Nordic Journal of Computing 1(1994), 433–457.

FINDING ALL WEAKLY-VISIBLE CHORDS OF A POLYGON IN LINEAR TIME

GAUTAM DAS[*] PAUL J. HEFFERNAN GIRI NARASIMHAN[†]

Mathematical Sciences Department
The University of Memphis
Memphis, TN 38152, U.S.A.

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. Previous algorithms for the problem run in $O(n \log n)$ time, and only compute a *single* weakly-visible chord, if one exists.

CR Classification: F.2.2

Key words: computational geometry, visibility, polygons, chords

1. Introduction

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 *chord* of a simple polygon is a line segment that connects two points on its boundary and lies entirely inside the polygon. A *weakly-visible chord* c of a polygon P is a chord such that c and P are weakly-visible. 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 [5, 11] require $O(n \log n)$ time and compute only one weakly-visible chord.

We state four versions of the weakly-visible chords problem:

(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.

In earlier papers [5, 11], version 3 has been solved in $O(n \log n)$ time. 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

[*] *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 Grants INT-911-5870 and CCR-940-9752

Received May 1994. Accepted November 1994.

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.

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. Weak-visibility of a polygon from an edge was first studied by Avis and Toussaint [1]; Sack and Suri [14] subsequently gave a linear-time algorithm that computes all weakly-visible edges of a simple polygon. Chen [4] gave a linear-time algorithm that finds the shortest weakly-visible edge, if one exists. 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. An $O(n \log n)$-time algorithm that computes all LR-visible pairs x and y is given by Tseng and Lee [15], and Das, Heffernan and Narasimhan [6] subsequently gave a linear-time algorithm that computes all LR-visible pairs s and t. The weakly-visible chords problem was studied by Doh and Chwa [5] and by Ke [11]. They developed algorithms that required $O(n \log n)$ time and computed only one weakly-visible chord.

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 [6] is used as a subprocedure.

What is interesting about the techniques used here and in [6] 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 [7] to solve to optimality the problem of finding the shortest weakly-visible segment (if one exists) in the interior of a simple polygon.

A related result is a linear-time algorithm due to Bhattacharya et al. [2] to compute a *single* (not necessarily shortest) weakly-visible line segment inside a given polygon. The advantage of their algorithm is that it does not use the linear-time triangulation algorithm of Chazelle [3] or the linear-time shortest path algorithm of Guibas et al. [8].

Another problem that 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) [9], and $O(n \log n)$-time algorithms which find all pairs x and y (version 4) for various formulations [15].

For this paper, 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. Since our algorithm uses many other algorithms as subroutines, where similar assumptions have been made, it is not clear whether this assumption can be eliminated from our algorithm.

2. Preliminaries

In this section we define some of the notation used in this paper. We also discuss some of the properties of shortest paths in polygons. 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, $int(P)$. A clockwise (resp. counterclockwise) traversal of P is a traversal along P such that $int(P)$ always lies immediately to the right (resp. left). 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.

We let $d(x, y)$ denote the direction of a ray or line from x through y, and $\vec{r}(x, \alpha)$ represent the ray rooted at x in direction α. Two rays with common endpoint x partition the plane into two regions, each of which is the union of a set of rays with endpoint x. A *cone* is defined as the region containing all rays encountered as we sweep counterclockwise from $\vec{r}(x, y_1)$ to $\vec{r}(x, y_2)$, and is denoted as $cone(d(x, y_1), d(x, y_2))$ (or $cone(y_1, x, y_2)$). We can also think of a cone as an interval of directions. We write $int(cone(y_1, x, y_2))$ to represent $cone(y_1, x, y_2) \setminus \{\vec{r}(x, y_1), \vec{r}(x, y_2)\}$, a cone minus its boundary directions. 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.

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, for a ray $\vec{r}(v, \alpha)$ rooted at v, where $\alpha \in int(cone(v^+, v, v^-))$, the *hit point* of this ray shot is the point of $(P \setminus \{v\}) \cap \vec{r}(v, \alpha)$ closest to v. We will sometimes denote a ray shot by writing its corresponding ray. Note that the ray shot $\vec{r}(v, \alpha)$ is defined only if $\alpha \in int(cone(v^+, v, v^-))$. A *reflex vertex* of P is defined as a vertex where the angle between the two edges incident on that vertex towards the interior of P is greater than 180 degrees. Each reflex vertex defines two special ray shots as follows. We let $\vec{r}_{CW}(v) = \vec{r}(v, d(v^+, v))$

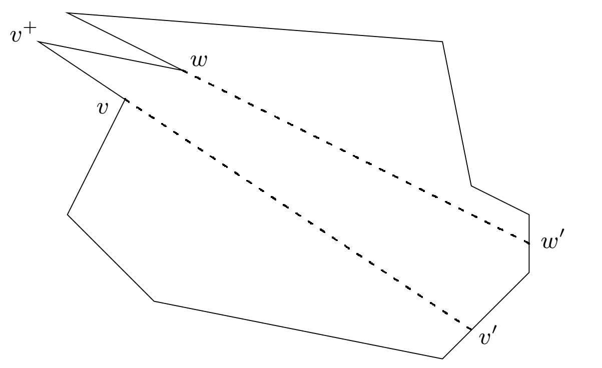

Fig. 1: Reflex vertex and redundant component

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. Counterclockwise ray shots and components are defined in the same way. A component is *redundant* if it is a *super*set of another component. Fig. 1 shows an example of two reflex vertices v and w, two clockwise components, namely, $P_{CW}(v, v')$ and $P_{CW}(w, w')$. Note that $P_{CW}(v, v')$ is a redundant component since it is a superset of the component $P_{CW}(w, w')$.

The *shortest path* between two vertices w and v of P, denoted $SP(w, v)$, is the (Euclidean) minimum-distance curve with endpoints w and v lying entirely in $P \cup int(P)$. Several properties of shortest paths were noted in [6] and [9], These properties are fairly intuitive. However, here we reproduce the ones that we need: Shortest paths are unique. Two shortest paths from a vertex to two distinct vertices cannot cross twice, since this would imply distinct shortest paths between a pair of points. The path $SP(w, v)$ is always a polygonal chain, whose interior vertices are all reflex vertices of P. This is because: if one of the above two conditions is violated, some small amount of local improvement is possible. The following lemma is reproduced from [6].

LEMMA 1. *If w and v are vertices of P, and $SP(w, v)$ is the shortest path directed from w to v, then any vertex of $SP(w, v) \setminus \{w, v\}$ that lies on $P_{CW}(w, v)$ is a left turn, while a vertex of $SP(w, v)$ on $P_{CCW}(w, v)$ is a right turn.*

We write $FE(w, v)$ to denote the *first edge* of $SP(w, v)$; that is, the edge of $SP(w, v)$ incident to w. The direction of this edge away from w is denoted $dFE(w, v)$. The following lemma is established in [9].

LEMMA 2. *Given points x and y of P, and a direction α contained in the interior of $cone(x^+, x, x^-)$, the ray shot $\vec{r}(x, \alpha)$ hits $P_{CCW}(x, y)$ if $\alpha \in int(cone(d(x, x^+), dFE(x, y)))$ and it hits $P_{CW}(x, y)$ if α is contained in the interior of $cone(dFE(x, y), d(x, x^-))$.*

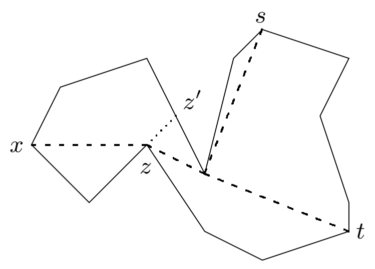

Fig. 2: Proof of lemma 3

3. LR-visibility

Since the LR-visibility algorithm is used as a subprocedure by our algorithm, we present an important lemma about LR-visibility and briefly describe the output of the LR-visibility algorithm in [6], We show in lemma 3 that the set of all components of P completely determines LR-visibility of P. Lemma 3 is implied by the results in [10]. However, we present a complete proof here since we use this lemma extensively throughout the paper.

Lemma 3. *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.*

Proof. If s and t both miss a clockwise component $P_{CW}(v, v')$, then the edge (v, v^+) is clearly not visible from any point on $P_{CW}(s, t)$, as can be seen from Fig. 1. Consequently, P is not LR-visible with respect to s and t.

To prove the converse, let s or t be contained in every non-redundant component. For the sake of contradiction, assume that there is a point $x \in P_{CCW}(s, t)$ which is not visible from $P_{CW}(s, t)$. Hence on a counterclockwise traversal of P the points are encountered in the order x, t, s. From the simple properties of shortest path, we know that on a counterclockwise sweep at x starting $d(x, x^-)$ to $d(x, x^+)$, $FE(x, t)$ will not be encountered after $FE(x, s)$, since otherwise the shortest path $SP(x, s)$ and $SP(x, t)$ would cross each other twice. If $FE(x, s) \neq FE(x, t)$ then consider a ray shot along a direction between $dFE(x, s)$ and $dFE(x, t)$ towards the interior of P. By lemma 2 this ray shot cannot hit $P_{CW}(x, s)$ or $P_{CCW}(x, t)$. Hence it must hit $P_{CW}(s, t)$. But this contradicts the assumption that x is not visible from $P_{CW}(s, t)$. So let $FE(x, s) = FE(x, t) = (x, z)$. Fig. 2 shows this situation with the two paths $SP(x, s)$ and $SP(x, t)$ shown using dashed lines. Without loss of generality assume that z is a point on $P_{CCW}(x, t)$, as shown in Fig. 2. Since z is also on $P_{CCW}(x, s)$, by lemma 1, $SP(x, s)$ must have a right turn at z. Now consider the clockwise component at z, namely $C = P_{CW}(z, z')$. Fig. 2 shows the ray shot from z using a dotted line. The

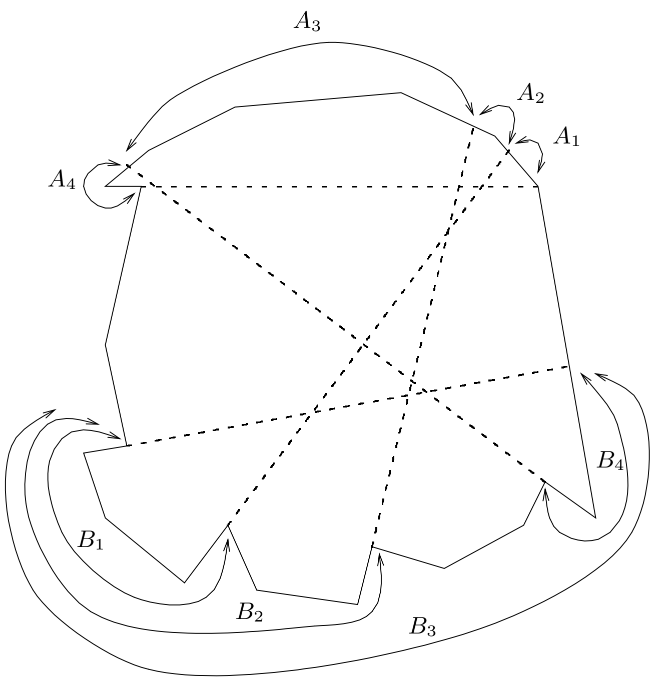

Fig. 3: A LR-visible polygon

hit point z' of the counterclockwise ray shot from z cannot be on $P_{CCW}(z, s)$ since otherwise $SP(x, s)$ would have a left turn instead of a right turn at z. Hence z' must be on $P_{CW}(z, s)$ and thus C cannot include s. Also, since z is on $P_{CCW}(x, t)$, and since C does not include s, it cannot include t either. Thus C misses both s and t. If C is not a non-redundant component, then there must exist a non-redundant component that is completely contained in C and hence misses both s and t. In any case, we show the existence of a non-redundant component that misses both s and t, which contradicts our assumption. Hence the proof. □

One consequence of the above lemma is that if a polygon has more than two disjoint components, then it is not LR-visible, i.e., it has no LR-visible pairs of points. The LR-visibility algorithm in [6] 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. For the example shown in Fig. 3, the algorithm outputs the pairs of subchains $(A_1, B_1), \ldots, (A_4, B_4)$. It is also easy to verify that if (s, t) is a pair of points that do not both lie in one of the four pairs of subchains, then both s and t will miss at least one component. 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). It is possible for a degenerate basic interval consisting of a single point to exist. Thus a basic interval may or may not contain either of its endpoints. In any case, for any interval F, let $b(F)$ $(e(F))$ denote the starting point (ending point) of an interval F encountered in the counterclockwise direction. By lemma 3, 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 all points y such that (x, y) is a LR-visible pair for all $x \in A_i$. The following two lemmas are proved in [6].

Lemma 4. *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 5. *If $A_i \cap B_i \neq \emptyset$, then $B_i = P$.*

In [6], 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 weakly-visible chords algorithm. Additional notation is introduced as and when required.

4. More geometric properties

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

Lemma 6. *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.*

Proof. The proof follows easily from the definitions involved. Assume $\overline{xy}$ is a weakly-visible chord of P. Let $A = P_{CW}(x, y)$ and $B = P_{CCW}(x, y)$. Consider any point z on A. There must be a point z' on chord $\overline{xy}$ from which z is visible. The line $\overline{zz'}$ when extended must hit a point on B. Hence every point on A (B) is visible from some point on B (A). Hence P is LR-visible for x and y.

Now if $\overline{xy}$ is a chord of P and P is LR-visible for x and y, then any point z on $A = P_{CW}(x, y)$ must be visible from some point w on $B = P_{CCW}(x, y)$. Hence $\overline{zw}$ is a chord of P that must intersect chord $\overline{xy}$, implying that z must be visible from some point on chord $\overline{xy}$. A similar argument proves that any point on B must be visible from some point on the chord $\overline{xy}$. $\square$

The lemma suggests the following skeleton for the algorithm: first compute LR-visibility using the algorithm in [6], then compute all chords $\overline{xy}$ such that $x \in A_i$ and $y \in B_i$, i.e., find all pairs of LR-visible pairs of points that are visible to each other. While lemma 4 shows that B_i is connected, the crucial point that we use is that for every $x \in A_i$, the set of points on B_i that are visible to x form a subinterval of B_i, i.e., form a connected set in B_i. The rest of the section focuses on proving this critical point.

We first 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 defined as the intersection of the half-planes formed by extending the edges of the polygon (see [13] for details). The intersection of one of these half-planes with the polygon P is exactly a component of P. The kernel is thus a convex set and 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 of chords between the pair of intervals $(K \cap P, P)$.

LEMMA 7. *For some 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 $A_i \cap B_i \neq \emptyset$, by lemma 5 we have $B_i = P$. $\square$

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 we define 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 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.

We now introduce a concept called *well-behavedness* that is very crucial for understanding the correctness and efficiency of our algorithm. 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 by lemma 1 $SP(w, v)$ contains no left turns. This is a stronger statement than simply saying that $SP(w, v)$ is a convex chain, since it specifies the direction of all the turns. The following lemmas prove that the A_is, B_is, D_is, and E_is are all well-behaved; this fact will be useful in their efficient computation, as shown later.

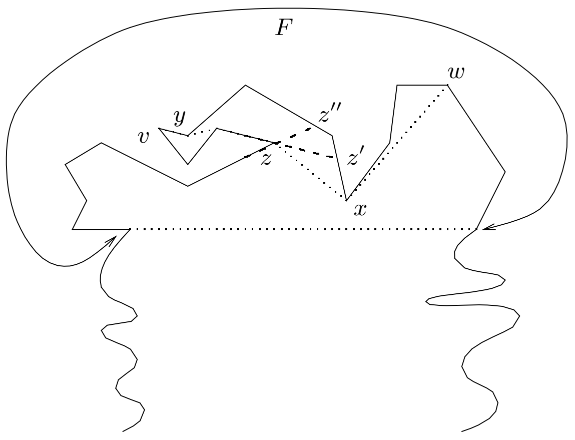

Fig. 4: Proof of lemma 8

LEMMA 8. *Any subchain of a non-redundant component is well-behaved.*

PROOF. Consider the subchain $P_{CCW}(w, v)$ of a non-redundant component F. If it is not well behaved, then $SP(w, v)$ must contain a point z of $P_{CW}(w, v) \setminus \{w, v\}$. Let x (resp. y) be the last (resp. first) point of $P_{CCW}(w, v)$ preceding (resp. succeeding) z on $SP(w, v)$ (see Fig. 4). 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 z' and z'' lie on F, one of the two components generated by z ($P_{CW}(z, z')$ and $P_{CCW}(z, z'')$) is strictly contained in F, a contradiction of the fact that F is non-redundant. □

The following is a simple consequence of the above lemma.

LEMMA 9. *Any non-kernel basic interval A_i as well as its corresponding B_i is well-behaved.*

PROOF. Clearly, by the definition each non-kernel basic interval A_i is contained in some non-redundant component. Hence by lemma 8, A_i is well-behaved. Also, every non-kernel A_i must miss some non-redundant component, say F. By lemma 3, the corresponding B_i must be included in F. Again by lemma 8 B_i is also well-behaved. □

We next show that even the side intervals D_i and E_i are well-behaved. We first consider the case where P does not have two disjoint non-redundant components. It may be noted that even if P does not have two disjoint non-redundant components, it does not guarantee that P has a kernel. This is because it is possible to have three non-redundant components that cover the entire polygon and that intersect each other. This justifies the need for the following lemma.

Lemma 10. *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. Every component corresponds to a ray shot and a corresponding chord. We first prove that if P does not have two disjoint components, then the chord corresponding to any non-redundant component 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 contained in F, which is not possible since F is non-redundant. Therefore the chord that forms F has endpoints intersecting every component of P, and thus by lemmas 3 and 6 is a weakly-visible chord of P.

Thus there are (at least two) weakly-visible chords connecting one of the endpoints of each non-kernel A_i and its corresponding B_i. Any one of these weakly-visible chords 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 8, 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 $b(A_i)$, the first point of A_i in counterclockwise order. This means that there exists a pair of points s and t such that neither of them are contained in H. By lemma 3, this contradicts the assumption 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 is somewhat less general.

Lemma 11. *Let P be a polygon with at least two disjoint components. If there exists a weakly-visible chord in P then each subchain of D_i and E_i is well-behaved.*

Proof. This proof is more involved than that of lemma 10. We first show that if a subchain of D_i (resp. E_i) is not well-behaved, then it cannot be due to obstruction by some portion of D_i (resp. E_i). Next we show that if a subchain of D_i (resp. E_i) is not well-behaved, then it cannot be because of obstruction by A_i or B_i. Finally we show that if a subchain of D_i (resp. E_i) is not well-behaved due to obstruction by E_i (resp. D_i), then the LR-visible pairs of points are not visible from each other, and consequently no weakly-visible chords exist.

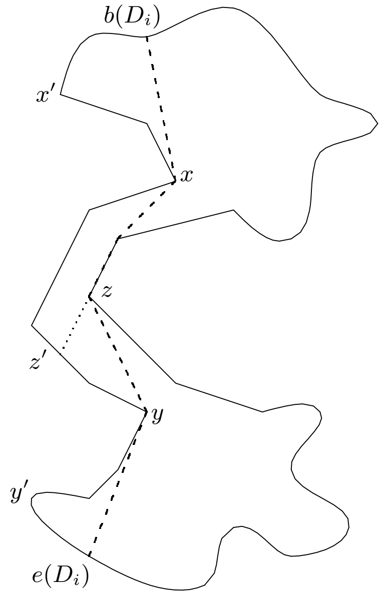

Fig. 5: Proof of lemma 11

Using arguments similar to the ones used in lemmas 9 and 10, it is easy to prove that if a subchain of D_i (resp. E_i) is not well-behaved then it cannot be because of D_i (resp. E_i), since this would produce a component wholly contained inside D_i (resp. E_i).

Next assume that a subchain of D_i is not well-behaved because of A_i. This happens when $SP(x', y')$ (for some x' and y' on D_i) contains a reflex vertex z of A_i (see Fig. 5). If z is a reflex vertex of $A_i \setminus \{b(A_i)\}$ then let z' be the hitpoint of the counterclockwise ray shot from z. The component $P_{CCW}(z, z')$ is not touched by the left endpoint of A_i, namely $b(A_i)$. It also does not touch any point of B_i. Thus the LR-visible pair of points $(b(A_i), b(B_i))$ misses the component $P_{CCW}(z, z')$, which is a contradiction by lemma 3. One other case to consider is if $z = b(A_i)$. Let x (resp. y) be the first point of D_i preceding (resp. succeeding) z on $SP(x', y')$, as shown in Fig. 5. Consider the clockwise component from x. The clockwise ray shot from x must lie within $P_{CW}(x, z)$ and hence the corresponding component does not touch $b(A_i)$ or B_i, contradicting the fact that $(b(A_i), b(B_i))$ is an LR-visible pair of points (by lemma 3). Using similar arguments it can be proved that if either D_i or E_i are not well-behaved it cannot be because of either A_i or B_i.

Now we assume that a subchian of D_i is not well-behaved because of E_i, i.e., $SP(x', y')$ contains a reflex vertex z of E_i. Note that by an argument similar to the one in the previous paragraph, it can be shown that z is neither $b(E_i)$ nor $e(E_i)$. Since $SP(x, y)$ passes through z, x and y cannot be visible from each other. This is a special case since we now have two disjoint components, namely the clockwise component at x (call it C_x) and the counterclockwise component at y (call it C_y), that are completely hidden from each other. By lemma 3, the pair of intervals (A_i, B_i) cannot miss any components. Hence every basic interval A_i must lie in one of the two components C_x or C_y, while the corresponding interval B_i must lie in the other component. This implies that none of the LR-visible pairs of intervals (A_i, B_i) are visible to each other and consequently P has no weakly-visible chords. Hence the proof. □

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 D_i and E_i. 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.

Lemma 12. *Let x be a point on A_i and y a point on B_i. If the ray shot from x along $\overline{xy}$ intersects the polygon at a point $w \in P_{CCW}(x, e(A_i)) \cup P_{CCW}(b(B_i), y)$, then there must be at least one point of D_i to the left of the ray. Similarly, if the ray shot from x along $\overline{xy}$ intersects the polygon at a point $w \in P_{CCW}(y, e(B_i)) \cup P_{CCW}(b(A_i), x)$, then there must be at least one point of E_i to the right of the ray.*

Proof. Assume that a portion of A_i obstructs the ray from x towards y. Consider the point z on $P_{CCW}(x, b(B_i))$ that lies to the left of the ray along $\overline{xy}$ and such that the angle subtended by the segment $\overline{xz}$ with the segment $\overline{xy}$ is the greatest (see Fig. 6). Since z is an extreme point, it must be a reflex vertex. Let z' be the hit point of the clockwise ray shot from z. Let z be a point on A_i and $z \neq e(A_i)$. The component $P_{CW}(z, z')$ does not touch $e(A_i)$ or $b(B_i)$, contradicting the LR-visibility of the pair of points $(e(A_i), b(B_i))$. Hence $z = e(A_i)$ or z must be on D_i. In either case there is a point of D_i to the left of the ray from x towards y.

Similar arguments prove that (1) if a portion of $P_{CCW}(b(B_i), y)$ obstructs the ray shot from x towards y, then it must have at least one point of D_i to its left, and that (2) if a portion of $P_{CCW}(y, e(B_i)) \cup P_{CCW}(b(A_i), x)$ obstructs the ray shot from x towards y, then it must have at least one point of E_i to its right. □

The consequence of the above lemma is that if a ray shot from $x \in A_i$ towards $y \in B_i$ has all of D_i to its right and all of E_i to its left, then it cannot be obstructed by any other point of A_i or B_i. Also, if a ray shot from x along $\overline{xy}$ has D_i completely to its right, then this ray does not intersect $P_{CCW}(y, e(B_i)) \cup P_{CCW}(b(A_i), x) \setminus \{x, y\}$.

5. Overview of the algorithm

We first give an overview of the algorithm. There are several preliminary steps. Our algorithm first constructs the kernel K using the linear time

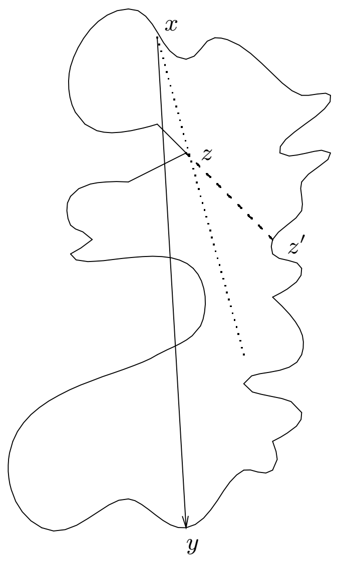

Fig. 6: Proof of lemma 12

method of [12] and then constructs $K \cap P$. This latter step is easily accomplished since the algorithm of [12] 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.

We then run the linear-time algorithm of Bhattacharya and Mukhopadhyay [2] to compute a single weakly-visible line segment. Clearly, P has a weakly-visible line segment if and only if P has a weakly-visible chord. Thus if P does not have a single weakly-visible segment then the algorithm can stop and report that there are no weakly-visible chords. Henceforth we will assume P has at least one weakly-visible chord, and consequently (by lemmas 10 and 11) that the side intervals are well-behaved.

The next step is to run the LR-visibility algorithm from [6], which gives us the non-redundant components with endpoints in counterclockwise order, as well as the LR-visible pairs of intervals (A_i, B_i), $i = 1, \ldots, k$. We next determine whether there exist two disjoint components. This can be done in linear time by using the scheme described in [15] or in [6]. 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.

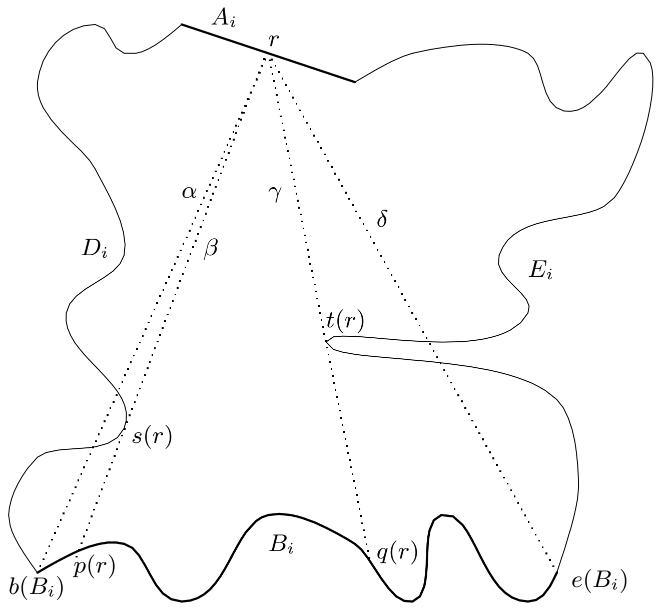

Fig. 7: Pseudo-tangents, pseudo-tangent points, and their extensions

In either case, the algorithm goes through k iterations, where $k = O(n)$ is the number of pairs of intervals output by the LR-visibility algorithm. The i-th iteration consists of determining the weakly-visible chords between a non-kernel A_i and its corresponding B_i. Since a point $r \in A_i$ forms an LR-visible pair with every point $p \in B_i$, by lemma 6 we can construct the set of weakly-visible chords by determining the points of B_i that are visible from r. By lemma 12, for each $r \in A_i$ its visibility from points on B_i is restricted only by the side intervals D_i and E_i.

The visibility between a point $r \in A_i$ and points on the chain B_i is determined by a pair of constructs we call the *pseudo-tangents*. Consider Fig. 7. The pseudo-tangent from r to D_i (resp. E_i) is the unique line directed along a ray shot from r towards a point $s(r)$ of D_i (resp. $t(r)$ of E_i) such that all of D_i (resp. E_i) lies on or to the right (resp. left) of the ray shot. The points $s(r)$ and $t(r)$ are called the *pseudo-tangent points*. Pseudo-tangents are different from *tangents*, since tangents are also required to be chords. For example, the pseudo-tangent from r to D_i may intersect E_i and hence may not be a chord. We denote the direction of the pseudo-tangent to D_i (resp. E_i) by $\beta(r)$ (resp. $\gamma(r)$). We give labels to two other special directions: the direction from r to $b(B_i)$ (resp. $e(B_i)$) is denoted $\alpha(r)$ (resp. $\delta(r)$). Let $p(r)$ (resp. $q(r)$) be the other endpoint of the longest chord of P with one endpoint at r in direction $\beta(r)$ (resp. $\gamma(r)$). We call $p(r)$ and $q(r)$

the *extension points* of the two pseudo-tangents. If direction β is clockwise of γ with respect to $r \in A_i$, then by lemma 12, all points on B_i between $p(r)$ and $q(r)$ are visible from r and hence form weakly-visible chords with r. Let $SP(D_i)$ denote the shortest path between the endpoints of D_i. Since D_i is well-behaved, it is easy to see that the pseudo-tangents to D_i are the same as the pseudo-tangents to $SP(D_i)$. In the following discussion, sometimes we will use abbreviated notation, for example α instead of $\alpha(r)$, when the r under consideration is clear.

Hence the strategy used by the algorithm is to traverse P with a point r. The i-th iteration is started off by computing the subchains $SP(D_i)$, and $SP(E_i)$ and by computing the pseudo-tangents from $b(A_i)$ to D_i and E_i. Then, as r traverses along A_i, for each $r \in A_i$, it computes the pseudo-tangents to D_i and E_i, thus computing the values of β and γ. If r is visible from some point on B_i, it computes the extensions of the pseudo-tangents p and q as well as the points of tangency s and t. Then the algorithm reports all chords between r and points on B_i between p and q.

Below we give the details of the algorithm. We also show that as r traverses along P, the values of $\alpha(r)$, $\beta(r)$, $\gamma(r)$, $\delta(r)$, $s(r)$, $t(r)$, $p(r)$, and $q(r)$ can be efficiently maintained and updated (they all satisfy some monotonicity properties), thus obtaining a linear-time algorithm.

6. Computing all weakly-visible chords

Consider the i-th iteration of the algorithm, which involves the traversal with r along A_i. The preprocessing at the start of the i-th iteration involves computing the subchains $SP(D_i)$ (as described in section 6.1) and $SP(E_i)$, and computing the pseudo-tangents from $b(A_i)$ to D_i and E_i. As r traverses along A_i, the pseudo-tangents are computed for each r (as described in section 6.2).

For a fixed point r, the directions α and δ are easily computed. The pseudo-tangents from r effectively give the directions β and γ. The point r 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 towards the interior of P, as viewed from r"). Consider the scenario for a point $r \in A_i$ which is visible from some point on B_i. In this case the pseudo-tangencies are actually tangencies, in the sense that $\overline{rp}$ and $\overline{rq}$ are chords of P. Since α, β, γ and δ are ordered counterclockwise, the points p and q lie on B_i. Points of B_i lying between $b(B_i)$ and p (resp. q and $e(B_i)$) are not visible from r because visibility is blocked by D_i (resp. E_i). However, by lemma 12 all points of $P_{CCW}(p,q)$ are visible from r. Also, if $b(B_i)$ (resp. $e(B_i)$) is the pseudo-tangent point of D_i (resp. E_i), i.e. if $\alpha = \beta$ (resp. $\gamma = \delta$), then $b(B_i)$ (resp. $e(B_i)$) is also visible from r. Thus if r is visible from some point on B_i, by lemma 6 and 12, its set of weakly-visible chord partners consists of the closed subchain $P_{CCW}(p,q)$ of B_i.

If r is not visible from any point on B_i, then the four special directions are not ordered properly. If D_i (resp. E_i) blocks all of B_i from r, then we have the subordering $\alpha \leq_{ccw} \delta <_{ccw} \beta$ (resp. $\gamma <_{ccw} \alpha \leq_{ccw} \delta$). 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 p and q 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 q precedes p counterclockwise on B_i.

In order to find the pseudo-tangencies from every point r, we traverse P once clockwise with a point r, calculating for each r on A_i the pseudo-tangent to the side interval E_i. We then perform the counterclockwise traversal of P and compute for each r the pseudo-tangent to D_i (as described in section 6.2). During this traversal, we also determine for which points r the ordering $\alpha \leq_{ccw} \beta \leq_{ccw} \gamma \leq_{ccw} \delta$ is obeyed. For these points r we compute the extensions p and q to obtain the partner interval $P_{CCW}(p, q)$. The computation of the extensions of the pseudo-tangents is described in section 6.2. Note that if $\gamma <_{ccw} \beta$ then r sees no point of B_i and the extension p is undefined. Also if $\alpha = \beta$ (resp. $\gamma = \delta$) for any point r, then $b(B_i)$ (resp. $e(B_i)$) is also a partner. Since p and q are different for each r, and there are an infinitude of values of r, we must exhibit care in our manner of computing and storing the output; this issue will be addressed during the discussion below.

We now show how $SP(D_i)$ can be efficiently computed and stored in a structure we will call as the *side shortest path tree* (or simply the SSPT structure).

6.1 Computing $SP(D_i)$

The computation of the $SP(D_i)$s are handled in *groups*. Consider the first side interval D_1, which corresponds to the first basic interval A_1. Assume that it contains the collection of basic intervals $A_2, \cdots, A_j$, as shown in Fig. 8. Assume that the side interval D_j contains 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-1} as the first *group*. Similarly, D_j through D_{l-1} will be dealt with as the second *group*, and so on. In the next three paragraphs, we will describe how to preprocess an entire group in a manner that allows efficient updating. Note that the first and the last group may overlap. However, no basic interval is in more than two groups. Hence this overlap only introduces a constant multiplicative factor in the time complexity. Note also that if P has two disjoint components, then all the basic intervals $A_2, \ldots, A_k$ are contained in D_1, and only one group of side intervals is formed. If P does not have two disjoint components, then many groups of side intervals may exist. Hence the procedure described below for one group is repeated once for each group.

For what follows, we assume that a *group* consists of side intervals D_1 through D_{j-1}. The preprocessing for the group involves the construction

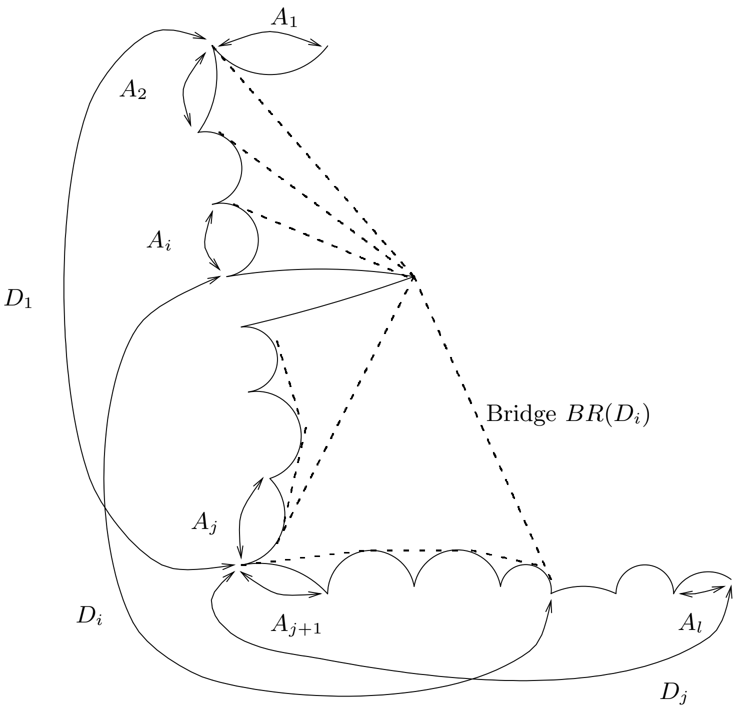

Fig. 8: Processing the side intervals

of the shortest path tree from $e(D_1)$ as well as from $b(D_1)$ to all vertices of D_1. We also construct shortest path trees from $e(D_j)$ and $b(D_j)$ to all vertices of D_j. Since D_1 and D_j are well-behaved we perform this step in time proportional to the size of D_1 and D_j, by a modification of the algorithm of [8]. We refer to a common tangent between two convex chains as the *bridge* between them. At any time $SP(D_i)$ is stored in three fragments, 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 denoted by $BR(D_i)$. Thus, as r 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 $SP(e(D_1), e(D_i))$. The fragment $SP(b(D_i), e(D_1))$ is obtained from $SP(b(D_{i-1}), e(D_1))$ by a depth-first search of the shortest path tree from $e(D_1)$, which allows one to visit all vertices of D_1 according to the counterclockwise order on P. The fragment $SP(e(D_1), e(D_i))$ is obtained from $SP(e(D_1), e(D_{i-1}))$ by performing a depth-first search of the shortest path tree from $b(D_j)$, which allows one to visit all vertices of D_j according to the counterclockwise order on P. Note that $SP(e(D_1), e(D_i))$ is "ballooning out" while $SP(b(D_i), e(D_1))$ is "shrinking". More formally, the region of $int(P)$ enclosed by $SP(e(D_1), e(D_i))$ contains the region of $int(P)$ enclosed by $SP(e(D_1), e(D_{i-1})$ and the region of $int(P)$ enclosed by $SP(b(D_i), e(D_1))$

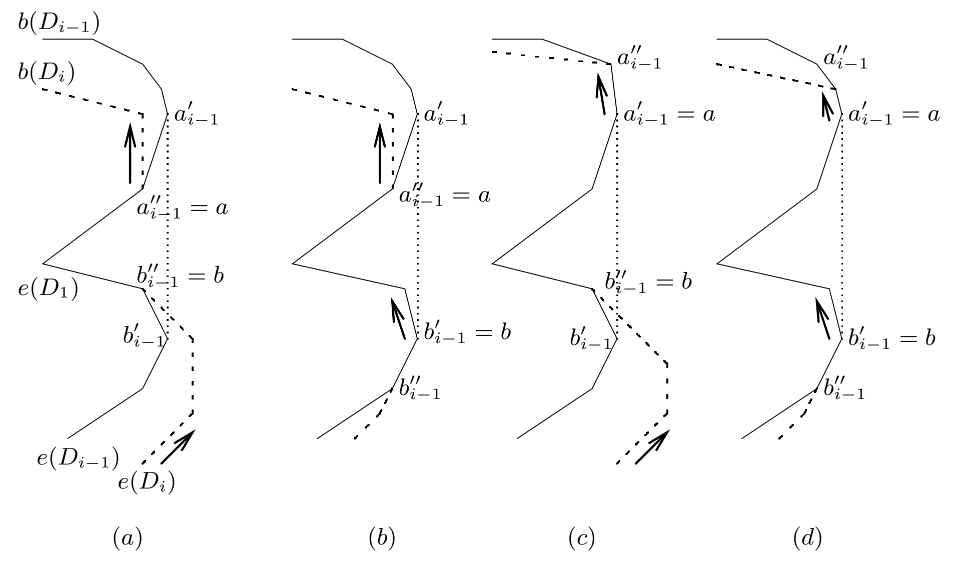

Fig. 9: Processing the side intervals

is contained in the region of $int(P)$ enclosed by $SP(b(D_{i-1}), e(D_1)$. We point out that because of the above observations, the chains $SP(D_{i-1})$ and $SP(D_i)$ must intersect.

Computing the bridge between two convex chains that share a common point can be done in one of many ways. One way to do it is as follows: traverse the first chain starting from the common point moving towards its extreme end, while traversing the second chain from its extreme end moving towards the common point. For each vertex x traversed on the second chain, compute the point of tangency for tangents to the first chain passing through x. Keep moving to the next vertex on the second chain until the pair of vertices representing the points of common tangency (i.e. the bridge between the chains) between the chains is found. To check whether a line joining two points is the required bridge (in constant time) compare the slopes of the incident edges from the two chains with that of the tangent line.

The above method for finding the bridge is used by our algorithm for computing $BR(D_1)$, the bridge for the first side interval. However, we use a slightly modified method for computing the other bridges $BR(D_i)$, $i > 1$, since these can be computed from $BR(D_{i-1})$ as described below.

Let the bridge $BR(D_{i-1})$ connect point a'_{i-1} on $SP(b(D_{i-1}), e(D_1))$ and point b'_{i-1} on $SP(e(D_1), e(D_{i-1}))$. Denote by a''_{i-1} the point where the paths $SP(b(D_i), e(D_1))$ and $SP(b(D_{i-1}), e(D_1))$ begin to separate, and by b''_{i-1} the point where $SP(e(D_1), e(D_i))$ and $SP(e(D_1), e(D_{i-1}))$ begin to separate. Let a be the point that is *closer* to $e(D_1)$ among the two points a'_{i-1} and a''_{i-1}. Note that when we say that point x is closer to $e(D_1)$ than y we mean that point x lies on the shortest path from y to $e(D_1)$. Similarly, let b be the point that is closer to $e(D_1)$ among the two points b'_{i-1} and b''_{i-1}. Let the bridge $BR(D_i)$ connect the points a'_i and b'_i. Due to the shrinking of $SP(b(D_i), e(D_1))$, a'_i must lie on $SP(b(D_i), a)$. Similarly, due to the ballooning out of $SP(e(D_1), e(D_i)$, b'_i must lie on $SP(e(D_1), b'_{i-1})$ if $b = b'_{i-1}$, otherwise b'_i must lie on $SP(e(D_1), e(D_i))$. Examples of the four cases generated if $a = a'_{i-1}$ or $a = a''_{i-1}$ and if $b = b'_{i-1}$ or $b = b''_{i-1}$ are shown in Figs. 9(a)-(d). These figures show the two fragments of $SP(D_{i-1})$ as a polygonal chain, the two fragments of $SP(D_i)\backslash SP(D_{i-1})$ as a dashed chain, and the bridge $BR(D_{i-1})$ as a dotted line. A thick arrow in the figure shows the direction of traversals for finding $BR(D_i)$.

To find the bridge $BR(D_i)$, our algorithm traverses along $SP(b(D_i), e(D_1))$ from a (instead of starting the search from $e(D_1)$) in a direction away from $e(D_1)$. If $b = b'_{i-1}$, then the algorithm traverses along $SP(e(D_1), e(D_i))$ from b in a direction towards $e(D_1)$. If, instead, $b = b''_{i-1}$, then the algorithm traverses along $SP(e(D_1), e(D_i))$ from $e(D_i)$ in a direction towards $e(D_1)$. At any point along these traversals the algorithm computes the point of tangency from the point on $SP(e(D_1), e(D_i))$ to the convex chain $SP(b(D_i), e(D_1))$. Once the point is found, the algorithm moves to the next vertex along $SP(e(D_1), e(D_i))$ and finds the point of tangency for this new vertex. This is continued until the bridge is found. Note that this is a slight modification of the scheme presented earlier for finding $BR(D_1)$. The main difference is that additional information about the location of the points of tangency a'_i and b'_i are used, thereby avoiding repeated traversals of portions of the shortest path tree.

The data structure to store all the $SP(D_i)$s is called the side shortest path tree (SSPT) structure. It has one structure for each group of side intervals. As mentioned earlier, for a group of side intervals, say $D_1, \ldots, D_j$, each $SP(D_i), 1 \leq i \leq j$, is stored in three fragments, namely, $SP(b(D_i), e(D_1)), SP(e(D_1), e(D_i))$ and $BR(D_i)$. The collection of the first fragments $(SP(b(D_i), e(D_1)))$ of each $SP(D_i), 1 \leq i \leq j$ is stored in a tree, while the collection of second fragments $(SP(e(D_1), e(D_i)))$ is stored in another tree. In order to store $BR(D_i)$, pointers are stored in both the trees to indicate its endpoints. By the construction described above, it should be clear that if an edge of one of the trees is on $SP(D_l)$ and on $SP(D_u)$, then it is also on $SP(D_{l+1}), \ldots, SP(D_{u-1})$. Each edge e of the SSPT structure has information attached to it of the form $[l_e, u_e]$ to indicate that l_e (resp. u_e) is the smallest (resp. largest) index such that e is on $SP(D_{l_e})$ (resp. $SP(D_{u_e})$). Note that the values l_e and u_e are initialized during iterations l_e and $u_e + 1$ respectively. In iteration l_e a subchain that includes e will be added to a fragment of $SP(D_{l_e})$, while in iteration $u_e + 1$ a subchain that includes e will be removed from $SP(D_{u_e+1})$. For convenience, we will assume that the SSPT structure is a separate data structure built to store all the $SP(D_i)$s, although it is possible to incorporate this data structure into the shortest path trees that are built from $b(D_1)$ and $e(D_1)$.

We now discuss the complexity of computing the SSPT structure. The first step involved is computing the shortest path trees from the starting and ending points of each group of side intervals. Computing these trees is done once for each group of side intervals and can be done in linear time by the algorithm of [8], since each of them is well-behaved. The size of the shortest path tree for a group is at most a constant times the number of vertices in that group and thus the union of these trees has $O(n)$ vertices. It is easy to see that the overlap between the first and the last group will not affect the size by more than a factor of 2. Hence computing shortest path trees takes time $O(n)$. Computing $SP(b(D_i), e(D_1))$ and $SP(e(D_1), e(D_i))$ involves performing a depth first search of these shortest path trees, and the time spent on this computation for an entire group is linear in the size of the tree for that group. Overall, this work amortizes to $O(n)$ time. Efficient computation and maintenance of the bridge is possible because the bridge endpoints display a monotonicity property similar to that of the pseudo-tangents from r: the bridge endpoints progress monotonically clockwise along the shortest paths. Because of the starting points (points a and b from above) as well as the clockwise directions of the traversals in each iteration it is clear that no edge of the shortest path trees is traversed more than once for the purpose of finding a bridge. Thus computing all the bridges can also be done in $O(n)$ time. Hence the preprocessing to compute the SSPT structure that stores all the $SP(D_i)$s can be performed in linear time.

6.2 Computing pseudo-tangents and their extension points

We now describe the computation of the pseudo-tangents to D_i as r traverses counterclockwise along A_i; the procedure for the pseudo-tangent to E_i is similar. The well-behavedness of D_i makes it easy to find and update pseudo-tangencies. Also 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 r with those of the adjacent edges.

The following lemma is crucial to understanding the correctness as well as the complexity of computing these pseudo-tangents.

LEMMA 13. *As r is traversing counterclockwise within a basic interval A_i, the point of pseudo-tangency from r on D_i, namely $s(r)$, moves clockwise along $SP(D_i)$ towards $b(D_i)$.*

PROOF. Since D_i is well-behaved, the shortest path $SP(D_i)$ is a convex chain consisting of only right turns (by lemma 1) and the point of pseudo-tangency must lie on $SP(D_i)$. If A_i were convex then clearly as r moves counterclockwise along A_i, the point of pseudo-tangency moves clockwise along $SP(D_i)$. A_i may not be convex; however, it is well-behaved. For the sake of contradiction assume that for some r on A_i, a counterclockwise

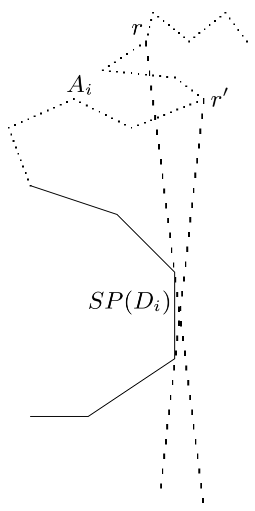

Fig. 10: Proof of lemma 13

traversal from r causes the point of pseudo-tangency to move counterclockwise along $SP(D_i)$. As can be seen easily from Fig. 10, this will cause some portion of A_i to obstruct visibility of B_i without any portion of D_i causing the obstruction. This contradicts lemma 12 and hence the proof. □

This also results in the monotonicity of many other values as r traverses along A_i. We first show that the functions $\beta(r)$ and $p(r)$ are monotonic, i.e., as r moves counterclockwise along the entire polygon, the direction β moves counterclockwise with respect to r and the point p moves counterclockwise along P. We first consider the traversal of r within a basic interval A_i. If the pseudo-tangent point remains the same, then as r moves counterclockwise the pseudo-tangent rotates counterclockwise around the pseudo-tangent point, with the consequence that p (if defined) moves counterclockwise on B_i. If the pseudo-tangent point changes (there is an instant when the point of pseudo-tangency lies on an edge of $SP(D_i)$), it moves in a clockwise direction along $SP(D_i)$ towards $b(D_i)$. But then $\beta(r)$ and $p(r)$ moves monotonically counterclockwise. Note also that the points r, $s(r)$ and $p(r)$ are collinear. So once r and $s(r)$ are computed, computing $p(r)$ involves a simple counterclockwise traversal of B_i.

Next we describe the computation (performed at the start of iteration i) of the pseudo-tangents from $b(A_i)$ to D_i. When r reaches $e(A_{i-1}) = b(A_i)$, iteration i is started. Then $SP(D_i)$ is computed from $SP(D_{i-1})$ as described in section 6.1. The point of pseudo-tangency $s(b(A_i))$ lies on $SP(D_i)$. If $\alpha(b(A_i)) \leq_{ccw} \beta(e(A_{i-1}))$, then the pseudo-tangent point does not change (see Fig. 11(a)), and in fact, $s(e(A_{i-1}))$ lies on $SP(D_{i-1}) \cap SP(D_i)$. Con-

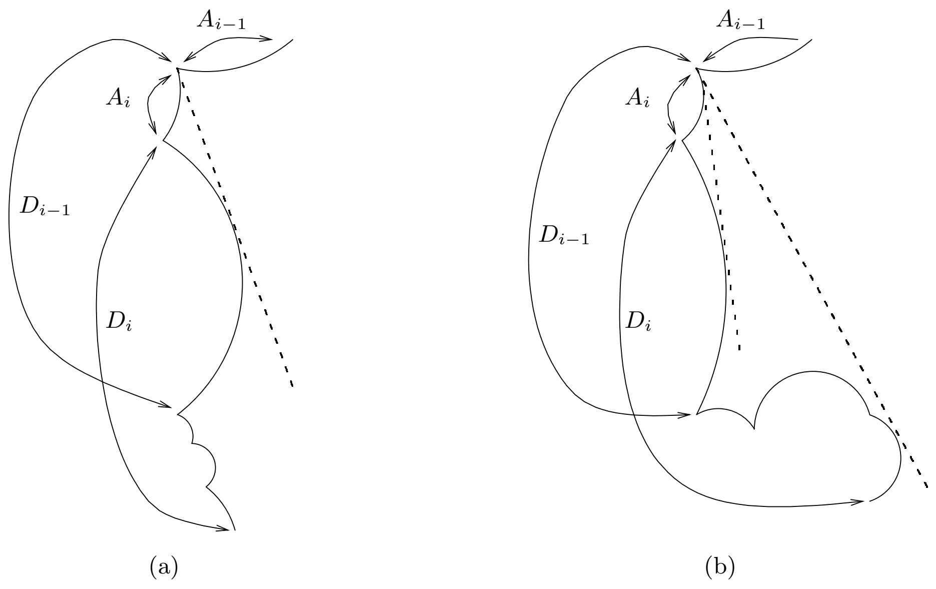

Fig. 11: Computing pseudo-tangents from $b(A_i)$ to D_i

sequently, $s(e(A_{i-1})) = s(b(A_i))$ and $p(e(A_{i-1})) = p(b(A_i))$. Now assume that $\beta(e(A_{i-1})) \leq_{ccw} \alpha(b(A_i))$ (see Fig. 11(b)). Since $SP(D_i)$ is a convex chain, although $s(e(A_{i-1}))$ may lie on it, it does not touch the rest of the subchain of $SP(D_{i-1})$ from $s(e(A_{i-1}))$ to $e(D_{i-1})$ (otherwise $SP(D_{i-1})$ will lie entirely to the left of the direction $\beta(e(A_{i-1})))$. In this case the algorithm computes the new point of pseudo-tangency by traversing $SP(D_i)$ starting from $e(D_i)$ until $s(b(A_i))$ is found. Note that the subchain of $SP(D_{i-1})$ from $s(e(A_{i-1}))$ to $b(D_{i-1})$ has not been traversed for the purpose of finding pseudo-tangents. Hence any portion of $SP(D_{i-1}) \cap SP(D_i)$ on this subchain of $SP(D_{i-1})$ has also not been traversed for this purpose (although it may be traversed in a later iteration). In this case the pseudo-tangent from r to D_i is counterclockwise to the pseudo-tangent from r to D_{i-1}. Hence $p(b(A_i))$ is counterclockwise of $p(e(A_{i-1}))$ and can be found by a counterclockwise traversal of B_i starting from $b(B_i)$. Traversing along $SP(D_i)$ simply involves traversing along the SSPT structure described in section 6.1. As argued above, the portion of the SSPT structure traversed during the i-th iteration for computing pseudo-tangents consists only of those edges of the SSPT structure that have not been traversed in earlier iterations and by previous arguments this must take $O(n)$ time when summed over all iterations.

Thus it is clear that pseudo-tangents points, pseudo-tangents and their extension points for all r can be computed in linear time.

6.3 Representing and storing the output

Section 6.2 details how to find $p(r)$ and $q(r)$ for any given r, and also how to maintain it as r traverses along P. For any given r, it forms weakly-visible chords with all points on the subchain from $p(r)$ to $q(r)$. Now we discuss how to represent $p(r)$ for a set of points. This is important in order to describe how the output is represented and stored. As r traverses along A_i, several events can occur:

(1) r can reach a vertex of A_i (other than $e(A_i)$),

(2) p can reach a vertex of B_i (other than $e(B_i)$),

(3) the pseudo-tangent point s can pivot about an edge of $SP(D_i)$,

(4) r can reach $e(A_i)$, or

(5) p can reach $e(B_i)$.

In order to easily store the output, we introduce Steiner points at r and p (if r and/or p are not already vertices) whenever one of the events (1)-(5) above occurs. The union of A_is and B_is have a total of $O(n)$ vertices. The total number of vertices and edges in the SSPT structure storing the $SP(D_i)$s is also $O(n)$. Hence, there are at most a total of $O(n)$ events that can occur. Thus the total number of Steiner points introduced is $O(n)$. The description of the output is piecewise and a different constant-sized piece is output at each event. Between events, however, we have (1) an edge of A_i containing r, (2) a similar edge of B_i containing p, and (3) a point s on $SP(D_i)$ that lies on $\overline{rp}$. Both the edges containing r and p connect two Steiner points and can be described by a linear equation involving their x- and y-coordinates. The point r can be described as a point with x-coordinate x and y-coordinate as a linear function of x. The point p can be described as a point with x-coordinate x' and y-coordinate as a linear function of x'. The line joining points r and p can be expressed as a linear equation in x and x'. Since this line must pass through s, it can be expressed as a linear equation in only one variable, say x. Furthermore, the coordinates of point p can also be expressed in terms of x. The range for x can be determined based on consecutive events, Thus, as r traverses through the infinitude of points on an edge, each has a unique p; the function $p(r)$ can be described in constant time and space. Given a point r, its corresponding $p(r)$ can thus be found in constant time. In this way, every point of a non-kernel basic interval has a linear function $p(r)$. In case (4) both s and therefore p may need to be updated. In case (5) the remainder of A_i does not have weakly-visible partners in B_i.

A symmetric procedure has r traversing clockwise around P in order to compute $q(r)$ for each r. By merging the Steiner points introduced while computing $q(r)$ with those from the computation of $p(r)$, we have that for every edge of a non-kernel basic interval, $p(r)$ and $q(r)$ are linear functions. Checking whether $p(r)$ precedes $q(r)$ counterclockwise for all points r on the edge can be done by comparing the values of $\beta(r)$ and $\gamma(r)$. For those r which violate this order, we return that they have no weakly-visible partners.

For those r which obey the order, the weakly-visible partners are the points on the interval from $p(r)$ to $q(r)$.

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 r, even though it is outside the interval $P_{CCW}(p, q)$. Throughout the above procedure, then, $b(B_i)$ and $e(B_i)$ are stored separately as weakly-visible partners whenever appropriate.

Tying up all the pieces together, we have shown a linear-time algorithm to compute all weakly-visible chords of a simple polygon.

7. Conclusion

This paper presents linear-time algorithms to compute all weakly-visible chords in a simple polygon. It may be noted that the algorithm presented in this paper uses Chazelle's linear-time triangulation [3] as part of its preprocessing. One of the open problems mentioned in the preliminary version of this paper was to design an algorithm for the weakly-visible chords problem without utilizing the triangulation algorithm. We believe that this open problem has been solved and will appear in a journal version of [7].

It is interesting to note that the results in this paper were used to develop an optimal linear-time algorithm for computing the shortest weakly-visible segment in the interior of a simple polygon [7]. We believe that the results and techniques presented here will be useful in finding efficient algorithms for other visibility problems in polygons.

8. Acknowledgements

The authors wish to thank the anonymous referees for their careful reading and for their many comments that greatly improved the presentation.

References

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

[2] B. Bhattacharya, A. Mukhopadhyay, Computing in linear time an internal line segment from which a simple polygon is weakly internally visible, *Manuscript*, 1993.

[3] B. Chazelle, Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, **6** pp.485–524, 1991.

[4] D.Z. Chen, Optimally computing the shortest weakly-visible edge of a simple polygon, *Proc. Fourth ISAAC*, LNCS **762**, pp.323–332, 1993.

[5] J. Doh, K. Chwa, An algorithm for determining visibility of a simple polygon from an Internal Line Segment, *J. of Algorithms*, **14**(1), pp.139–168, 1993.

[6] G. Das, P.J. Heffernan, G. Narasimhan, LR-visibility in polygons, *Proceedings of the 5th Canadian Conference on Computational Geometry*, pp.303–308, 1993. Submitted to special issue of *Computational Geometry - Theory and Appln.*.

[7] G. Das, G. Narasimhan, Optimal Linear-Time Algorithm for the Shortest Illuminating Line Segment in a Polygon, *Proceedings of the 10th Annual ACM Symp. on Computational Geometry*, pp. 259–268, 1994.

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

[9] P.J. Heffernan, An optimal algorithm for the two-guard problem, *Proceedings of the 9th Annual ACM Symp. on Computational Geometry*, pp.348–358, 1993.

[10] C. Icking, R. Klein, The two guards problem, *Proceedings of the 7th Annual ACM Symp. on Computational Geometry*, pp.166–175, 1991.

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

[12] D.T. Lee, F.P. Preparata, An optimal algorithm for finding the kernel of a polygon, *Journal of the ACM*, **26**(3), pp.415–421, 1979.

[13] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, *Springer Verlag*, 1985.

[14] J.-R. Sack, S. Suri, An optimal algorithm for detecting weak visibility, *IEEE Transactions on Computers*, **39**(10), pp.1213–1219, 1990.

[15] L.H. Tseng, D.T. Lee, Two-guard walkability of simple polygons, *Manuscript*, 1993.