t. Lemma 1 below shows that the family of non-redundant components also determines the family of weakly-visible segments, while lemma 2 describes another property satisfied

by all weakly-visible segments in P.

Lemma 3.1 *A line segment $\overline{uv} = l$ is weakly visible from P iff the segment l intersects the C-polygon corresponding to every non-redundant component of P.*

Lemma 3.2 *If a line segment $\overline{uv} = l$ is weakly visible from P, then the chord l', which is obtained by extending l in both directions until it hits P, is a weakly-visible chord, and the endpoints of l' form a LR-visible pair of points with respect to P.*

The obvious implication of Lemma 3.2 is that a polygon has at least one weakly-visible chord iff it has at least one weakly-visible segment and consequently a shortest weakly-visible segment.

Before giving an overview of the algorithm, we describe the peculiar output of the $O(n)$-time algorithm for computing *all weakly-visible chords* of a polygon [DHN2] (the chords algorithm, in short), since this algorithm is used by our scheme. The chords algorithm generates $k = O(n)$ pairs of the form (A_i, B_i). Note that for the rest of the paper k will be used to denote the number of pairs output by the chords algorithm. Here A_i is a line segment on P that is described in terms of a parameter x (for example, $x = 0$ (1) refers to the left (resp. right) endpoint of A_i). B_i is a polygonal chain on P such that every line segment joining a point on A_i and a point on B_i forms a weakly-visible chord. However, B_i has endpoints that are linear functions of the parameter x. It may also be noted that each of the A_is are disjoint line segments, while the B_is are possibly overlapping polygonal chains.

4 Overview of algorithm

The first step of our algorithm is to run the chords algorithm. If the polygon has no weakly-visible chords, then the algorithm stops and declares that there are no weakly-visible segments either.

Note that for the rest of the paper all references to the term components are references to non-redundant components. For every A_i output by the chords algorithm, let SA_i be the set of components that contain A_i, and let CA_i be the intersection of the C-polygons corresponding to the components in SA_i. Let the envelope of CA_i be denoted by α_i. Let SB_i be the set of all components not in SA_i, and let CB_i be the intersection of the C-polygons of all the components in SB_i. Let β_i denote the envelope of CB_i.

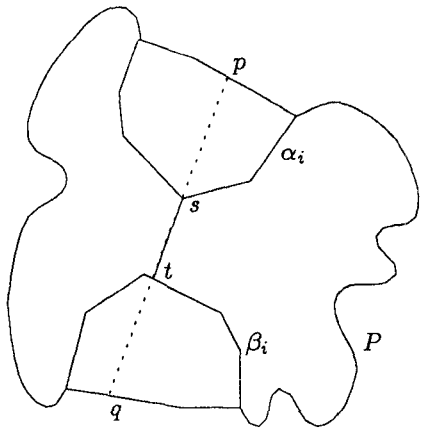

Figure 2:

It is clear that any line segment inside P that touches both α_i and β_i must intersect every C-polygon and by lemma 3.1 must be a weakly-visible segment. However, the converse is not so obvious. Lemma 4.1 below, which implies the converse, proves that the shortest weakly-visible segment must join a point on α_i and a point on β_i, for some i. This vital property is necessary to make our algorithm work in linear time. Note that SA_i corresponds to a subsequence of the sorted sequence of components. What lemma 4.1 proves is that only such *subsequences* (and not an arbitrary *subset*) of non-redundant components need to be considered for computing the shortest weakly-visible segment.

Lemma 4.1 *If $\overline{st} = l$ is the shortest weakly-visible segment, then s must lie on α_i and t must lie on β_i, for some $i = 1, \ldots, k$.*

Proof:

Extend the line segment $l = \overline{st}$ until it hits P at points p and q as shown in Fig. 2. By Lemma 3.1, segment l must intersect every C-polygon. This implies that for every C-polygon C, one of the two line segments $\overline{ps}$ and $\overline{qt}$ must lie completely in C. Also, if the envelope of C intersects l', it must include precisely one of the two segments $\overline{ps}$ and $\overline{qt}$.

By the minimality of l, it is clear that the point s (t) lies on the envelope of one or two C-polygons. If s (resp. t) lies on the envelope of some C-polygon C_s (resp. C_t), then since the envelope of C_s (resp. C_t) intersects l', C_s (resp. C_t) includes the point p (resp. q), and does not include q (resp. p).

Let $l' = \overline{pq}$. By Lemma 3.2, l' is a weakly-visible chord. Consider the output of the chords algorithm.

Let $p \in A_i$. By the correctness of the chords algorithm, $q \in B_i$. Note that C_s must include all of A_i because of the way the chords algorithm works. Thus, C_s must be a C-polygon that determines α_i, and C_t must be a C-polygon that determines β_i. Hence s must lie on α_i and t must lie on β_i thus concluding the proof. □

The above lemma suggests the following skeleton for our algorithm. For every $i = 1, \ldots, k$, construct the envelopes α_i and β_i, and then compute the shortest line segment joining a point on α_i and a point on β_i. The shortest of these is the shortest weakly-visible segment.

Note that since both α_i and β_i are convex polygonal chains, computing the shortest line segment connecting them can be computed in time $O(|\alpha_i| + |\beta_i|)$, where $|\alpha_i|$ and $|\beta_i|$ are the lengths of the two chains. However, in general there may be considerable overlap between α_i and α_{i+1}, as well as between β_i and β_{i+1}. For the i-th iteration, instead of simply finding the shortest line segment that joins α_i and β_i, the algorithm only scans the portion of α_i that is not part of α_{i+1} (and the portion of β_i that is not part of β_{i+1}), and finds the shortest segment between those parts. The assumption is that the rest of the portions of the two polygonal chains will be scanned as part of a later iteration. Repetitious scanning of the polygonal chains is thus prevented by delaying the scanning of overlapping portions as much as possible.

In section 5.1, we precisely characterize how α_i changes to become α_{i+1}, and correspondingly how β_i changes to become β_{i+1}. In section 5.2, we describe a data structure that stores $\alpha_i, i = 1, \ldots, k$, and another identical structure that stores $\beta_i, i = 1, \ldots, k$. The planarity of these data structures is sufficient proof that the size of the union of α_i and the size of the union of β_i for $i = 1, \ldots, k$ is $O(n)$.

The problem with the skeleton algorithm described above is that the shortest line segment joining α_i and β_i for some i may not lie entirely within P. This happens because even though the line segment when extended may hit A_i, it might not hit B_i because of obstruction from the rest of P, i.e., the extended line is not a weakly-visible chord. In this case, if there is a weakly-visible chord connecting A_i and B_i, then the shortest weakly-visible segment joining α_i and β_i would touch a vertex of the P. This suggests that our algorithm needs to deal with two main cases. The first case is when the shortest illuminating segment does not touch a vertex of P except at its endpoints; the second case is when it touches a vertex of P in its interior. If the first case occurs, the algorithm briefly described earlier will output the shortest illuminating segment. The details of this case are described in the next section. The second case is handled separately in section 6. The algorithm for the second case is a modification of our earlier algorithm for computing all weakly-visible chords of a polygon [DHN2]. If a weakly-visible segment touches a vertex of P it is referred to as a *non-tangential weakly-visible segment*; otherwise it is referred to as a *tangential weakly-visible segment.*

By putting all the pieces together, we show a linear-time algorithm to obtain the shortest non-tangential weakly-visible segment, and a linear-time algorithm to compute the shortest tangential weakly-visible segment. The shortest of the two segments is the shortest weakly-visible segment in a polygon, thus giving us the desired algorithm.

5 Case 1: Non-tangential weakly-visible segment

As mentioned earlier, this case corresponds to the situation when the shortest weakly-visible segment does not touch any vertex of the polygon except possibly at its endpoints. For each $i = 1, \ldots, k$, let SN_i be the shortest non-tangential weakly-visible segment that joins α_i and β_i with at least one endpoint on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$. It can be shown that the shortest of the segments SN_i, $i = 1, \ldots, k$ must be the shortest non-tangential weakly-visible segment that joins α_i and β_i, $i = 1, \ldots, k$.

5.1 Structure of α_i and β_i

As mentioned earlier both α_i and β_i are the boundaries (internal to P) of the intersection of a set of C-polygons. Hence it is clear that both of them are convex polygonal chains. It may be possible that $\alpha_i = \alpha_{i_1} = \alpha_{i_2} = \cdots = \alpha_{i_p}$. This simply means that no component starts or ends on the portion of P covered by $A_{i_1}, A_{i_2}, \ldots, A_{i_p}$.

We now describe the structural differences between α_i and α_{i+1} (in case they do differ), and the corresponding differences between β_i and β_{i+1}. The main purpose of studying this structure is to identify the polygonal chains $\alpha_i - \alpha_{i+1}$ and $\beta_i - \beta_{i+1}$ so that they can be processed in the i-th iteration. Clearly, if $\alpha_i = \alpha_{i+1}$, then no processing is required in iteration i.

Assume that $\alpha_i \neq \alpha_{i+1}$. From [DHN2] we know

that A_i and A_{i+1} are disjoint line segments. There are various events that trigger the chords algorithm to go from iteration i to iteration $i+1$, thus outputting pairs (A_i, B_i) and pairs (A_{i+1}, B_{i+1}). An event occurs if a component starts or ends between A_i or A_{i+1}. The other possible events have to do with changes in the points of tangency for the boundaries of the weakly-visible chords. The chain α_i is different from α_{i+1} only when a component starts or ends between A_i and A_{i+1}. For the next three paragraphs we will assume that the counterclockwise end for any polygonal chain is the *front* end, while the clockwise end is the *tail* end.

If a component c starts between A_i and A_{i+1}, the changes from α_i to α_{i+1} are as shown in Fig. 3(b). Note that the C-polygon corresponding to component c lies to the left of the line and that the component c consists of the polygonal chain $P_{CW}(p_2, q_2)$. A_i lies to the right of p_2, while A_{i+1} lies to the left of p_2. α_i consists of the chain from a_t to s_2 to a_f, while α_{i+1} consists of the chain from p_2 to s_2 to a_f, i.e., a portion of the tail of α_i gets replaced by a portion of the ray shot corresponding to the component c. At the same time, as shown in Fig. 3(b) β_i has a portion of its front replaced by a new polygonal chain. β_i consists of the chain from b_t to t_2 to q_2, while β_{i+1} consists of the chain from b_t to t_2 to b_f. In other words, CA_i shrinks at its tail end, and CB_i grows at its front end, while both their boundaries remain convex. Note that $\alpha_i - \alpha_{i+1}$ comprises of the polygonal chain from a_t to s_2, while $\beta_i - \beta_{i+1}$ comprises of the segment from q_2 to t_2.

Note that in the Figs. 3(a) and (b), the region $CA_i \cap CA_{i+1}$ (as well as the region $CB_i \cap CB_{i+1}$) have been shown as a filled region. The area occupied by CA_{i+1} (but not by CA_i) is indicated as a dot-filled region, while the area occupied by CA_i and CB_i is left blank.

By a similar argument, if a component c ends between A_i and A_{i+1}, the portion of the ray shot corresponding to c at the front (right end or the counterclockwise end) of α_i gets replaced by a new polygonal chain, causing CA_i to grow in the front. As shown in figure 2(a), β_i has a portion of its tail (right end or counterclockwise end) replaced by a portion of the ray shot corresponding to c, thus causing CB_i to shrink at its tail end. In this case note that $\alpha_i - \alpha_{i+1}$ comprises of the segment from p_1 to s_1, while $\beta_i - \beta_{i+1}$ comprises of the chain from b_t to t_1.

The above description describes the changes that take place to the α and β chains while moving from the i-th iteration to the $(i+1)$-st iteration.

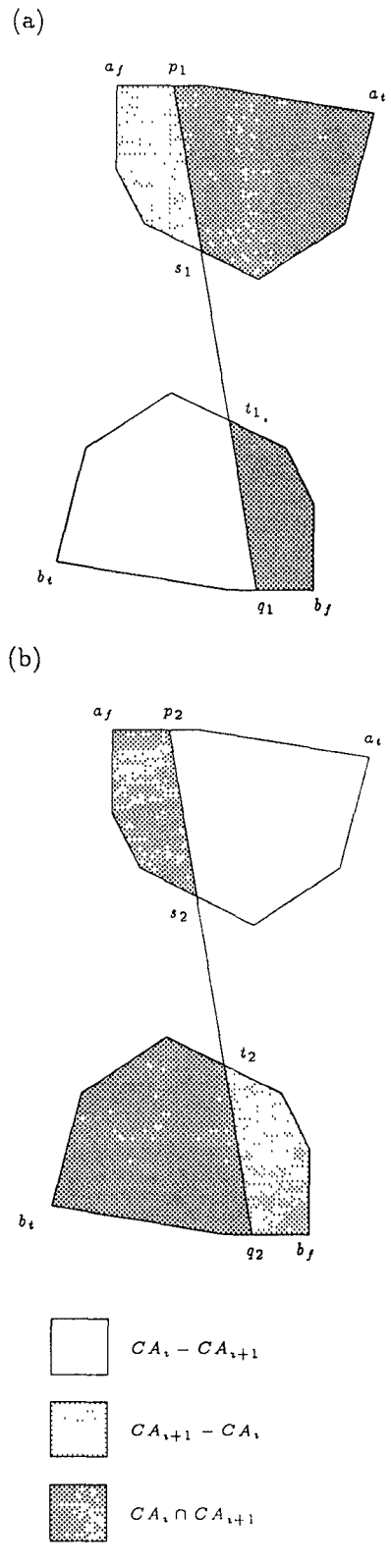

Figure 3: Changes in the structure of α_i and β_i

5.2 Data structure for storing the α and β chains

The union of all the chains α_i (β_i) form a tree. We claim here without elaborating that both the sets of chains α_i and β_i can be stored in simple linear-sized data structures that are similar to the *persistent search trees* of Sarnak and Tarjan [ST]. The data structure will be detailed in a full version of this paper.

5.3 Computing SN_i

As described in the previous section, the algorithm goes through k iterations. In the i-th iteration, the chains $\alpha_i - \alpha_{i+1}$ and $\beta_i - \beta_{i+1}$ are identified, and the shortest segment that joins α_i and β_i with one endpoint on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$ is computed. Let this segment be $SN_i = \overline{s_i t_i}$.

Given any two convex polygonal chains α and β, there is a simple sweep algorithm to find the shortest line segment that joins the two chains. In this case, α and β are two convex chains that form part of the boundary of two disjoint convex polygons. The algorithm involves sweeping the two chains, one from its clockwise end and in counterclockwise order, the other from its counterclockwise end in clockwise order. Informally speaking, the sweep algorithm works because of three simple facts: (1) for a fixed point $a \in \alpha$, its distance to visible points $b \in \beta$ is unimodal, (2) as point a moves monotonically on α, its closest point on β moves monotonically on β, (3) for points $a \in \alpha$, its shortest distance to β (i.e., the distance to its closest point on β) is unimodal. Intuitively speaking, fact (3) states that the *local minimum* is also the *global minimum.*

In the i-th iteration, it is possible that the segment $\overline{s_i t_i}$ may not lie entirely within P. To identify this situation, we exploit the fact that given a point x on P, the chords algorithm has already identified which directions from x give rise to weakly-visible chords. Hence to check whether SN_i lies in P, the algorithm computes the endpoint of the chord generated when the line segment is extended. Call this point p_i. Using the output of the chords algorithm our algorithm checks whether the chord in the direction $\overline{p_i s_i}$ is a weakly-visible chord. It should be pointed out that it is possible that p_i may not lie on A_i, but on some other segment A_j. To identify A_j, the algorithm traverses from A_i to A_j along P. It is non-trivial to show that this portion of P is not traversed again for this purpose (and hence does not contradict the claimed $O(n)$ time complexity). The intuition behind the claim is that if p_i lies on A_j then SN_i is also the shortest segment between α_j and β_j as well as between α_l and β_l for all values of l between i and j. On the other hand, if the chord is not weakly visible, then the segment $\overline{s_i t_i}$ is ignored, and will be handled by the second phase of the algorithm (corresponding to Case 2).

Another subtle complication is introduced by the possibility that SN_i may have one endpoint on $\beta_i - \beta_{i+1}$ and another endpoint on $\alpha_i \cap \alpha_{i+1}$ (instead of $\alpha_i - \alpha_{i+1}$). This could happen if the sweep algorithm (described at the start of this subsection) for finding the shortest line segment joining two convex polygonal chains reaches the end of one of the chains without hitting a *local minimum.* For example, assume that the end of $\alpha_i - \alpha_{i+1}$ is reached before reaching the end of $\beta_i - \beta_{i+1}$ and before a minimum was encountered. In this case, our algorithm continues sweeping on $\beta_i - \beta_{i+1}$, while continuing the sweep on $\alpha_i \cap \alpha_{i+1}$. Our algorithm needs to be modified to ensure that this portion of $\alpha_i \cap \alpha_{i+1}$ is not swept again in iteration $i+1$ (or later). In this case, it can be shown that SN_j cannot have an endpoint on this portion of $\alpha_i \cap \alpha_{i+1}$ and hence need not be considered in any later iteration. The relevant portion of $\alpha_i \cap \alpha_{i+1}$ is marked *visited* so that a sweep in a later iteration can skip over this portion of the chain. The entire arguments in this paragraph could have been carried out with α replaced by β and vice versa.

Once a local minimum is found for the i-th iteration, the algorithm also verifies if it is a global minimum for the shortest segment between α_i and β_i. If the point on $\alpha_i - \alpha_{i+1}$ or $\beta_i - \beta_{i+1}$ is not the endpoint of that subchain, then the global minimum for the shortest segment between α_i and β_i must have been reached. Otherwise, a simple test can check whether the global minimum has been reached or not. If it is not a global minimum, then SN_i can be ignored since the shortest segment between α_i and β_i connects points that are not on $\alpha_i - \alpha_{i+1}$ as well as $\beta_i - \beta_{i+1}$. Since such a segment would connect α_{i+1} and β_{i+1} it will be encountered in a later iteration. The algorithm with the minor modifications mentioned above is guaranteed to sweep every portion of the α and β chains exactly once and hence achieves the claimed linear-time complexity.

6 Case 2: Shortest tangential weakly-visible segment

This case occurs when the interior of the shortest weakly-visible segment in the polygon touches a ver-

tex of the polygon. However, in this case, the corresponding weakly-visible chord obtained by extending the segment is also a tangential chord, i.e., it touches a vertex of the polygon in its interior. The crucial point to observe is that these are exactly the weakly-visible chords that are output by the linear-time chords algorithm [DHN2]. A suitable modification of the chords algorithm can output all tangential weakly-visible segments, of which the shortest can be computed.

The chords algorithm uses the following strategy. It traverses along the polygon in a counterclockwise direction with a point x. When x is on A_i, the points $y(x)$ and $z(x)$ corresponding to the other endpoints of the two tangential chords from x are computed. The points $y(x)$ and $z(x)$ move monotonically on P; so do the points of tangency for the tangential chords, namely $s(x)$ and $t(x)$. As x moves on A_i, there are several possible events that can take place, which would change the description of the tangents: the point $y(x)$ (or $z(x)$) could move to a vertex of P; the point $s(x)$ (or $t(x)$) could move to a vertex of P. These events cause a recomputation of the equations of the tangential chords as a function of x. In [DHN2] we show that the number of these events are $O(n)$, thus giving us a linear-time algorithm.

The modification for computing the tangential weakly-visible segments is as follows. During iteration i, the chains α_i and β_i are computed. When the point x is on A_i, the points of intersection of the tangential chords with α_i and β_i are also maintained (call them $a_1(x), a_2(x), b_1(x), b_2(x)$). The segment from $a_1(x)$ to $b_1(x)$ and the segment from $a_2(x)$ to $b_2(x)$ are the two tangential weakly-visible segments with respect to x. The situation is described in Fig. 4. There are, however, an additional number of events that could cause a change in the description of the tangential weakly-visible segments: the points $a_1(x)$ or $a_2(x)$ ($b_1(x)$ or $b_2(x)$) could move to a vertex of α_i (β_i). This would cause additional recomputations of the equations as well as the lengths of the tangential segments. The crucial point is that in between events, the length of the tangential segments can be computed in terms of x, from which the minimum can be computed for that interval in constant time. It can also be shown that the total number of events that will be encountered is $O(n)$ and that $a_1(x), b_1(x), a_2(x), b_2(x)$ move monotonically on the α and β chains.

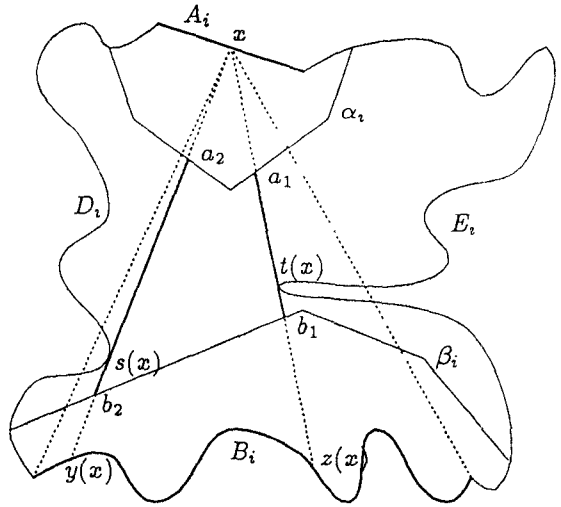

Figure 4: Case 2: Determining tangential shortest weakly-visible segments

7 All minimal weakly-visible segments algorithm

One of the by-products of our algorithm is a linear-time algorithm to generate all minimal weakly-visible segments of a polygon. This algorithm is a modification of the algorithm described in section 6 for computing the shortest tangential weakly-visible segment. It outputs a set of pairs $(U_i, V_i), i = 1, \ldots, m$. Here U_i and V_i are subchains of the polygonal chains α and β, $m = O(n)$, and any segment joining points $u \in U_i$ and $v \in V_i$ is a minimal weakly-visible segment. One note of caution is that U_i and V_i have left and right endpoints that are linear functions of a parameter x in a spirit similar to that of the endpoints of the chain B_i that is output by the chords algorithm. For a point x on A_i, the polygonal chains α_i and β_i can be computed along with the points $a_1(x), a_2(x) \in \alpha_i$ and $b_1(x), b_2(x) \in \beta_i$. The output of the algorithm consists of $(U_i, V_i) = ((a_1(x), a_2(x)), (b_2(x), b_1(x)))$. The discussion at the end of section 6 can also be used to show that the number of these pairs produced is $m = O(n)$. Lemma 1 can be used to show that these segments are minimal in the sense that any subsegment of these segments is not weakly visible.

8 Conclusion and open problems

We show optimal linear-time algorithms to compute the shortest weakly-visible segment and all minimal weakly-visible segments in a given simple polygon. Some interesting open questions are:

- Can the *exhaustive* sweeping techniques from this paper be used to solve other weak visibility problems efficiently? For example, are there linear-time algorithms for the *all-pairs* version of any of the 2-guard walk problems (see [DHN1])?
- Can the algorithm from this paper be designed without using the triangulation or the shortest path algorithm as a subroutine?
- Ntafos [Nt] introduced the notion of *d–visibility*, where an observer's visibility is limited to distance *d*. Can the shortest illuminating segment be computed efficiently under *d*–visibility?

References

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

[BMT] B. K. Bhattacharya, A. Mukhopadhyay, and G.T. Toussaint, "A linear-time algorithm for computing the shortest line segment from which a polygon is weakly externally visible," in *Proc. of 2nd WADS*, 1991, pp. 412-424.

[BM] B. K. Bhattacharya, and A. Mukhopadhyay, "Computing in linear time an internal line segment from which a simple polygon is weakly internally visible," Personal Communication, 1993.

[Cha] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, **6** (1991), pp. 485-524.

[GHLST] L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. Tarjan, "Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons," *Algorithmica*, **2** (1987), pp. 209-233.

[Che] D.Z. Chen, "Optimally computing the shortest weakly visible subedge of a simple polygon," Tech. Report No. 92-028, Dept. of Computer Sciences, Purdue University, May 1992.

[CN] W.P. Chin, and S. Ntafos, "Shortest Watchman Routes in Simple Polygons," *Discrete and Computational Geometry*, **6** (1991), pp. 9-31.

[DHN1] G. Das, P. Heffernan and G. Narasimhan, "LR-visibility in polygons,", *Proc. 5th Canadian Conference on Computational Geometry*, (1993), pp. 303-308.

[DHN2] G. Das, P. Heffernan and G. Narasimhan, "Finding all weakly-visible chords of a polygon in linear time," manuscript, 1993.

[DC] J. Doh and K. Chwa, "An algorithm for determining internal line visibility of a simple polygon," *J. of Algorithms*, 14 (1993), pp. 139-168.

[H] P. J. Heffernan, "An optimal algorithm for the two-guard problem," *Proc. Ninth Annual ACM Symp. on Computational Geometry*, 1993, pp. 348-358; to appear in *Inter. J. on Computational Geometry and Applications.*

[IK] C. Icking and R. Klein, "The two guards problem," *Inter. J. on Computational Geometry and Applications*, **2** (1992), pp. 257-285.

[Ke87] Y. Ke, "Detecting the weak visibility of a simple polygon and related problems," Tech. Report, The Johns Hopkins University, (1987).

[LP] D. T. Lee and F. P. Preparata. "An optimal algorithm for finding the kernel of a polygon," *JACM*, **26**(3), (1979), pp. 415-421.

[Nt] S. Ntafos, "Watchman Routes Under Limited Visibility," *Computational Geometry: Theory and Applications*, **1**(3) (1991), pp. 149-170.

[O'R] J. O'Rourke, "Computational geometry column 18," *SIGACT News*, **24** (1993), pp. 20-25.

[PV] P. Pradeep Kumar, and C.E.Veni Madhavan, "Shortest watchman tours in weak visibility polygons,", *Proc. 5th Canadian Conference on Computational Geometry*, (1993), pp. 91-96.

[SS] J.-R. Sack and S. Suri, "An optimal algorithm for detecting weak visibility," *IEEE Transactions on Computers*, **39** (1990), pp. 1213-1219.

[ST] Sarnak and R. Tarjan, "Planar Point Location Using Persistent Search Trees", *CACM*, **29**, (1986).

[TL] L.H. Tseng and D.T. Lee, "Two-guard walkability of simple polygons," manuscript, 1993.