Characterizing LR-Visibility Polygons and Related Problems

Binay K. Bhattacharya
School of Computing Science
Simon Fraser University
Burnaby, B.C. Canada V5A 1S6
binay@cs.sfu.ca

Subir Kumar Ghosh
School of Computer Science
Tata Institute of Fundamental Research
Bombay 400005, India
ghosh@tifrvax.tifr.res.in

Abstract

A simple polygon P is said to be LR-visibility polygon if there exists two points s and t on the boundary of P such that every point of the clockwise boundary of P from s to t (denoted as L) is visible from some point of the counterclockwise boundary of P from s to t (denoted as R) and vice versa. In this paper we derive properties of shortest paths in LR-visibility polygons and present a characterization of LR visibility polygons in terms of shortest paths between vertices. This characterization suggests a simple algorithm for the following recognition problem. Given a polygon P with distinguished vertices s and t, the problem is to determine whether P is a LR-visibility polygon with respect to s and t. Our algorithm for this problem checks LR-visibility by traversing shortest path trees rooted at s and t in DFS manner and it runs in linear time.

Using our characterization of LR-visibility polygons, we show that the shortest path tree rooted at a vertex or a boundary point can be computed in linear time for a class of polygons which contains LR-visibility polygons as a subclass. As a result, this algorithm can be used as a procedure for computing the shortest path tree in our recognition algorithm as well as in the recognition algorithm of Das et al [8]. Our algorithm computes the shortest path tree by scanning the boundary of the given polygon and it does not require triangulation as a preprocessing step.

1 Introduction

Characterizing, recognizing and computing visibility polygons under various criteria are central issues in visibility problems in computational geometry and related application areas [1]. The notion of visibility of a polygon from an internal segment arose when Avis and Toussaint [2] considered variations of the following art gallery problem: to place minimum number of stationary guards in an art gallery so that, together they can see every point in the interior of the gallery. In formal setting, the art gallery can be viewed as a simple polygon and guards as some points in the polygon. Avis and Toussaint [2] considered the case when number of guards is restricted to one but the guard is allowed to move along an edge of the polygon. Formally, this corresponds to finding an edge of the polygon such that every point in the polygon is visible from some point on the edge. Avis and Toussaint referred to visibility of a polygon from an edge as weak visibility of the polygon. A more general notion of (weak) visibility is one which allows visibility of the polygon from an internal segment, not necessary an edge. We refer to polygons, which have such an internal segment as weak visibility polygons. Weak visibility polygons have several interesting geometric properties, which allow simple and efficient algorithms for the class of weak visibility polygons [6, 7, 9, 11, 15, 17].

A weak visibility polygon P from a chord can be viewed as a polygon such that there exits two points s and t on the boundary of P such that (i) s and t are mutually visible and (ii) every point of the clockwise boundary from s to t (denoted as L) is visible from some point of the counterclockwise boundary from s to t (denoted as R) and vice versa. If we remove the restriction (i), then it defines a new class of polygons, called LR-visibility polygons which contains weak visibility polygons as a subclass.

In this paper we derive properties of shortest paths in LR-visibility polygons and present a characterization of LR-visibility polygons in terms of shortest paths between vertices. This characterization is similar to the characterization of weak visibility polygons given by Ghosh et al. [11]. Das, Heffernan and Narasimhan [8] also characterized LR-visibility polygons and their characterization is in terms of non-redundant components.

Our characterization of LR-visibility polygons suggests a simple algorithm for the following recognition problem. Given a polygon P with distinguished vertices s and t, the problem is to determine whether P is a LR-visibility polygon with respect to s and t. Our algorithm for this recognition problem checks LR-visibility by traversing shortest path trees rooted at s and t in DFS manner and it runs in linear time. The shortest path trees rooted at s and t can be computed in linear time by the algorithm of Guibas et al. [12]. This algorithm computes the shortest path tree by splitting funnels using a finger search tree and needs a triangulation of the given polygon which can be done in linear time by the algorithm of Chazelle [5].

Instead of using the algorithm of Guibas et al. as a procedure for our recognition algorithm, we show that using our characterization of LR-visibility polygons, the shortest path tree rooted at a vertex or a boundary point can also be computed in linear time for a class of polygons which contains LR-visibility polygons as a subclass. So, our algorithm for computing shortest path tree may terminate by reporting that the given polygon is not a LR-visibility polygon. If it computes the shortest path trees rooted at s and t, then they have to be traversed in DFS manner to check LR-visibility as stated earlier. Our algorithm computes the shortest path tree by scanning the boundary of the given polygon and it does not require triangulation as a preprocessing step.

Let us now look at the previous results on characterizing and recognizing LR-visibility polygons. Heffernan [13] presented a linear time algorithm for recognizing LR-visibility polygons with respect to the given pairs of points. Tseng, Heffernan and Lee [18] proposed an $O(nlogn)$ time algorithm and Das, Heffernan and Narasimhan [8] proposed a linear time algorithm for recognizing LR-visibility polygons by locating all pairs of points such that the given polygon is a LR-visibility polygon with respect to each of these pairs of points. However, all these algorithms require triangulation of P as a preprocessing step.

The algorithm of Das et al. needs a procedure for computing the shortest path tree rooted at some vertex as a preprocessing step in order to compute non-redundant components that are used later for checking LR-visibility. Their algorithm uses the linear time algorithm of Guibas et al. for computing the shortest path tree inside a polygon. Instead of using the algorithm of Guibas et al. as a procedure for their recognition algorithm, our algorithm for computing shortest path tree can be used. As a result, their overall recognition algorithm becomes simple as it does not require triangulation of the polygon as a preprocessing step and can also avoid using intricate data structure like finger search trees.

Above discussions claim that one of the merits of our algorithm for computing the shortest path tree is that our algorithm does not need triangulation of a polygon as a preprocessing step. Since a polygon can be triangulated in linear time by the algorithm of Chazelle [5], there is no advantage for the time complexity point of view to avoid the preprocessing step of triangulation. However, let us have a look at the algorithm of Chazelle [4, 5]. The algorithm is very intricate and uses involved tools and notions such as a *planar separator theorem*, *polygon cutting theorem*, *conformality*, etc. Though the algorithm is asymptotically optimal, it is conceptually difficult and too complex to be considered practical. Chazelle [4] mentions that the existence of a truly simple linear time algorithm remains open question and conjectures that such an algorithm is unlikely. Since the triangulation of a polygon can easily be obtained from the shortest path tree in linear time, our algorithm for computing the shortest path tree establishes that a truly simple asymptotically optimal algorithm for triangulating a simple polygon is possible for the class of LR-visibility polygons. Since the class of LR-visibility polygons contains many special classes of polygons such as spiral polygons, star-shaped polygons, weak visibility polygons, monotone polygons, etc., our algorithm can compute the shortest path tree in linear time in any of these special classes of polygons and therefore, triangulating these special classes of polygons is also possible in linear time.

The paper is organized as follows. In Section 2, we characterize LR-visibility polygons and present the recognition algorithm. In Section 3, we present the algorithm for computing the shortest path tree rooted at some vertex. In Section 4, we conclude the paper with a few remarks.

2 Characterizing and recognizing LR-visibility polygons

Let $SP(u,v)$ denote the Euclidean shortest path inside P from a point u to another point v. For any vertex u of P the *shortest path tree of P rooted at u*, denoted as $SPT(u)$, is the union of the shortest paths from u to all vertices of P. L (or, R) is referred to as the *same chain* of every point of L (respectively, R). Similarly, L (or, R) is referred to as the *opposite chain* of every point of R (respectively, L). In the following theorem we characterize LR-visibility polygons.

Theorem 2.1 *Let P denote a simple polygon. The following statements are equivalent.*

(i) P is a LR-visibility polygon with respect to vertices s and t.

(ii) Let u, x and v be any three vertices of L (or R) (including s and t) such that they are in clockwise (respectively, counterclockwise) order while traversing L (respectively, R) from s to t. $SP(u,x)$ and $SP(v,x)$ meet only at x.

(iii) For every vertex x of P, $SP(s,x)$ and $SP(t,x)$ meet only at x.

(iv) For any vertex $x \in L$ (or, $x \in R$), $SP(s,x)$ makes a left turn (respectively, a right turn) at every vertex of L (respectively, R) in the path. Analogously, for any vertex $x \in L$ (or, $x \in R$), $SP(t,x)$ makes a right turn (respectively, a left turn) at every vertex of R (respectively, L) in the path.

Proof: Firstly, we show that (i) implies (ii). Since P is a LR-visibility polygon with respect to s and t, x is visible some point y of the *opposite chain* of x by definition of LR-visibility polygons (Figure 1). So, the line segment xy partitions the polygon into two simple polygons, one containing s and u, and the other containing t and v. So, $SP(u,x)$ and $SP(v,x)$ cannot cross the line segment xy and therefore, they meet only at x.

Secondly, (ii) implies (iii) as a special case.

Thirdly, we show (iii) implies (iv). We prove only for the case when x belongs to L (Figure 2). Analogous arguments hold for the case when x belongs to R. If $SP(s,x)$ makes a right turn at some vertex $y \in L$, then $SP(t,x)$ also passes through y which contradicts (iii) as $SP(s,x)$ and $SP(t,x)$ meet at a vertex other than x. So, $SP(s,x)$ makes a left turn at y. Analogous arguments show that $SP(t,x)$ makes a right turn at every vertex of L in the path.

Fourthly we show that (iv) implies (i). It suffices to show that any vertex x of P is visible from some point of the *opposite chain* of x (Figure 1). Let z_c and z_{cc} be the vertices preceding x on $SP(s,x)$ and $SP(t,x)$ respectively. If z_c or z_{cc} belongs to the *opposite chain* of x, then the claim holds. So we assume that z_c, z_{cc} and x belong to the *same chain*. So, $SP(s,x)$ and $SP(t,x)$ make opposite turns at z_c and z_{cc} respectively by condition (iii) which means that extensions of xz_c and xz_{cc} from z_c and z_{cc} respectively cannot meet the *same chain* of x. Therefore, both extensions meet at points of the *opposite chain* of x which implies x is visible for some point of the *opposite chain* of x. Hence P is a LR-visibility polygon with respect to s and t. □

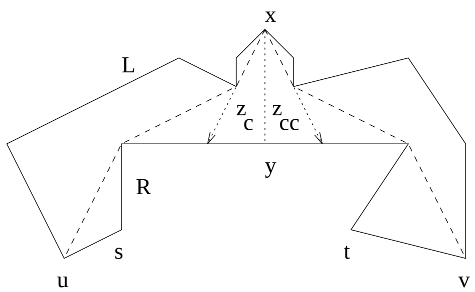

Figure 1: $SP(u,x)$ and $SP(v,x)$ meet only at x and x is visible from the *opposite chain* of x.

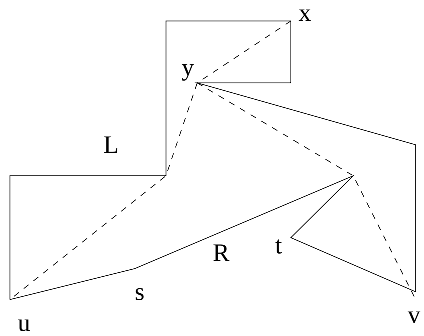

Figure 2: $SP(u,x)$ and $SP(v,x)$ meet at a vertex y other than x.

Theorem 2.1 suggests the following simple procedure for recognizing LR-visibility polygons given s and t.

1. Compute $SPT(s)$ and $SPT(t)$.

2. Starting from s, traverse $SPT(s)$ using DFS traversal and for every vertex x in $SPT(s)$, check the turn at x. If the path in $SPT(s)$ makes a right turn at $x \in L$ or makes a left turn at $x \in R$, then goto step 5.

3. Starting from t, traverse $SPT(t)$ using DFS traversal and for every vertex x in $SPT(t)$, check the turn at x. If the path in $SPT(t)$ makes a left turn at $x \in L$ or makes a right turn at $x \in R$, then goto step 5.

4. Report 'The given polygon is a LR-visibility polygon with respect to s and t' and goto step 6.

5. Report 'The given polygon is not a LR-visibility polygon with respect to s and t'.

6. Stop.

We now analyze the time complexity of the algorithm. $SPT(s)$ and $SPT(t)$ can be computed in linear time by the algorithm mentioned in Section 3. Steps 2 and 3 require the traversal of $SPT(s)$ and $SPT(t)$ in DFS fashion which takes linear time. Hence the overall time complexity of the algorithm is linear. We summarize the result in the following theorem.

Theorem 2.2 *Given a simple polygon P with distinguished vertices s and t, it can be determined in linear time whether or not P is a LR-visibility polygon with respect to s and t.*

3 An algorithm for computing the shortest path tree

In this section, we present a linear time algorithm for computing $SPT(a)$ rooted at some vertex a inside a given simple polygon P. If the given P is not a LR-visibility polygon with respect to any pair of boundary points of P, then the algorithm may terminate before computing the entire shortest path tree. If P is a LR-visibility polygon, then algorithm always succeeds in computing $SPT(a)$.

The visibility polygon of P from some internal point z of P is the set of all points of P that are visible for z and is denoted by $VP(z)$. Our algorithm first computes $VP(a)$ by the algorithm of Lee [16]. If $VP(a)$ is removed from P (Figure 3), then P is split into disjoint regions of P called *pockets* of $VP(a)$. Since the vertices of $VP(a)$ are visible from a, they are children of a in $SPT(a)$. So, the remaining task for computing $SPT(a)$ is to compute the shortest path from a to each vertex of all pockets. Since the shortest path from a to any two vertices b and c of different pockets are disjoint, i.e., $SP(a,b)$ and $SP(a,c)$ meet only at a, the shortest path from a to the vertices of one pocket can be computed independent of other pockets of $VP(a)$. So, it is enough to state the procedure for computing $SPT(a)$ to all vertices of one pocket. We have the following lemma.

Lemma 3.1 *If P is a LR-visibility polygon with respect to some pair of boundary points s and t, then at most one of s and t can lie in a pocket of $VP(a)$.*

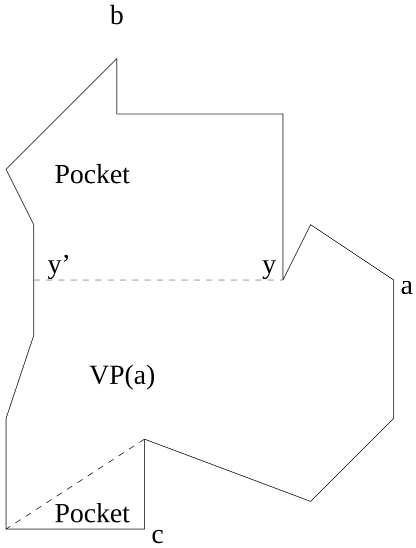

Figure 3: $P - VP(a)$ splits the polygon into pockets.

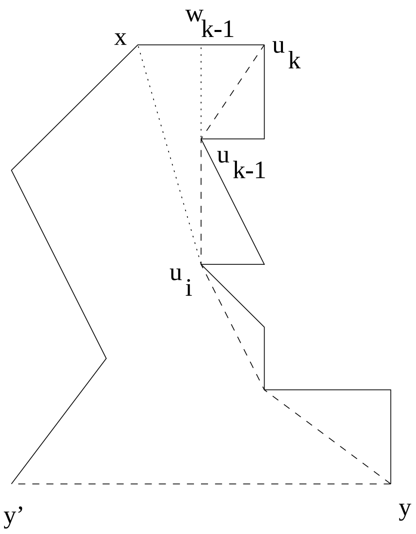

Figure 4: Figure shows that (u_{i-1}, u_i, u_{i+1}) is a right turn for all i and (u_{k-1}, u_k, x) is a left turn.

Proof: If both s and t lie in the same pocket of $VP(a)$, then the entire *opposite chain* of a lies in the pocket of $VP(a)$. Since no point of the pocket is visible from a, a is not visible from any point of the *opposite chain* of a. Hence P is not a LR-visibility polygon. $\square$

Keeping the above lemma in mind, we proceed to develop the procedure for computing the shortest path tree in a pocket. Let yy' be a non-polygonal edge of $VP(a)$ where y is a vertex of P and y' is some boundary point of P (Figure 3). Note that a, y and y' are collinear. We assume that if the boundary of P is traversed from a to y' in counterclockwise order, them y is encountered before reaching y'. So, the boundary of the pocket consists of the counterclockwise boundary of P from y to y' and the segment yy'. The counterclockwise (or clockwise) boundary from a boundary point z to another boundary point z' is denoted by $bd_{cc}(z, z')$ (respectively, $bd_c(z, z')$. Since the shortest path from a to any vertex of this pocket passes through y, it is enough to compute $SPT(y)$ to the vertices of the pocket.

We now present the algorithm for computing $SPT(y)$ to the vertices of this pocket. Let $SP_{cc}(y, u_k) = (y = u_0, u_1, ..., u_k)$ denote the shortest path restricted to $bd_{cc}(y, u_k)$ from y to a vertex u_k (Figure 4) such that (i) vertices u_1,..., u_k are vertices of $bd_{cc}(y, u_k)$ and (ii) for any $0 < i < k$, u_{i-1}, u_i and u_{i+1} is a right turn. Note that $SP(y, u_k)$ and $SP_{cc}(y, u_k)$ may be different as $SP(y, u_k)$ may pass through vertices of $bd_c(y', u_k)$ and therefore, $SP_{cc}(s, u_k)$ may not lie totally inside P. Union of $SP_{cc}(y, x)$ for all x of $bd_{cc}(y, y')$ is denoted by $SPT_{cc}(y)$. Analogously, union of $SP_c(y', x)$ for all x of $bd_c(y', y)$ is denoted by $SPT_{cc}(y)$.

The algorithm for computing $SPT_{cc}(y)$ scans $bd_{cc}(y, y')$ in counterclockwise order starting from y and computes $SP_{cc}(y, x)$ for every vertex x of $bd_{cc}(y, y')$ till it reaches y' or at some special vertex where the path from y is no longer a right turning path (Figure 6). Assume that $SP_{cc}(y, u_k) = (y = u_0, u_1, ..., u_k)$ has been computed and the algorithm wants to compute $SP_{cc}(y, x)$ where x is the next counterclockwise vertex of u_k. The following three cases can arise. To make the presentation simple, we assume from now on that no three vertices of P

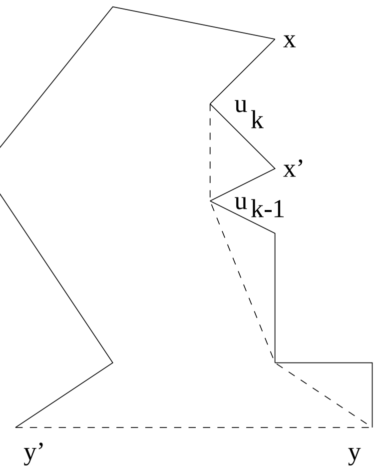

Figure 5: Figure shows that $SP_{cc}(y, x) = (u_0, u_1, ..., u_k, x)$.

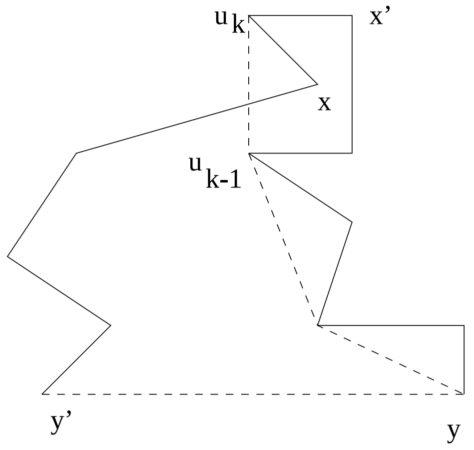

Figure 6: Figure shows a reverse turn at u_k.

are collinear.

Case 1: (u_{k-1}, u_k, x) is a left turn (Figure 4). Scan $SP_{cc}(y, u_k)$ from u_k till y is reached or a vertex u_i is reached such that (u_{i-1}, u_i, x) is a right turn. So, $SP_{cc}(y, x) = (u_0, u_1, ..., u_i, x)$. For every vertex u_j where $i < j < k$, extend $u_j u_{j-1}$ from u_j to xu_k and insert the point of intersection w_j on xu_k.

Case 2: (u_{k-1}, u_k, x) is a right turn and (x', u_k, x) is a right turn (Figure 5), where x' is the next clockwise vertex of u_k. So, $SP_{cc}(y, x) = (u_0, u_1, ..., u_k, x)$.

Case 3: (u_{k-1}, u_k, x) is a right turn and (x', u_k, x) is a left turn where x' is the next clockwise vertex of u_k (Figure 6). The algorithm returns $SPT_{cc}(y)$ up to the vertex u_k (denoted by $SPT_{cc}(y, u_k)$).

We refer Case 3 as a *reverse turn at* u_k. The above procedure can reach y' if there is no reverse turn. In that situation, entire $SPT_{cc}(y)$ (i.e., $SPT(y)$) has been computed and the algorithm terminates. Otherwise, for some vertex u_k, $SP_{cc}(y, u_k)$ has been computed and there is a reverse turn at u_k. In order to overcome the reverse turn at u_k, the algorithm tries to compute $SPT_c(y')$ by the analogous procedure mentioned as Case 1, Case 2 and Case 3 by scanning $bd_c(y', y)$ in clockwise order starting from y'. So, the following three cases can arise depending upon whether or not there is a reverse turn in $bd_c(y', y)$.

Case (i): Entire $SPT_c(y')$ has been computed.

Case (ii): $SPT_c(y', v_l)$ has been computed and there is a reverse turn at v_l where $v_l \in bd_{cc}(y, u_k)$ (Figure 7).

Case (iii): $SPT_c(y', v_l)$ has been computed and there is a reverse turn at v_l where $v_l \in bd_c(y', u_k)$ (Figure 8).

Consider Case (i). Since entire $SPT_c(y')$ has been computed, $SPT_c(y')$ is same as $SPT(y')$. We can compute $SPT(y)$ using the algorithm of Guibas et al [12] since the subpolygon can be triangulated easily once $SPT(y')$ is known. We can also use the algorithm of Hershberger

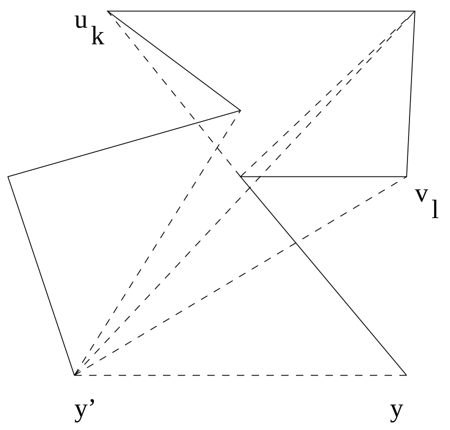

Figure 7: Figure shows a reverse turn at v_l where $v_l \in bd_{cc}(y, u_k)$.

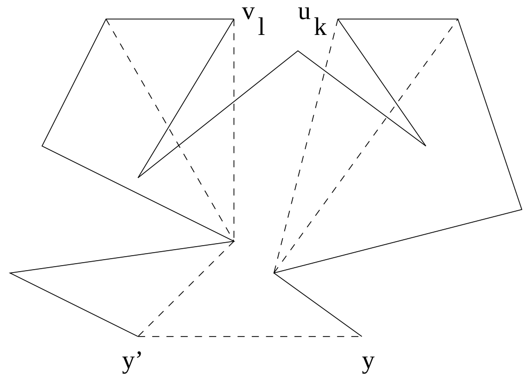

Figure 8: Figure shows a reverse turn at v_l where $v_l \in bd_c(y', u_k)$.

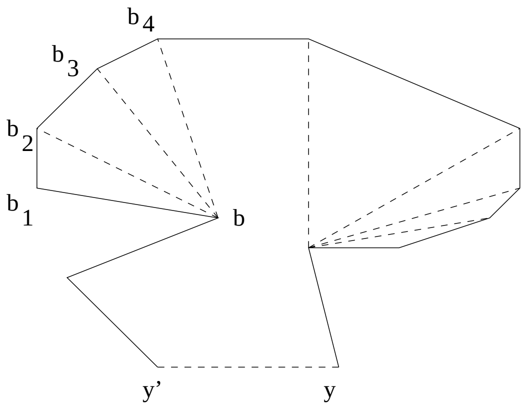

Figure 9: Figure shows that b_4 and b_1 are the last child and the first child of b respectively.

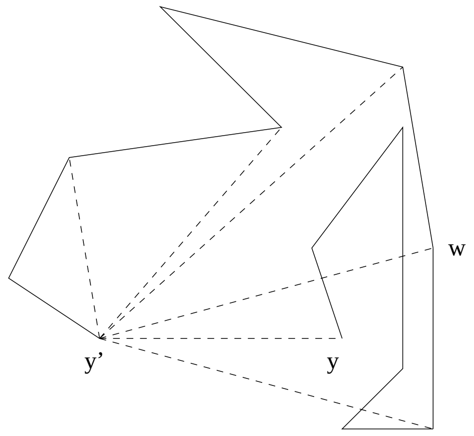

Figure 10: The vertex y lies inside the clockwise pocket induced by the segment y' and its last child in $SPT_c(y', v_l)$.

[14] to determine $SPT(y)$ from $SPT(y')$ and the time needed is proportional to the difference of their shortest path trees.

Consider Case (ii). We need the following definition. Let $b_1, b_2..., b_j$ be the children of a vertex b in $SPT_c(y')$ (or $SPT_{cc}(y)$) in clockwise (respectively, counterclockwise) angular order with respect to b (Figure 9), then b_1 is called the *first child* of b, b_i is called the *next sibling* of b_{i-1} for all i, and b_j is the *last child* of b. Observe that the edge connecting the first child and its parent is always a polygonal edge.

The algorithm has computed $SPT_c(y', v_l)$ and $SPT_{cc}(y, u_k)$ where $v_l \in bd_{cc}(y, u_k)$. Our algorithm computes $SPT(y')$ by merging $SPT_c(y', v_l)$ and $SPT_{cc}(y, u_k)$. The algorithm is quite similar to the algorithm of merging vertical decompositions of poygonal chains described in [3]. Once $SPT(y')$ is computed, then $SPT(y)$ can be computed from $SPT(y')$ by the procedure stated in Case (i). A *lid* is a line segment connecting two boundary points such that the segment contains two vertices. Note that a segment connecting two vertices is also called a lid. A lid always divides the polygon into two pockets, called the *clockwise pocket* and the *counterclockwise pocket* of the lid.

The merging procedure starts by initializing an edge of $bd_{cc}(y, u_k)$ to xz (called *inward edge*) and an edge uw of $SPT_c(y', v_l)$ as follows. If y lies inside the clockwise pocket induced by the lid connecting y' and the last child of y' in $SPT_c(y', v_l)$ (Figure 10), traverse the children of y' in $SPT_c(y', v_l)$ starting from the last child till a child w is located such that y does not lie in the clockwise pocket of $y'w$. Otherwise, the last child of y' in $SPT_c(y', v_l)$ is assigned to w. Now, z is initialized to y, x is initialized to the next counterclockwise vertex of z, u is initialized to y'.

Intuitively, the situation after initialization can be viewed as the inward edge xz is entering into the pocket induced by uw. So, in general, there are two situations: either the inward

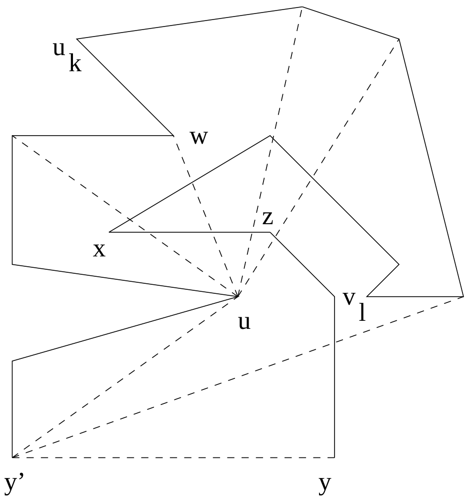

Figure 11: The inward edge xz from $bd_{cc}(y, u_k)$ is entering in the clockwise pocket induced by uw.

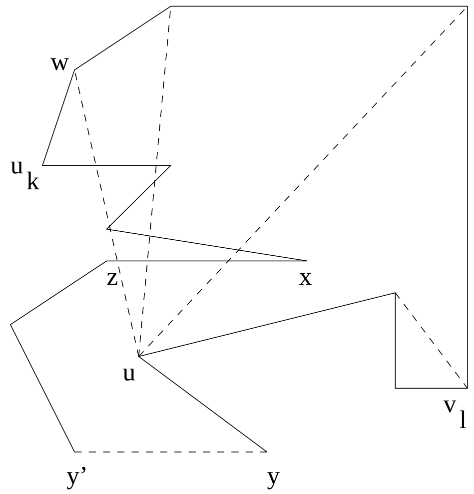

Figure 12: The inward edge xz from $bd_c(y', v_l)$ is entering in the counter-clockwise pocket induced by uw.

edge xz from $bd_{cc}(y, u_k)$ is entering in the clockwise pocket induced by uw in $bd_c(y', v_l)$ (Figure 11) or the inward edge xz from $bd_c(y', v_l)$ is entering in the counterclockwise pocket induced by uw in $bd_{cc}(y, u_k)$ (Figure 12). Since two cases are analogous, we present here only the case when inward edge xz from $bd_{cc}(y, u_k)$ is entering the clockwise pocket induced by uw in $bd_c(y', v_l)$ (Figure 11).

Step 1. *While* xz intersects uw, remove the edge uw from $SPT_c(y', v_l)$ and $w :=$ the last child of u in $SPT_c(y', v_l)$ (Figure 13);

Step 2. *If* xz intersects the extension ww' (if it exists) of uw (Figure 14), *then begin* $u := w$; $w :=$ the last child of w in $SPT_c(y', v_l)$; goto Step 1 *end*;

Step 3. *If* uw is an edge of $SP_c(y', v_l)$ and xz intersects the ray drawn from u through w (Figure 15), *then begin* $u := w$; w:= the last child of w in $SPT_c(y', v_l)$; goto step 1 end;

Step 4. Let x' be the next counterclockwise vertex of x. *If* (u, x, x') is a left turn (Figure 16), *then begin* $z := x$; $x := x'$; goto Step 1 end;

Step 5. *If* (z, x, x') is a left turn (Figure 17), *then begin* scan the boundary in counterclockwise order starting from x' till an edge bc intersecting either ux or ux'' is found; if bc intersects ux, then assign bc as the inward edge xz; goto Step 1 end; if bc intersects ux'', assign bc as the inward edge xz; u:=the parent of u in $SPT_{cc}(y')$; $w := u$; goto Step 1 end; { Old u will always have a parent if it is not y' }

Step 6. Let q be the point obtained by extending ux from x to the boundary. *If* q belongs to an edge of $bd_{cc}(y, v_l)$ (Figure 18), *then begin* assign u as the parent of x in $SPT_c(y', v_l)$; by scanning $bd_{cc}(x, q)$, compute the descendants of x by using the procedure stated as Case 1 and Case 2; assign the edge containing q as the inward edge xz; goto Step 1 *end*;

Step 7. *If* q belongs to an edge of $bd_c(u_k, v_l)$ (Figure 19), *then begin* assign u as the parent of x in $SPT_c(y', v_l)$; by scanning $bd_{cc}(x, q)$, compute the descendants of x by using the procedure

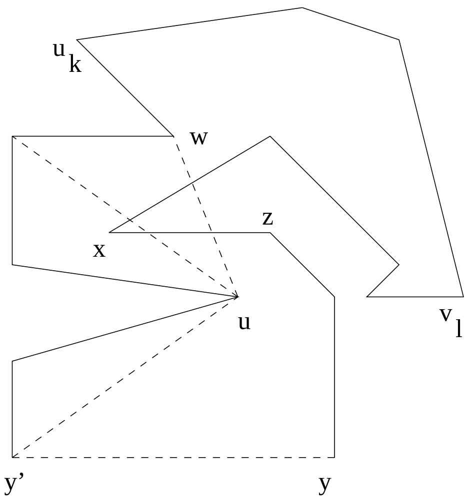

Figure 13: The inward edge xz intersects uw.

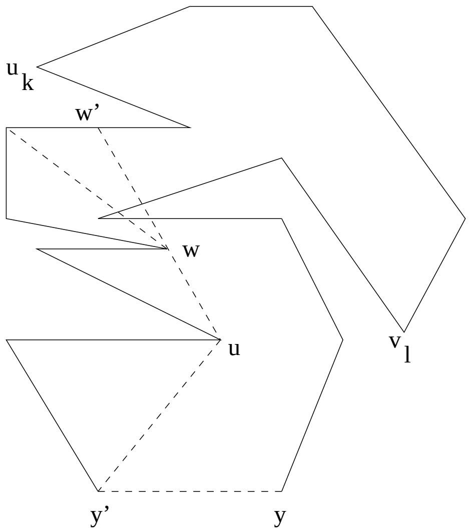

Figure 14: The inward edge xz intersects the extension ww'.

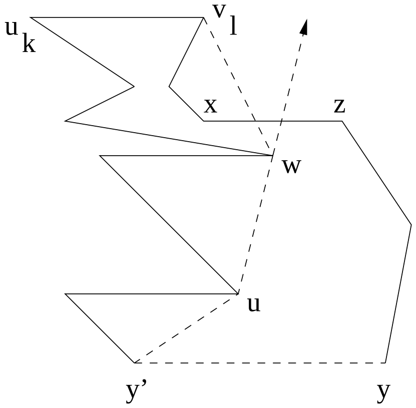

Figure 15: The inward edge xz intersects the ray drawn from u through w.

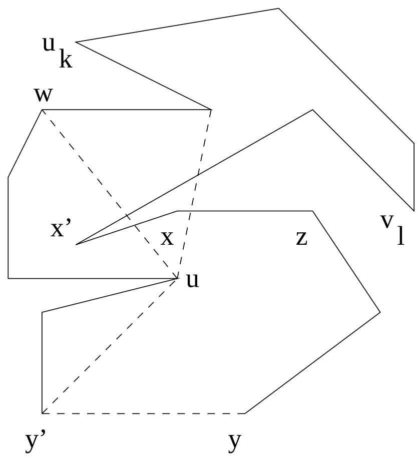

Figure 16: The inward edge xz is updated.

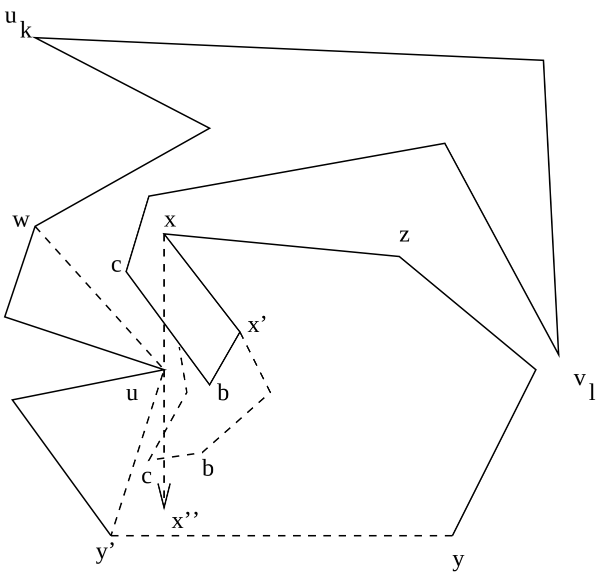

Figure 17: The inward edge xz is updated to an edge bc.

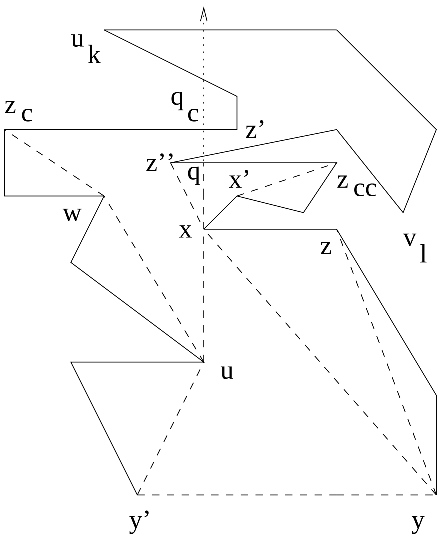

Figure 18: The extension point q belongs to an edge of $bd_{cc}(y, v_l)$.

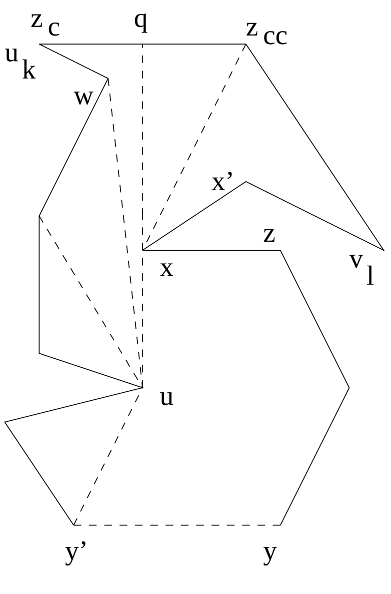

Figure 19: The extension point q belongs to an edge of $bd_{cc}(u_k, v_l)$.

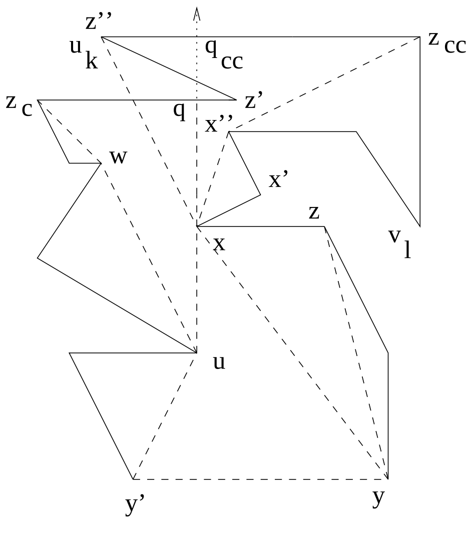

Figure 20: The extension point q belongs to an edge of $bd_c(y', u_k)$.

stated earlier as Case 1 and Case 2; goto Step 9 *end*;

Step 8. *If* q belongs to an edge of $bd_c(y', u_k)$ (Figure 20), *then begin* assign u as the parent of x in $SPT_c(y', v_l)$; locate the child x'' of x in $SPT_{cc}(y, u_k)$ such that the ray drawn from u through x passes between x'' and its next sibling; $w := x''$; $u := x$; assign the edge containing q as the inward edge xz; call the analogous procedure for checking the intersection of inward edge xz entering into the counterclockwise pocket induced by uw *end*;

Step 9. STOP.

After the above procedure is executed, all edges in $SPT_c(y', v_l)$ lie inside the polygon and therefore we call it as $SPT(y')$. Observe that the computation of $SPT_c(y')$ is not complete as it may not contain paths from y' to all vertices of $bd_c(y', y)$. If there is no path in $SPT_c(y')$ from y' to some vertex, then the vertex must be a vertex of some inward edge. So, once the parent of each vertex of inward edges is located then entire $SPT_c(y')$ has been computed.

Let z_1z_2, z_2z_3,..., $z_{j-1}z_j$ be the consecutive inward edges such that the parents of z_1 and z_j in $SPT(y')$ are x_1 and x_j respectively. We wish to compute the parents of z_2, z_3,..., z_{j-1} in $SPT(y')$. We know that either x_j is a descendant of x_1 or x_1 is a descendant of x_j. Without loss of generality we assume that x_j is a descendant of x_1. Consider the polygon consisting of $SP(x_1, x_j)$, x_jz_j, $z_{j-1}z_j$,..., z_2z_1, z_1x_1. In this polygon, $SPT(x_1)$ can be computed using the procedure mentioned as Case 1 and Case 2 if the chain of inward edges belongs to $bd_c(y', u_k)$ or by the analogous procedure of Case 1 and Case 2 if the chain of inward edges belongs to $bd_{cc}(y, v_l)$. Note that Case 3 (i.e., a reverse turn) cannot arise because it will contradict Lemma 3.1. Using this method, the parents of vertices of all inward edges can be computed.

Let us now explain how the extension point q in Step 6 (Figure 18) or Step 7 (Figure 19) or Step 8 (Figure 20) can be located on an edge. Let z_c be a leaf in $SPT_c(y', v_l)$ such that $SP_c(u, z_c)$ passes through w and every vertex in the path is the last child of its parent. Let z' be the next clockwise vertex of z_c. Let q_c be the point of intersection of the ray drawn from u through x and the polygonal edge $z'z_c$. Now, the child x'' of x is located in $SPT_{cc}(y, u_k)$ such that the ray drawn from u through x passes between x'' and its next sibling. Let z_{cc} be a leaf in $SPT_{cc}(y, u_k)$ such that every vertex in $SP_{cc}(x'', z_{cc})$ is the last child of its parent. Let z'' be the next counterclockwise vertex of z_c. Let q_{cc} be the point of intersection of the ray drawn from u through x and the polygonal edge $z''z_c$. If q_c lies on the segment xq_{cc} then q is assigned to q_c (Figure 20). Otherwise, q is assigned to q_{cc} (Figure 18).

Consider Case (iii). $SPT_c(y', v_l)$ and $SPT_{cc}(y, u_k)$ have been computed and there is a reverse turn at v_l as well as at u_k where $v_l \in bd_c(y', u_k)$ (Figure 8). We have the following lemma.

Lemma 3.2 *If P is a LR-visibility polygon with respect to two boundary points s and t, then either the parent of v_l in $SPT(y')$ is a vertex of $bd_{cc}(y, u_k)$ or the parent of u_k in $SPT(y)$ is a vertex of $bd_c(y', v_l)$.*

Proof: Let b_l and c_k be the parents of v_l and u_k in $SPT(y')$ and $SPT(y)$ respectively (Figure 21). If $b_l \in bd_c(y', v_l)$, then one of s and t, say s, must lie on $bd_c(b_l, v_l)$. If $b_l \in bd_c(v_l, u_k)$, then one of s and t, say s, must lie on $bd_c(v_l, b_l)$. So, if the parent of v_l in $SPT(y')$ is not a vertex of $bd_{cc}(y, u_k)$, then s must lie on $bd_c(y', v_l)$. Analogously, if the parent of u_k in $SPT(y)$

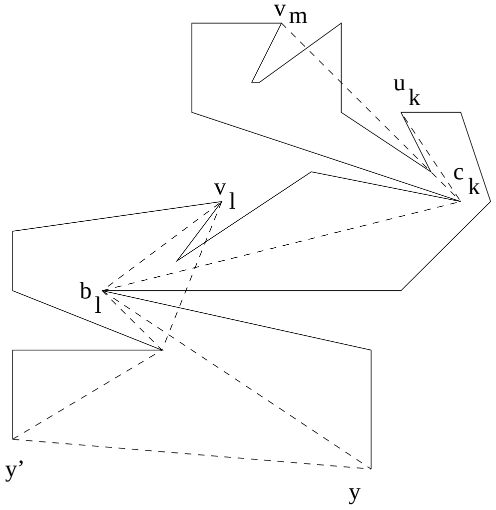

Figure 21: Figure shows that $SP(y', v_l)$ passes through a vertex of $bd_{cc}(y, u_k)$.

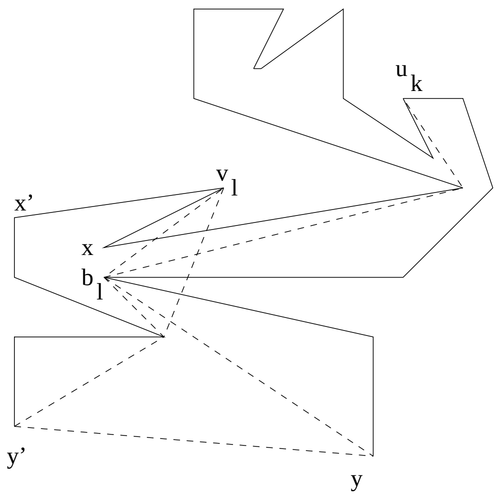

Figure 22: The reverse turn at v_l remains.

is not a vertex of $bd_c(y', v_l)$, then t must lie on $bd_{cc}(y, u_k)$. Since both s and t cannot lie on $bd_c(y', y)$ by Lemma 3.1, the parent of v_l in $SPT(y')$ is a vertex of $bd_{cc}(y, u_k)$ or the parent of u_k in $SPT(y)$ is a vertex of $bd_c(y', v_l)$. □

Above lemma suggests that if P is a LR-visibility polygon, then $bd_{cc}(y, u_k)$ has intersected $SP_c(y', v_l)$ or $bd_c(y', v_l)$ has intersected $SP_{cc}(y, u_k)$. These two tests should be performed in parallel. Since these tests are analogous, we state the procedure when $bd_{cc}(y, u_k)$ has intersected $SP_c(y', v_l)$ and $bd_c(y', v_l)$ has not intersected $SP_{cc}(y, u_k)$. This intersection can be checked by the merging procedure stated as Step 1 to Step 9 till v_l is reached as a vertex of an inward edge. If u_k is reached before v_l is reached as a vertex of an inward edge, the algorithm reports that the given polygon is not a LR-visibility polygon. So, we assume that the merging procedure has reached v_l as a vertex of an inward edge before reaching u_k. Let x and x' be the next clockwise and counterclockwise vertices of v_l. If (b_l, v_l, x) is a left turn and (x', v_l, x) is a right turn, i.e., the reverse turn at v_l remains (Figure 22), then the parent of v_l in $SPT(y')$ belongs to $bd_c(v_l, u_k)$. In that case P is not a LR-visibility polygon by Lemma 3.2. So, we assume that there is no reverse turn at v_l (Figure 21). Treating b_l as y' and v_l as the last vertex scanned so far, scan the clockwise boundary from v_l using the analogous procedure mentioned as Case 1, Case 2 and Case 3 till another reverse turn is encountered at some vertex v_m. Now there are reverse turn at v_m and u_k which is Case (iii) itself. The entire process of scanning and merging are repeated till it becomes Case (ii) or P is found not to be a LR-visibility polygon.

We now analyze the time complexity of the algorithm. Since $SPT_c(y')$ and $SPT_{cc}(y)$ are computed by considering each vertex of $bd_c(y', y)$ at most twice and each edge of $SPT_c(y')$ and $SPT_{cc}(y)$ is considered at most twice by the merging procedure, the overall complexity of the algorithm is linear. We summarize the result in the following theorem.

Theorem 3.1 *The shortest path tree from a point inside a simple polygon can be computed in linear time if the given polygon is LR-visibility polygon.*

4 Concluding remarks

As stated earlier, our algorithm computes the shortest path tree from a point a in each pocket of $VP(a)$ separately. Suppose the given polygon is a not a LR-visibility polygon but each pocket of $VP(a)$ satisfies the LR-visibility property. Then our algorithm succeeds in computing the shortest path tree. It will be interesting to identify the class of polygons for which our algorithm always succeeds in computing the shortest path tree from some vertex.

As mentioned in the introduction, one of the aims for developing the algorithm for computing the shortest path tree for the class of LR-visibility polygons is to show that the shortest path tree can be computed in linear time without the preprocessing step of triangulating the given polygon. Suppose it is possible to decompose a polygon into LR-visibility polygons in linear time. In that case, it is possible to triangulate each LR-visibility polygon by using our algorithm for computing the shortest path. So, the entire polygon can be triangulated once the polygon is decomposed into LR-visibility polygons. We feel that an arbitrary polygon can be decomposed into some numbers of LR-visibility polygons in linear time.

References

[1] T. Asano, S. K. Ghosh and T. C. Shermer. Visibility in the plane. *Handbook in Computational Geometry (ed. J.-R. Sack and J. Urruta)*, Elsevier Science Publishers B. W., to appear.

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

[3] B. Chazelle and J. Incerpi. Triangulation and shape complexity. *ACM Transactions on Graphics*, 3:135-152, 1984.

[4] B. Chazelle. Triangulating a simple polygon in linear time. Technical Report No. CS-TR-264-90, Department of Computer Science, Princeton University, 1990.

[5] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete and Computational Geometry*, 6:485-529, 1991.

[6] D. Chen. Optimally computing the shortest weakly visible subedge of a simple polygon. *In Proceedings of the 4th International Symposium on Algorithms and Computation*, pages 323-332, 1993.

[7] G. Das, P. J. Heffernan and G. Narasimhan. Finding all weakly-visible chords of a polygon in linear time. *In Scandinavian Workshop on Algorithm Theory*, pages 119-130, 1994.

[8] G. Das, P. J. Heffernan and G. Narasimhan. LR-visibility in polygons. *Computational Geometry: Theory and Applications*, 7:37-57, 1997.

[9] J. Doh and K. Chwa. An algorithm for determining internal line visibility of a simple polygon. *Journal of Algorithms*, 14:139-168, 1993.

[10] S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja, C. E. Veni Madhavan. Computing the shortest path tree in a weak visibility polygon. *In Proceedings of the 11th Symposium on Foundations of Software Technology and Theoretical Computer Science*, Lecture Notes in Computer Science, Springer-Verlag, 560:369-389, 1991.

[11] S. K. Ghosh, A. Maheshwari, S. P. Pal, S. Saluja and C. E. Veni Madhavan. Characterizing and recognizing weak visibility polygons. *Computational Geometry: Theory and Applications*, 3:213-233, 1993.

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

[13] P. J. Heffernan. An optimal algorithm for the two-guard problems. *Proceedings of the ACM Symposium on Computational Geometry*, pages 139-168, 1993.

[14] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4:141-155, 1989.

[15] S-H Kim, S. Y. Shin and K-Y Chwa. Efficient algorithms for solving diagonal visibility problems in a simple polygon. *International Journal of Computational Geometry and Applications*, 5:433-458, 1995.

[16] D. T. Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22:207-221, 1983.

[17] S. P. Pal. Weak visibility and related problems on simple polygons. Ph. D. Thesis, Department of Computer Science and Automation, Indian Institute of Science, India, September 1990.

[18] L.H. Tseng, P. Heffernan and D. T. Lee. Two-guard walkability of simple polygons. *International Journal of Computational Geometry*, 8:85-116, 1998.