Finding All Weakly-Visible Chords of a Polygon in Linear Time

(Extended Abstract)

Gautam Das * and Paul J. Heffernan and Giri Narasimhan **

Memphis State University, Memphis, TN 38152.

Abstract. A chord of a simple polygon P is *weakly-visible* if every point on P is visible from some point on the chord. We give an optimal linear-time algorithm which computes *all* weakly-visible chords of a simple polygon P with n vertices.

1 Introduction

In this paper we present an optimal-time algorithm which computes all *weakly-visible* chords of a simple polygon. For a simple polygon P with n vertices, our algorithm requires time $O(n)$. Previous results [3, 9] require $O(n \log n)$ time and compute only one weakly-visible chord. We also consider the case of rectilinear simple polygons, and show a much simpler linear-time algorithm to compute all weakly-visible chords of the polygon.

Two sets of points are said to be *weakly-visible* if *every* point in either set is visible from *some* point in the other set. A weakly-visible chord c of a polygon P is one such that c and P are weakly-visible. We state four versions of the weakly-visible chords problem for a polygon P: (1) determine whether a given chord c is weakly-visible; (2) determine whether there exists a weakly-visible chord; (3) return a weakly-visible chord c, if indeed such a chord exists; and (4) return *all* weakly-visible chords. Version 4 is the strongest, and an algorithm for it also solves the first three versions. In this paper we solve to optimality version 4 and prove the theorem given below. Although a polygon can have an infinite number of weakly-visible chords, the output can be described in a piece-wise manner using only $O(n)$ space as described later in the paper. In earlier papers [3, 9], version 3 has been solved in $O(n \log n)$ time.

Theorem 1 *Given a simple polygon P, there exists a linear-time algorithm that computes all weakly-visible chords of P.*

The question of weakly-visible chords falls in the larger area of weak-visibility in polygons, which has received much attention by researchers. A simple polygon P is weakly-visible from an edge e if e and $P \setminus e$ are weakly-visible. Any two points x and y of a polygon P partition P into two chains, which we call L and R, for left and right chains. A polygon is *LR-visible* for x and y if L and R are weakly-visible. A weakly-visible chord c of P is one such that c and P are weakly-visible. Weak-visibility of a

* *e-mail:* dasg@next1.msci.memst.edu; Supported in part by NSF Grant CCR-930-6822

** *e-mail:* giri@next1.msci.memst.edu; Supported in part by NSF Grant INT-911-5870

polygon from an edge was first studied in [1], and Sack and Suri [11] subsequently gave a linear-time algorithm which computes all weakly-visible edges of a simple polygon. Chen [2] gave a linear-time algorithm that finds the shortest weakly-visible edge, if one exists. An $O(n \log n)$-time algorithm that computes all LR-visible pairs x and y is given by Tseng and Lee [12], and Das, Heffernan and Narasimhan [4] subsequently gave a linear-time algorithm that computes all LR-visible pairs s and t. The weakly-visible chords problem was studied in [3, 9], and algorithms were developed which require $O(n \log n)$ time and compute only one weakly-visible chord. In this paper we present a linear-time algorithm which computes all weakly-visible chords.

This paper is of interest not only because we present an optimal result for an intriguing problem in polygonal visibility, but also on account of the techniques we employ, and because of the relationship between weakly-visible chords and other problems in polygonal visibility, such as LR-visibility. LR-visibility is a subproblem of weakly-visible chords, for it can be shown that two points x and y of P are the endpoints of a weakly-visible chord of P if and only if $\overline{xy}$ is a chord of P and P is LR-visible with respect to x and y. In the current paper, the linear-time algorithm for computing all LR-visible pairs in [4] is used as a subprocedure.

What is interesting about the techniques used here and in [4] is that both the linear-time algorithms output a mass of information, which when sifted appropriately can provide a wealth of visibility information for a simple polygon. Furthermore, the result and techniques reported here were used effectively by Das and Narasimhan [5] to solve to optimality the problem of finding the shortest weakly-visible segment (if one exists) in the interior of a simple polygon.

Another problem which is closely related to weak-visibility problems is the *two-guard* problem. While the two-guard problem has many formulations, we will state just one for the sake of illustration: a polygon P is walkable from point x to point y if one "guard" can traverse the left chain L and the other the right chain R from x to y while always remaining co-visible. Other formulations require the guards to move monotonically or that one guard traverses from y to x. For the two-guard problem, currently there exist optimal linear-time algorithms for various formulations for fixed x and y (version 1) [7], and $O(n \log n)$-time algorithms which find all pairs x and y (version 4) for various formulations [12]. The authors are currently working to develop optimal solutions for the all-pairs version (version 4) of various formulations of the two-guard problem, and we feel that our recent efforts are important steps towards this goal.

2 Preliminaries

In this section we define notation for this paper, and summarize the LR-visibility algorithm in [4], which will be used as a subprocedure. A *polygonal chain* in the plane is a concatenation of line segments or *edges* that connect *vertices*. 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, and its interior, $int(P)$. Two points $x, y \in P$ are *visible* if $\overline{xy} \subset P \cup int(P)$, i.e., $\overline{xy}$ is a ***chord*** of P. For $x, y \in P$, $P_{CW}(x, y)$ ($P_{CCW}(x, y)$) is the subchain obtained by traversing P clockwise (counterclockwise) from x to y.

The ***ray shot*** from a vertex v in direction d consists of "shooting" a "bullet" from v in direction d which travels until it hits a point of P. Formally, if r is the ray rooted at v in direction d, then the ***hit point*** of this ray shot is the point of $(P \setminus \{v\}) \cap \mathrm{r}$ closest to v. Each reflex vertex defines two special ray shots as follows. Let v be a reflex vertex and v'' the vertex adjacent to v in the clockwise direction. Then the ray shot from v in the direction from v'' to v is called the ***clockwise ray shot*** of 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. Counterclockwise ray shots and components are defined in the same way. A component is ***redundant*** if it is a superset of another component.

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 at a common point. As noted in [8], the family of components completely determines LR-visibility of P, since a pair of points x and y admits LR-visibility if and only if each component of P contains either x or y. The definition of redundant gives the following.

Lemma 1 *A polygon P is LR-visible with respect to s and t if and only if each non-redundant component of P contains either s or t.*

If a polygon has more than two disjoint components, this lemma shows that it is not LR-visible. The LR-visibility algorithm in [4] outputs $O(n)$ pairs of subchains of the form (A_i, B_i) such that any point s on a subchain A_i is LR-visible to any point t on the corresponding subchain B_i. We now describe A_i and B_i more rigorously. The endpoints of non-redundant components partition P into a collection of intervals that we call ***basic intervals***, and denote $A_1, \cdots, A_k$, ordered counterclockwise. (In the rest of the paper, we use the term ***interval*** to denote a subchain of the polygon's boundary). A basic interval may or may not contain either of its endpoints. It is possible for a degenerate basic interval consisting of a single point to exist. By lemma 1, all points of a basic interval form LR-visible pairs with the same collection of partners. Thus, we denote as B_i the set of points such that (x, y) is a LR-visible pair for all $x \in A_i$ and $y \in B_i$. The following two lemmas are proved in [4].

Lemma 2 *B_i is a connected set; that is, it is either the entire polygon P, or the empty set, or a non-empty subinterval of P composed of the union of adjacent basic intervals.*

Lemma 3 *Also, if $A_i \cap B_i \neq \emptyset$, then $B_i = P$.*

In [4], we gave a linear-time algorithm that constructs all LR-visible pairs of intervals $(A_1, B_1), \cdots, (A_k, B_k)$. The intervals $A_1, \cdots, A_k$ are disjoint and ordered counterclockwise on P. The intervals $B_1, \cdots, B_k$ are also ordered counterclockwise but are not necessarily disjoint. As one moves counterclockwise from A_i to A_{i+1}, one either leaves or enters a non-redundant component, which may result in either the starting or ending endpoint of B_i to move counterclockwise in order to form B_{i+1}. In the remaining sections we develop the algorithm for constructing weakly-visible chords. Additional notation is introduced where required.

3 Geometric Properties of the Problem

In this section we present the important geometric properties on which the algorithm depends. The actual algorithm is presented in the next section. The following lemma relates weakly-visible chords to LR-visibility and has a trivial proof.

Lemma 4 *For points x and y on P, the segment $\overline{xy}$ is a weakly-visible chord of P if and only if (1) $\overline{xy}$ is a chord of P and (2) P is LR-visible for x and y.*

The lemma suggests the algorithm: first compute LR-visibility using the algorithm in [4], then compute all chords $\overline{xy}$ such that $x \in A_i$ and $y \in B_i$.

We now develop further properties of basic intervals necessary for this task. The ***kernel***, K, of a polygon P is defined as the collection of points in $P \cup int(P)$ which are visible from all points of P. The kernel is a convex set and a polygon with a non-empty kernel is called ***star-shaped***. The points of P in the kernel are exactly those which intersect all components. It is clear that a point $x \in K \cap P$ forms a weakly-visible chord with every other point of P. This means that the set of weakly-visible chords containing a kernel point as an endpoint can be succinctly represented as the set $K \cap P$.

Lemma 5 *For a basic interval A_i, $B_i = P$ if and only if A_i is contained in K.*

Proof. Let $x \in A_i$. If $B_i = P$ then (x, x) is an LR-visible pair, so x intersects all components and thus is in the kernel. Suppose $A_i \subset K$. Let $x \in A_i$. Then x intersects all components, so (x, x) is an LR-visible pair; thus $x \in B_i$, and since $A_i \cap B_i \neq \emptyset$ we have $B_i = P$. □

We know that each basic interval consists entirely of points in K or points not in K, and we call a basic interval that consists of kernel points a ***kernel interval***. A basic interval A_i which is not a kernel interval is disjoint from B_i; for such a case we define D_i as the interval of points encountered as one traverses counterclockwise from the ending point of A_i to the starting point of B_i, and E_i as the interval encountered counterclockwise from the ending point of B_i to the starting point of A_i. We call D_i and E_i the ***side intervals*** of A_i. The side intervals either contain or do not contain their endpoints in such a manner that the four intervals A_i, B_i, D_i and E_i partition P. It is possible for D_i and/or E_i to be empty.

As shown in [7], if w and v are points of P, and $SP(w, v)$ is the shortest path inside P directed from w to v, then any vertex of $SP(w, v)$ that lies on $P_{CW}(w, v)$ ($P_{CCW}(w, v)$) is a left (right) turn. We say that an interval F is ***well-behaved*** if the shortest path between its endpoints inside P only touches points of F and not the rest of the polygon. Thus if $P_{CCW}(w, v)$ is a well-behaved interval then $SP(w, v)$ contains no left turns. This is a stronger statement than simply saying that that $SP(w, v)$ is a convex chain, since it specifies the direction of any turns. The following lemmas prove that the A_is, D_is, and E_is are well-behaved, which is useful in their efficient computation as shown later.

Lemma 6 *Each non-kernel basic interval A_i is well-behaved.*

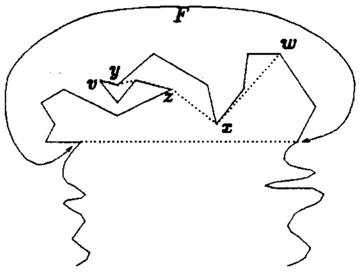

Fig. 1. Proof of lemma 6

Proof. Each basic interval is contained in some non-redundant component. Let $A_i = P_{CCW}(w, v)$, and let F be the non-redundant component containing it. Suppose $SP(w, v)$ contains a point z of $P_{CW}(w, v) \setminus \{w, v\}$. Let x (y) be the first point of $P_{CCW}(w, v)$ preceding (succeeding) z on $SP(w, v)$ (see Figure 1). The chord that forms F partitions P into two subpolygons, and since w and v are in the same subpolygon, any point on $SP(w, v)$ must also be in this subpolygon; thus $z \in F$. Since z is on $SP(w, v)$ it is a reflex vertex of P and therefore generates two components. The hit points of these components must both be on $P_{CCW}(x, y)$, as one can see by considering that the ray shots are contained in the subpolygon formed by $P_{CCW}(x, y)$ and $SP(x, y)$. Since z and its two hit points all lie on F, one of the two components generated by z is strictly contained in F, a contradiction of the fact that F is non-redundant. □

We next show that even the side intervals D_i and F_i are well-behaved. We first consider the case where P does not have two disjoint components.

Lemma 7 *Let P be a polygon with no two disjoint components. Then for each non-kernel A_i, the corresponding D_i and E_i are well-behaved.*

Proof. We first prove that any non-redundant chord is a weakly-visible chord. Suppose we have a non-redundant component F. Since no two components are disjoint, F intersects every other component. Furthermore, at least one endpoint of F intersects another component G unless G is nested inside F, which is not possible since F is non-redundant. Therefore the chord that forms F has endpoints which intersect every component of P, and thus by lemmas 1 and 4 this is a weakly-visible chord of P.

Thus there are (at least two) weakly-visible chords between each non-kernel A_i and its corresponding B_i. Any one of these weakly-visible chords has one endpoint on A_i and the other on B_i, and therefore separates D_i and E_i into different subpolygons. Thus, if D_i is not well-behaved it is because the shortest path between its endpoints contains a point of A_i or of B_i. Say it contains a point z of A_i that is not an endpoint of A_i. By an argument similar to that in the proof of lemma 6, z must be a reflex vertex whose hit points lie inside D_i. Thus z generates a component H that intersects both A_i and D_i yet contains neither. H does not intersect B_i, and does not contain the first point of A_i in counterclockwise order. This contradicts the fact that each point of A_i is LR-visible with each point of B_i. □

We now consider the case where P has at least two disjoint components. The following lemma is similar to the above, except that it somewhat less general.

Lemma 8 *Let P be a polygon with at least two disjoint components. If there exists a weakly-visible chord in P, or if there is more than one basic interval A_i with non-empty B_i, then each D_i and E_i is well-behaved.*

Proof. The proof of this lemma is more involved than lemma 7. In the proof we first show that if D_i (E_i) is not well-behaved, then it cannot be because of A_i or B_i. Then we show that if D_i (E_i) is not well-behaved due to obstruction by E_i (D_i), then the LR-visible pairs of points are not visible from each other, and consequently no weakly-visible chords exist. Details of the proof can be found in a full version of the paper. □

We introduce the following notation. Let $b(F)$ ($e(F)$) be the starting point (ending point) of an interval F encountered in the counterclockwise direction. The following lemma states that the visibility restrictions between points on A_i and points on B_i are imposed only by the side intervals.

Lemma 9 *Let x be a point on A_i and y a point on B_i. If the line $\overline{xy}$ intersects the polygon at a point $w \in P_{CCW}(x, e(A_i)) \cup P_{CCW}(b(B_i), y)$, then it also crosses D_i. Similarly, if $\overline{xy}$ intersects the polygon at a point $w \in P_{CCW}(y, e(B_i)) \cup P_{CCW}(b(A_i), x)$, then it also crosses E_i.*

By "crosses," we mean intersects such that there are points of D_i (E_i) on either side of the ray. The lemma essentially says that, if a point $x \in A_i$ cannot see a point $y \in B_i$, its visibility is blocked by D_i or E_i. The proof uses techniques similar to ones used in the previous lemmas, and we omit details in this version.

4 Computing Weakly-Visible Chords

4.1 Overview

We first give an overview of the algorithm. There are several preliminary steps. Our algorithm first constructs the kernel K using the linear time method of [10] and then constructs $K \cap P$. This latter step is easily accomplished since the algorithm of [10] can return the vertices of K which lie on P. If this is not empty, the algorithm outputs $(K \cap P, P)$ which describes all weakly-visible chords with one endpoint in the kernel.

Then the LR-visibility algorithm in [4] is run, which gives us the non-redundant components with endpoints in counterclockwise order, as well as the (A_i, B_i) pairs. If there is only one pair, we check if D_1 and E_1 are well-behaved, by running the shortest-path algorithm of [6] from their endpoints. If one of them is not well-behaved, the algorithm halts and reports that there are no weakly-visible chords.

We next determine in linear time whether there exist two disjoint components. Suppose we find that P does have a pair of disjoint components F and G. Any LR-visible pair of points must have one point on F and the other on G, so for any basic interval A_i in F we know that B_i is contained in G. It suffices to look only at A_i in F (and their corresponding B_i in G) and compute weakly-visible chords, if any. Suppose P does not have two disjoint components. In this case the algorithm examines each non-kernel A_i and its corresponding B_i in search of weakly-visible chords.

In either case, the basic step consists of determining the weakly-visible chords between a non-kernel A_i and its corresponding B_i. By the definition of these intervals we know that a point $x \in A_i$ forms an LR-visible pair with a point y if and only if $y \in B_i$. By lemma 4, therefore, we can construct the set of weakly-visible chords by determining for each point x of a basic interval A_i the points of B_i from which it is visible. Essentially, for each x we must determine the restrictions on its visibility with B_i. We also know that A_i, D_i and E_i are well-behaved, and visibility restrictions are only imposed by the side intervals. Below we describe how to compute chords between A_i and B_i and the transition from iteration i to $i+1$.

4.2 Implementation Details

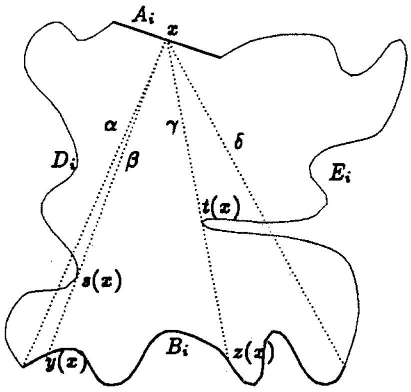

Fig. 2.

The visibility between a point $x \in A_i$ and B_i is determined by a pair of constructs we call the ***pseudo-tangents***. Consider Figure 2. The pseudo-tangent from x to D_i (E_i) is the unique line directed from x through a point $s(x)$ of D_i ($t(x)$ of E_i) such that all of D_i (E_i) lies on or to the left (right) of the line (pseudo-tangents are different from ***tangents***, which are also required to be chords). We denote the direction of the pseudo-tangent to D_i (E_i) by $\beta(x)$ ($\gamma(x)$). We give labels to two other special directions: the direction from x to $b(B_i)$ ($e(B_i)$) is denoted $\alpha(x)$ ($\delta(x)$). Sometimes we will use abbreviated notation, for example α instead of $\alpha(x)$, when the x under consideration is clear.

The point $x \in A_i$ is visible from some point of B_i if and only if the special directions satisfy the following relationship: $\alpha \leq_{ccw} \beta \leq_{ccw} \gamma \leq_{ccw} \delta$ (where $\leq_{ccw}$ means "precedes or equals in counterclockwise order, as viewed from x"). Let us consider the picture for a point $x \in A_i$ which is visible from B_i. In this event the pseudo-tangencies are actually tangencies, in the sense that they are chords of P. Let $y(x)$ ($z(x)$) be the other endpoint of the longest chord of P with one endpoint at x in direction β (γ). Since α, β, γ and δ are ordered counterclockwise, the points y and z lie on B_i. Points of B_i lying between $b(B_i)$ and y (z and $e(B_i)$) are not visible from x because visibility is blocked by D_i (E_i). However, all points of $P_{CCW}(y,z)$ are visible from x. Also, if $b(B_i)$ ($e(B_i)$) is the pseudo-tangent point of D_i (E_i), i.e. if $\alpha = \beta$ ($\gamma = \delta$), then $b(B_i)$ ($e(B_i)$) is also visible from x. Thus we see that if x is visible from B_i, its set of weakly-visible partners consists of a closed subinterval $P_{CCW}(y,z)$ of B_i, plus possibly one or both of the endpoints of B_i.

If x is not visible from B_i, then the four special directions are not ordered properly. If D_i (E_i) blocks all of B_i from x, then we have the subordering $\alpha \leq_{ccw} \delta <_{ccw} \beta$ ($\gamma <_{ccw} \alpha \leq_{ccw} \delta$). If the ordering is $\alpha \leq_{ccw} \gamma <_{ccw} \beta \leq_{ccw} \delta$, then neither D_i nor E_i totally block visibility individually, but together they do. In this case we can still define y and z as the extensions of the pseudo-tangencies until they hit B_i, where the opposite side interval is simply ignored. The fact that the ordering of β and γ is reversed means that z precedes y counterclockwise on B_i.

The algorithm uses the following strategy. We traverse P once counterclockwise with a point x, calculating for each x of a basic interval A_i the pseudo-tangent to the side interval D_i. We then traverse P once clockwise in order to compute for each x the pseudo-tangent to E_i. We determine for which x the ordering $\alpha \leq_{ccw} \beta \leq_{ccw} \gamma \leq_{ccw} \delta$ is obeyed, and for these x we compute the extensions y and z to obtain the partner interval $P_{CCW}(y,z)$. We also note whether $\alpha = \beta$ ($\gamma = \delta$) for any of these x, in which case $b(B_i)$ ($e(B_i)$) is also a partner. Since y and z are different for each x, and there are an infinitude of values of x, we must exhibit care in our manner of computing and storing the output; this issue will be addressed during the discussion below.

We describe the counterclockwise traversal of P with a point x, calculating for each x in A_i the pseudo-tangents to D_i; the procedure for E_i pseudo-tangents is symmetrical. If, for some x in A_i, β lies between α and δ, then we wish to compute the extension y. If $\alpha = \beta$ then the extension y equals $b(B_i)$. If $\delta <_{ccw} \alpha$ then x sees no point of B_i and the extension y is undefined.

We now show that the functions $\beta(x)$ and $y(x)$ are monotonic, i.e as x moves counterclockwise along the entire polygon, the direction $\beta(x)$ moves counterclockwise and $y(x)$ also moves counterclockwise along P. First, consider the motion of x within a basic interval A_i. As x moves counterclockwise the pseudo-tangent rotates counterclockwise around the ***pseudo-tangent point*** (the point on D_i which the pseudo-tangent touches), with the consequence that y (if defined) moves counterclockwise on B_i. The pseudo-tangent point may change to a point of D_i closer to $b(D_i)$, but this does not affect the monotone motion. Next consider the transition of x to the next basic interval. When x reaches $e(A_i) = b(D_i)$, the basic interval is updated to A_{i+1} and the side interval to D_{i+1}. The new side interval is obtained from D_i by subtracting A_{i+1} and possibly adding some additional basic intervals at the far end. There are two cases. In the first case (Figure 3(a)), the pseudo-tangents from x to both D_i and D_{i+1} are the same, hence $y(e(A_i)) = y(b(A_{i+1}))$. In the second case (Figure 3(b)), the pseudo-tangent from x to

D_{i+1} is counterclockwise to the the pseudo-tangent from x to D_i, hence $y(b(A_{i+1}))$ is counterclockwise of $y(e(A_i))$. Thus we see that $\beta(x)$ and $y(x)$ moves monotonically counterclockwise.

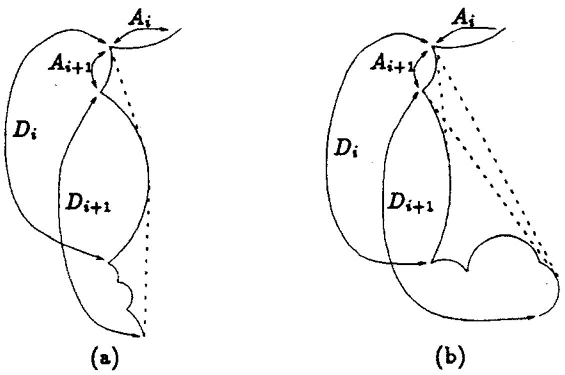

Fig. 3.

Another important observation regards the monotone motion of the pseudo-tangent point, $s(x)$. As x progresses counterclockwise within a basic interval A_i, s moves clockwise on D_i. Furthermore, when the side interval and the pseudo-tangent point are updated upon x reaching $e(A_i)$, the updating occurs in such a way that no point which has previously been the pseudo-tangent point can again become the pseudo-tangent point at a later time. This observation is clear from Figures 3(a) and (b), and is crucial in computing all pseudo-tangent points in overall linear time.

Since the side interval D_i is always well-behaved, the shortest path $SP(b(D_i), e(D_i))$ (which we will simply denote $SP(D_i)$) is a convex chain consisting of only right turns. This makes it easy to find and update tangencies. For $x = b(A_i)$, traverse $SP(D_i)$ from $e(D_i)$ until reaching the pseudo-tangent point (determining if a point of $SP(D_i)$ is the pseudo-tangent point is accomplished in constant time by comparing directions of the line from x with those of the adjacent edges). As x progresses counterclockwise we can update the pseudo-tangent point by continuing to traverse $SP(D_i)$ towards $b(D_i)$. No point of $SP(D_i)$ which is traversed will ever be a pseudo-tangent point in the future, and the total time of this operation is proportional to the number of vertices of $SP(D_i)$ traversed. The additional required ingredient, then, is an efficient manner of maintaining $SP(D_i)$ for the current D_i. We discuss this next.

Consider the first side interval D_1, which corresponds to the first basic interval A_1. It is composed of a collection of basic intervals $A_2, \cdots, A_j$. The side interval D_j is composed of the basic intervals $A_{j+1}, \cdots, A_l$, and is the first side interval that does not overlap with D_1. We will essentially deal with D_1 through D_j as a group. We will preprocess the entire group in a manner that allows efficient updating.

The preprocessing is the construction of the shortest path tree from $e(D_1)$ to all vertices of D_1. Since D_1 is well-behaved we perform this step in time proportional to the size of D_1, by a modification of the algorithm of [6]. At any time we will store $SP(D_i)$ in three pieces, namely as two shortest paths, $SP(b(D_i), e(D_1))$ and $SP(e(D_1), e(D_i))$, plus the bridge between them (see Figure 4). The chain $SP(b(D_i), e(D_1))$ is obtained from $SP(b(D_{i-1}), e(D_1))$ by using the shortest path tree. A depth-first search of this tree allows one to visit all vertices of D_1 according to the counterclockwise order on P, and thus allows us to maintain the current shortest path $SP(b(D_i), e(D_1))$. To maintain the other shortest path, $SP(e(D_1), e(D_i))$, we observe that updating from $SP(e(D_1), e(D_{i-1}))$ consists of adding zero or more basic intervals. Since each basic interval is well-behaved, the shortest path between its endpoints is convex. The bridge between two convex chains with a common endpoint can be found in time proportional to the number of vertices below the bridge on both chains. Since vertices below the bridge cannot be pseudo-tangent points, this time-complexity is acceptable to us.

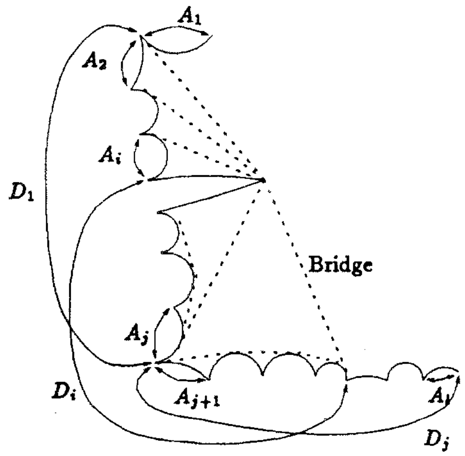

Fig. 4.

Thus, as x moves from A_{i-1} to A_i, we first update to obtain $SP(b(D_i), e(D_1))$ and then update to obtain $SP(e(D_1), e(D_i))$. Our true goal is $SP(D_i)$, and to obtain it we need the bridge between the two smaller shortest paths. Efficient computation and maintenance of the bridge is possible because the bridge endpoints display a property similar to that of the pseudo-tangents from x: the bridge endpoints progress monotonically clockwise, and no point which has previously been a bridge endpoint can again become a bridge at a later time.

In this manner we maintain the side interval while x traverses from A_1 to A_i, in total time proportional to the size of A_1 through A_l. If P has two disjoint components, then all side intervals intersect D, and a single iteration of this procedure suffices. If P does not have two disjoint components, then it is necessary to repeat the procedure, with the next step beginning with D_j and ending at D_l. This method "overlaps" portions of P, but no basic interval is used in more than two iterations, so maintaining $SP(D_i)$ while x traverses all of P requires only $O(n)$ time.

We discuss how $y(x)$ is efficiently computed and stored. Given x and the pseudo-tangent to D_i, we find y by extending the pseudo-tangent until it hits B_i. As x progresses counterclockwise, several events can occur. For example, (1) x can reach a vertex of A_i, (2) y can reach a vertex of B_i, or (3) the pseudo-tangent point s can pivot about an edge of $SP(D_i)$. Also, (4) when x reaches $e(A_i)$, both s and therefore y may need to be updated. Between events, however, we have an edge containing x and another containing y, and a point s which lies on $\overline{xy}$. Thus, even though we have an infinitude of values of x, each with a unique y, the function $y(x)$ can be described in constant time. A particular value $y(x)$ is found in constant time by computing the intersection of two lines.

In order to easily store $y(x)$, we introduce Steiner points at x and y (if x and/or y are not already vertices) whenever one of the events (1)-(4) above occurs. The total number of points introduced is $O(n)$. In this way, every edge of a non-kernel basic interval has a linear function $y(x)$.

A symmetric procedure has x traverse clockwise around P in order to compute $z(x)$ for each x. By merging the Steiner points introduced while computing $z(x)$ with those from the computation of $y(x)$, we have that for every edge of a non-kernel basic interval, $y(x)$ and $z(x)$ are linear functions. We check whether $y(x)$ precedes $z(x)$ counterclockwise for all x of the edge. For those x which violate this order, we return that they have no weakly-visible partners. For those x which obey the order, the weakly-visible partners are the points on the interval from $y(x)$ to $z(x)$.

A final consideration concerns $b(B_i)$ and $e(B_i)$. We stated that if $\alpha = \beta$ ($\gamma = \delta$) then $b(B_i)$ ($e(B_i)$) is a weakly-visible partner of x, even though it is outside the interval $P_{CCW}(y, z)$. Throughout the above procedure, then, $b(B_i)$ and $e(B_i)$ are stored separately as weakly-visible partners whenever appropriate.

5 The Rectilinear Case

In this section, we consider the simpler case where the given polygon P is a simple rectilinear polygon. Simple geometric observations about rectilinear polygons are used to construct a simpler algorithm for computing all non-redundant components and consequently all weakly-visible chords in the rectilinear case. It may be noted that the chords need not be rectilinear.

Before we proceed we need some notation. Components in rectilinear polygons are produced by two kinds of ray shots – vertical and horizontal. We call these components *vertical* (resp. *horizontal*) components. Furthermore, there are two kinds of horizontal (vertical) components – components that lie *above* (to the em left of) or *below* (to the *right* of) the ray shot. We call the four type of components as *left-vertical*, *right-vertical*, *above-horizontal*, and *below-horizontal* components respectively. The left-vertical and right-vertical types are called *complementary* types, as are the above-horizontal and the below-horizontal types. It is easy to see that two components of the same type are either disjoint or one contains the other (i.e., one is made redundant by the other). For example, there cannot be two partially overlapping left-vertical components. If P has more than 3 disjoint components, then by lemma 1 P has no weakly-visible chords. Hence for weakly-visible chords to exist there cannot be more than 12 non-redundant components. In fact, we can make a stronger statement. It can be proved that for a weakly-visible rectilinear polygon, if there are two disjoint non-redundant components of a certain type, then there are no non-redundant of the complementary type and there cannot be another non-redundant component that intersects both of them. Consequently for a weakly visible rectilinear polygon, if there are two disjoint left-vertical non-redundant components, then there are no right-vertical non-redundant components. We state the following lemma without proof.

Lemma 10 *There are at most 2 vertical non-redundant components, and at most 2 horizontal components in a LR-visible rectilinear polygon. Also every non-redundant vertical (resp. horizontal) component is y-monotone (resp. x-monotone).*

Without getting into the details, we mention here that in 4 sweeps of the entire polygon, all the non-redundant components can be identified.

References

1. Avis, D., Toussaint, G.T.: An optimal algorithm for determining the visibility of a polygon from an edge. IEEE Transactions on Computers 30 (1981) 910–914
2. Chen, D.Z.: Optimally computing the shortest weakly-visible edge of a simple polygon. Proc. Fourth ISAAC, LNCS 762 (1993) 323-332
3. Doh, J., Chwa, K.: An algorithm for determining visibility of a simple polygon from an Internal Line Segment. J. of Algorithms 14(1) (1993) 139-168
4. Das, D., Heffernan, P.J., Narasimhan, G.: LR-visibility in polygons. Proc. 5th Canadian Conference on Computational Geometry (1993) 303-308. Submitted to special issue of Computational Geometry - Theory and Appln.
5. Das, G., Narasimhan, G.: Optimal Linear-Time Algorithm for the Shortest Illuminating Line Segment in a Polygon. Proc. 10th Annual ACM Symp. on Computational Geometry (1994)
6. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.: Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2 (1987) 209-233
7. Heffernan, P.J.: An optimal algorithm for the two-guard problem. Proc. 9th Annual ACM Symp. on Computational Geometry (1993) 348-358
8. Icking, C., Klein, R.: The two guards problem. Proc. 7th Annual ACM Symp. on Computational Geometry (1991) 166-175
9. Ke, Y.: Detecting the weak visibility of a simple polygon and related problems. Tech. Report, The Johns Hopkins University (1987)
10. Lee, D.T., Preparata, F.P.: An optimal algorithm for finding the kernel of a polygon. Journal of the ACM, 26(3) (1979) 415-421
11. Sack, J.-R., Suri, S.: An optimal algorithm for detecting weak visibility. IEEE Transactions on Computers 39(10) (1990) 1213-1219
12. Tseng, L.H., Lee, D.T.: Two-guard walkability of simple polygons. manuscript (1993)