Computational Geometry 18 (2001) 19–36
Computational Geometry Theory and Applications
www.elsevier.nl/locate/comgeo

Characterizing LR-visibility polygons and related problems ☆

Binay K. Bhattacharya [a], Subir Kumar Ghosh [b,*]

[a] School of Computing Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6
[b] School of Computer Science, Tata Institute of Fundamental Research, Mumbai 400005, India

Communicated by J.-R. Sack; received 24 April 1999; received in revised form 15 April 2000; accepted 12 October 2000

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, Heffernan and Narasimhan. 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. © 2001 Elsevier Science B.V. All rights reserved.

Keywords: Visibility; Euclidean shortest path; Shortest path tree; Merging

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

☆ The extended abstract of this paper was presented at the Tenth Canadian Conference on Computational Geometry, 1998.

* Corresponding author.
E-mail addresses: binay@cs.sfu.ca (B.K. Bhattacharya), ghosh@tifr.res.in (S.K. Ghosh).

PII: S0925-7721(00)00024-9

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 [5,6,8–10,14,16].

A weak visibility polygon P from a chord can be viewed as a polygon such that there exist 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. [10]. Das et al. [7] 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. [11]. 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 [4].

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 the trees can 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 [12] presented a linear time algorithm for recognizing LR-visibility polygons with respect to the given pair of points. Tseng et al. [17] proposed an $O(n \log n)$ time algorithm and Das et al. [7] 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 [4], there is no advantage from the time complexity point of view to avoid the preprocessing step of triangulation. However, let us have a look at the algorithm of Chazelle [3,4]. 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 [3] mentions that the existence of a truly simple linear time algorithm remains an 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 refereed to as the *same chain* of every point of L (respectively R). Similarly, L (or R) is refereed 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 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, then x is visible from some point y of the *opposite chain* of x (Fig. 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.

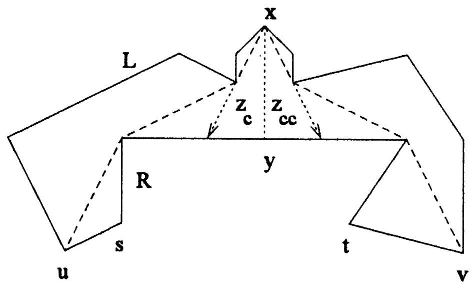

Fig. 1. $SP(u,x)$ and $SP(v,x)$ meet only at x and x is visible from the *opposite chain* of x.

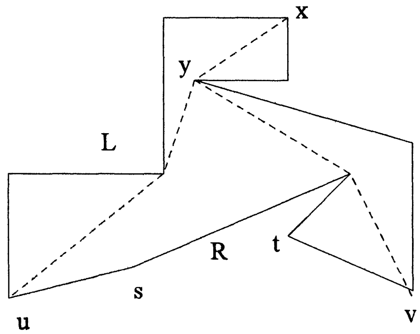

Fig. 2. $SP(u,x)$ and $SP(v,x)$ meet at a vertex y other than x.

Secondly, we show (ii) implies (iii). We prove only for the case when x belongs to L (Fig. 2). Assume on the contrary that $SP(s,x)$ makes a right turn at some vertex $y \in L$. Consider the convex angle formed by $SP(s,x)$ at y. If the angle is *facing* towards the interior of P, then by triangle inequality $SP(s,x)$ does not pass through y, a contradiction. If the angle is *facing* towards the exterior of P, then $SP(s,x)$ and $SP(t,x)$ meet at the vertex y other than x contradicting (ii) (Fig. 2). Hence, $SP(s,x)$ 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.

Thirdly, we show that (iii) implies (i). It suffices to show that any vertex x of P is visible from some point of the *opposite chain* of x (Fig. 1). We prove only for the case when x belongs to L. 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*. From the condition (iii) we know that $SP(s,x)$ makes a left turn at z_c and $SP(t,x)$ makes a right turn at z_{cc}. It means that extensions of xz_c and xz_{cc} from z_c and z_{cc} respectively cannot meet the *same chain* of x. So, both extensions meet at points of the *opposite chain* of x. Therefore, 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. □

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 [15]. If $VP(a)$ is removed from P (Fig. 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 $VP(a)$.*

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. □

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 (Fig. 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, then 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.

Before we present the algorithm for computing $SPT(y)$ in the pocket, we discuss the overall structure of the algorithm. It can be seen that for any vertex x in the pocket, $SP(y,x)$ satisfies one of the following properties.

1. $SP(y,x)$ passes through only the vertices of $bd_{cc}(y,x)$.
2. $SP(y,x)$ passes through only the vertices of $bd_c(y,x)$.
3. $SP(y,x)$ passes through vertices of both $bd_c(y,x)$ and $bd_{cc}(y,x)$.

If $SP(y,x)$ passes through only the vertices of $bd_{cc}(y,x)$, it means that $bd_c(y,x)$ does not interfere $SP(y,x)$ and therefore, $SP(y,x)$ can be computed by considering only $bd_{cc}(y,x)$. Similarly, if $SP(y,x)$ passes through only the vertices of $bd_c(y,x)$, it means that $bd_{cc}(y,x)$ does not interfere $SP(y,x)$ and

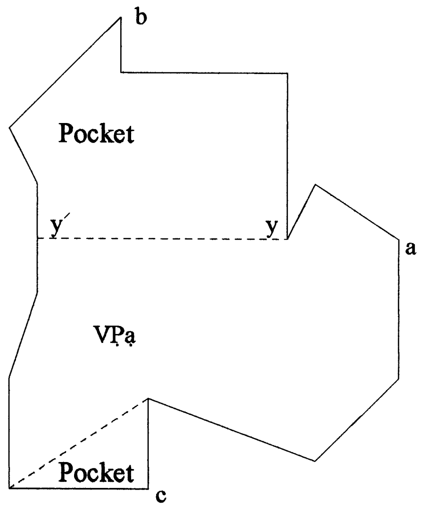

Fig. 3. $P - VP(a)$ splits the polygon into pockets.

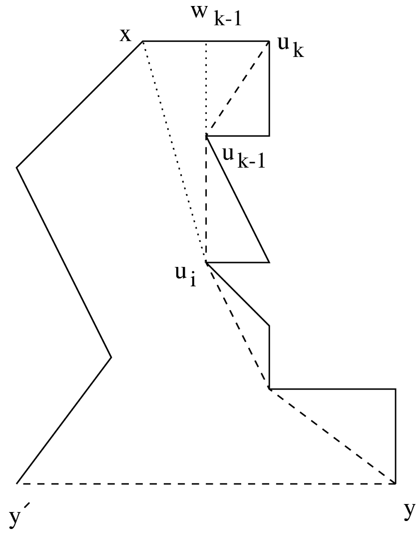

Fig. 4. Figure shows that (u_{i-1}, u_i, u_{i+1}) is a right turn for all i and (u_{k-1}, u_k, x) is a left turn.

therefore, $SP(y,x)$ can be computed by considering only $bd_c(y,x)$. However, if $SP(y,x)$ passes through vertices of both $bd_c(y,x)$ and $bd_{cc}(y,x)$, it is necessary to consider both $bd_c(y,x)$ and $bd_{cc}(y,x)$ for computing $SP(y,x)$. Instead of considering $bd_c(y,x)$ and $bd_{cc}(y,x)$ simultaneously, two convex paths from y to x can be constructed in $bd_c(y,x)$ and $bd_{cc}(y,x)$ separately and then merge these two convex paths to construct $SP(y,x)$.

Our algorithm scans $bd_{cc}(y,y')$ in counterclockwise order to construct convex paths from y to vertices of $bd_{cc}(y,y')$. During the scan, if the current vertex x interferes the convex path from y to the previous scanned vertex of x (say, u), then a partial tree consisting of convex paths from y to vertices of $bd_{cc}(y,u)$ is constructed. If no such vertex is encountered, convex paths from y to vertices of $bd_{cc}(y,y')$ form $SPT(y)$ because for all x, $SP(y,x)$ passes through only the vertices of $bd_{cc}(y,x)$. If u is located, $bd_c(y',y)$ is scanned in clockwise order to construct convex paths from y to vertices of $bd_c(y',y)$. During the scan, if the current vertex x interferes the convex path from y to the previous scanned vertex of x (say, v), then a partial tree consisting of convex paths from y to vertices of $bd_c(y,v)$ is constructed. If no such vertex is encountered, convex paths from y to vertices of $bd_c(y',y)$ form $SPT(y)$ because for all x, $SP(y,x)$ passes through only the vertices of $bd_c(y,x)$. If two partial trees from y to vertices of $bd_{cc}(y,u)$ and $bd_c(y,v)$ are constructed, it means that $SP(y,u)$ passes through vertices of $bd_c(y,v)$ or $SP(y,v)$ passes through vertices of $bd_{cc}(y,u)$. So, these two trees are merged to make one tree rooted at y such that paths from y to vertices of $bd_{cc}(y,u)$ now pass through vertices of $bd_c(y,v)$ and vice versa, i.e., paths in the merged tree are *outward convex*. Properties of LR-visibility polygons stated in Theorem 2.1 ensure that the outward convex path from y to u in the merged tree is $SP(y,u)$ or the outward convex path from y to v in the merged tree is $SP(y,v)$. The process of scanning and merging are repeated till $SPT(y)$ is constructed or the algorithm terminates by reporting that the given polygon P is not a LR-visibility polygon with respect to any pair of boundary points.

We now present the algorithm for computing $SPT(y)$ to the vertices of this pocket. Let $SP_{cc}(y,u_k) = (y = u_0, u_1, \ldots, u_k)$ denote the convex path restricted to $bd_{cc}(y,u_k)$ from y to a vertex u_k (Fig. 4) such

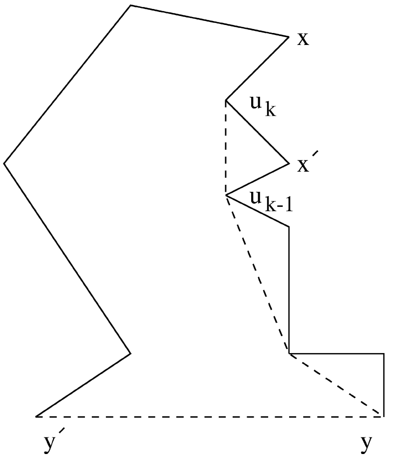

Fig. 5. Figure shows that $SP_{cc}(y, x) = (u_0, u_1, \ldots, u_k, x)$.

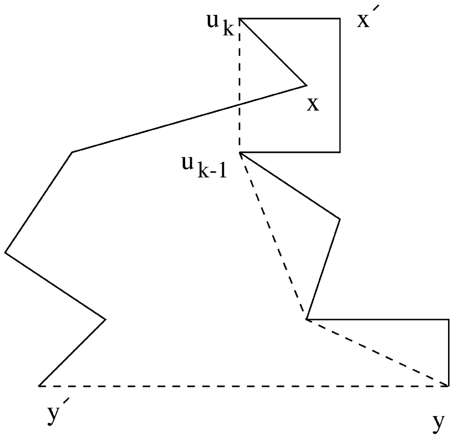

Fig. 6. Figure shows a reverse turn at u_k.

that (i) vertices $u_1, \ldots, u_k$ 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 $SPT_c(y)$.

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 $bd_{cc}(y, y')$ unless it encounters some special vertex where the path from y is no longer a right turning path (Fig. 6). Assume that $SP_{cc}(y, u_k) = (y = u_0, u_1, \ldots, u_k)$ 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 are collinear.

Case 1. (u_{k-1}, u_k, x) is a left turn (Fig. 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, $SP_{cc}(y, x) = (u_0, u_1, \ldots, u_i, x)$. 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 (Fig. 5), where x' is the next clockwise vertex of u_k. So, $SP_{cc}(y, x) = (u_0, u_1, \ldots, u_k, x)$.

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 (Fig. 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*. 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 Cases 1–3 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 3.1. Entire $SPT_c(y')$ has been computed.

Case 3.2. $SPT_c(y', v_l)$ has been computed and there is a reverse turn at v_l where $v_l \in bd_{cc}(y, u_k)$ (Fig. 7).

Case 3.3. $SPT_c(y', v_l)$ has been computed and there is a reverse turn at v_l where $v_l \in bd_c(y', u_k)$ (Fig. 8).

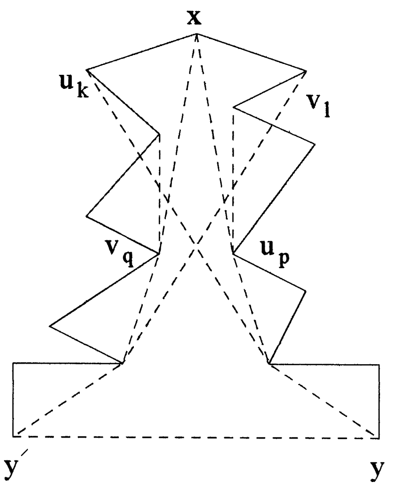

Fig. 7. Figure shows a reverse turn at v_l where $v_l \in bd_{\mathrm{cc}}(y, u_k)$.

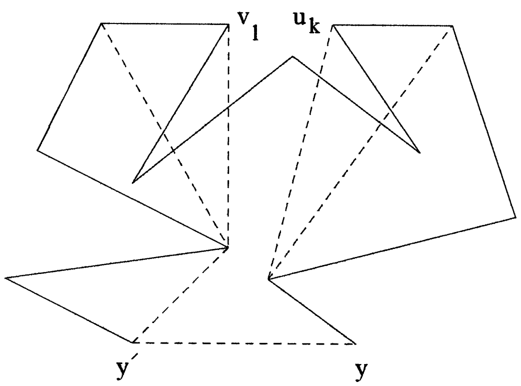

Fig. 8. Figure shows a reverse turn at v_l where $v_l \in bd_{\mathrm{c}}(y', u_k)$.

The remaining part of the algorithm consists of procedures for computing $SPT(y)$ in Cases 3.1–3.3 and they are stated in the following three subsections. In the last subsection, we analyze the time complexity of the entire algorithm.

3.1. Procedure for computing $SPT(y)$ in Case 3.1

In this subsection, we consider Case 3.1. Since entire $SPT_{\mathrm{c}}(y')$ has been computed, $SPT_{\mathrm{c}}(y')$ is the same as $SPT(y')$ as shown in the following lemma.

Lemma 3.2. *If no reverse turn is encountered during the clockwise scan of $bd_{\mathrm{c}}(y', y)$, then $SPT_{\mathrm{c}}(y')$ is the same as $SPT(y')$.*

Proof. In order to prove the lemma, we have to show that the parent of every vertex $x \in bd_{\mathrm{c}}(y', y)$ in $SPT(y')$ is the same as that of x in $SPT_{\mathrm{c}}(y')$. Since no reverse turn is encountered during the clockwise scan of $bd_{\mathrm{c}}(y', y)$, the parent of every vertex $x \in bd_{\mathrm{c}}(y', y)$ in $SPT_{\mathrm{c}}(y')$ is a vertex of $bd_{\mathrm{c}}(y', x)$. If the parent of x in $SPT_{\mathrm{c}}(y')$ is not the same as that of x in $SPT(y')$, then the parent z of x in $SPT(y')$ is a vertex of $bd_{\mathrm{cc}}(y, x)$. Let z' be the next counterclockwise vertex of z. So, z must be the parent of z' in $SPT(y')$ and by Theorem 2.1, the path from y' to z' in $SPT(y')$ makes a right turn at z. So, there is a reverse turn at z' while scanning $bd_{\mathrm{c}}(y', y)$ in clockwise order. □

The algorithm computes $SPT(y)$ from $SPT(y')$ by scanning $bd_{\mathrm{cc}}(y, y')$ in counterclockwise order starting from y. Though in this situation shortest paths in $SPT(y')$ make only left turns, we present the procedure for computing $SPT(y)$ for general situation when $SPT(y')$ is known and there are shortest paths in $SPT(y')$ that make right turns. Observe that the parent of a vertex x in $SPT(y')$ and $SPT(y)$ are different vertices if and only if x is visible from some point of the segment yy'. So, the algorithm has to compute shortest paths from y to only those vertices which are visible from some point of the segment yy' and for the remaining vertices, the parents are the same in both $SPT(y')$ and $SPT(y)$. For

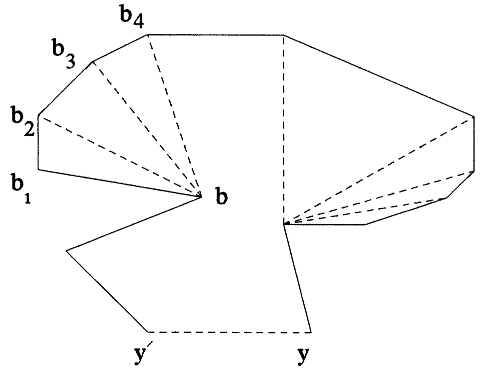

Fig. 9. Figure shows that b_4 and b_1 are the last child and the first child of b, respectively.

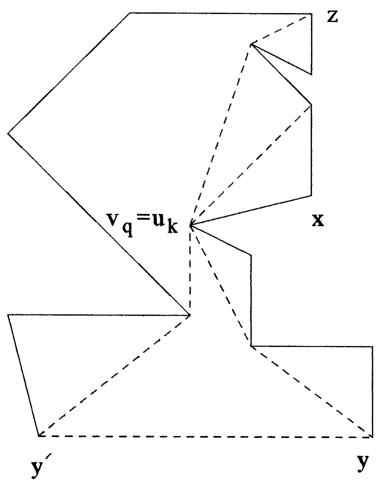

Fig. 10. Figure shows that v_q is same as u_k in $SP(y', u_k)$.

details of this concept, see Hershberger [13]. Before we use this concept in our algorithm, we need the following definition. Let $b_1, b_2, \ldots, b_j$ 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 (Fig. 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.

Assume that $SP(y, u_k) = (y = u_0, u_1, \ldots, u_k)$ has been computed for some vertex u_k where u_k is visible from some point of the segment yy'. Now, the algorithm wants to compute $SP(y, x)$ where x is the next counterclockwise vertex of u_k. Since u_k is visible from some point of yy', $SP(y', u_k)$ makes only left turns and $SP(y, u_k)$ makes only right turns and they meet only at u_k. Let $SP(y', x) = (y' = v_0, v_1, \ldots, v_q, x)$. If v_q and u_k are the same vertex (Fig. 10), then the subtree of $SPT(y')$ rooted at v_q becomes the subtree rooted at u_k in $SPT(y)$. Let z be a descendent in the subtree rooted at u_k such that each vertex in the path $SP(u_k, z)$ is the last child of its parent. Now the algorithm treats the next counterclockwise vertex of z as x and z as u_k. In the other situation, v_q is not the same as u_k (Fig. 11). Observe that yy', $SP(y', x)$, xu_k and $SP(y, u_k)$ form a *hourglass* where $SP(y', x)$ and $SP(y, u_k)$ are two sides of the hourglass. The algorithm locates the tangent ab in the hourglass between $SP(y', x)$ and $SP(y, u_k)$ where $a \in SP(y', x)$ and $b \in SP(y, u_k)$. So, $SP(y, x) = (y = u_0, u_1, \ldots, b, a, \ldots, v_q, x)$. Note that the tangent ab can be the edge xu_k. Observe that if a and x are not the same vertex, then the subtree of $SPT(y')$ rooted at a becomes the subtree rooted at a in $SPT(y)$ with b as the parent of a. Now the algorithm treats the next counterclockwise vertex of a as x and a as u_k. The procedure is repeated till y' is reached. Once the procedure reaches y', $SPT(y)$ has been computed.

3.2. Procedure for computing $SPT(y)$ in Case 3.2

In this subsection, we consider Case 3.2. The algorithm has computed $SPT_c(y', v_l)$ and $SPT_{cc}(y, u_k)$ where v_l belongs to $bd_{cc}(y, u_k)$ (Fig. 7). Our algorithm computes $SPT(y')$ by merging $SPT_c(y', v_l)$ and $SPT_{cc}(y, u_k)$. Once $SPT(y')$ is computed, then $SPT(y)$ can be computed from $SPT(y')$ by the procedure

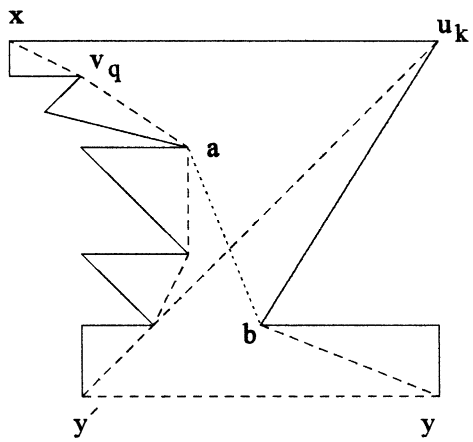

Fig. 11. Figure shows that v_q is not the same as u_k in $SP(y', u_k)$.

stated in Case 3.1. Before we state the procedure for merging $SPT_{\mathrm{c}}(y', v_l)$ and $SPT_{\mathrm{cc}}(y, u_k)$, we discuss the overall approach in the merging step. We start with the following lemma.

Lemma 3.3. *Let u_p and v_q be the parents of a vertex $x \in bd_{\mathrm{c}}(v_l, u_k)$ in $SPT_{\mathrm{cc}}(y, u_k)$ and $SPT_{\mathrm{c}}(y', v_l)$, respectively. If (u_p, x, v_q) is a left turn, then $SP_{\mathrm{cc}}(y, x)$ is the same as $SP(y, x)$ and $SP_{\mathrm{c}}(y', x)$ is the same as $SP(y', x)$ (Fig. 7).*

Proof. If (u_p, x, v_q) is a left turn, it means that $SP_{\mathrm{cc}}(y, x)$ and $SP_{\mathrm{c}}(y', x)$ meet only at x because $SP_{\mathrm{cc}}(y, x)$ makes only right turns and $SP_{\mathrm{c}}(y', x)$ makes only left turns. So, x is visible from some point of yy'. By Theorem 2.1, $SP_{\mathrm{cc}}(y, x)$ is the same as $SP(y, x)$ and $SP_{\mathrm{c}}(y', x)$ is the same as $SP(y', x)$. □

The above lemma suggests that if there exists such x, we can use the same procedure stated in Case 3.1 for computing $SPT(y)$ from $SPT(y')$ starting from x. If no such x exists, for every vertex $x \in bd_{\mathrm{c}}(v_l, u_k)$, both $SP(y, x)$ and $SP(y', x)$ pass through vertices of $bd_{\mathrm{cc}}(y, x)$ as well as vertices of $bd_{\mathrm{c}}(y', x)$. In order to identify vertices of $SP(y, x)$ and $SP(y', x)$, we need the following lemma.

Lemma 3.4. *For any vertex $x \in bd_{\mathrm{c}}(y', y)$, all vertices of $SP_{\mathrm{c}}(y', x)$ (or $SP_{\mathrm{cc}}(y, x)$) belong to $SP(y', x)$ (respectively $SP(y, x)$).*

Proof. The lemma follows from the fact that the line segment joining any two consecutive vertices of $SP_{\mathrm{c}}(y', x)$ (or $SP_{\mathrm{cc}}(y, x)$) is not intersected by any edge of $bd_{\mathrm{c}}(y', x)$ (respectively $bd_{\mathrm{cc}}(y, x)$). □

Though the above lemma suggests that all vertices of $SP_{\mathrm{c}}(y', x)$ belong to $SP(y', x)$, all edges of $SP_{\mathrm{c}}(y', x)$ do not belong to $SP(y', x)$. So, there exists at least one edge uw of $SP_{\mathrm{c}}(y', x)$ intersected by $bd_{\mathrm{cc}}(y, x)$. In order to compute $SP(y', x)$, the problem is to replace each such edge uw by $SP(u, w)$. Observe that since uw is intersected by an edge of $bd_{\mathrm{cc}}(y, x)$, $SP(u, w)$ must pass through some vertices of $bd_{\mathrm{cc}}(y, x)$. Moreover, $SP(u, w)$ may pass through some vertices of $bd_{\mathrm{c}}(u, w)$ (excluding u and w) where w is assumed to be the next vertex of u in $SP_{\mathrm{c}}(y', x)$. By testing the intersection between each edge of $SP_{\mathrm{c}}(y', x)$ with every edge of $bd_{\mathrm{cc}}(y, x)$, we can locate all such edges. However, this costly

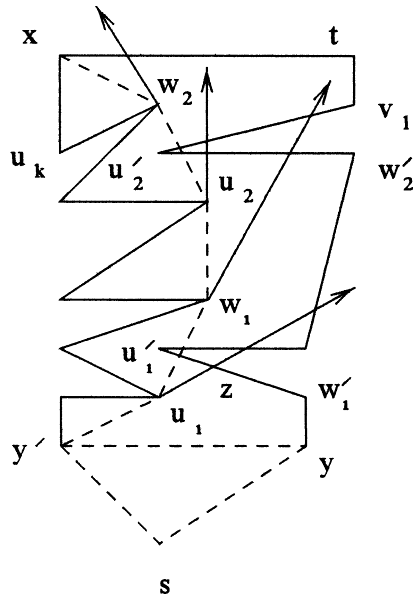

Fig. 12. The edge $u'_1w'_1$ intersects u_1w_1 and $u'_2w'_2$ intersects u_2w_2.

approach can be avoided because there exists an order in which edges of $bd_{cc}(y,x)$ intersect edges of $SP_c(y',x)$ in LR-visibility polygons. We state this property in the following lemmas.

Lemma 3.5. *Let P be a LR-visibility polygon with respect to boundary points s and t (Fig. 12). Let yy' be a segment inside P connecting two boundary points y and y'. Let u_1w_1 and u_2w_2 be two edges of $SP_c(y',x)$ for some vertex $x \in bd_{cc}(y,y')$ where u_1, w_1, u_2 and w_2 occur in this order in $SP_c(y',x)$. Let $u'_1w'_1$ be an edge of $bd_{cc}(y,x)$ such that $u'_1w'_1$ intersects u_1w_1. If any edge $u'_2w'_2 \in bd_{cc}(y,x)$ intersects u_2w_2, then $u'_2w'_2$ belongs to $bd_{cc}(u'_1,x)$.*

Proof. Without loss of generality, we assume that u'_1 is the next counterclockwise vertex of w'_1 and u'_1 lies in the region enclosed by u_1w_1 and $bd_c(u_1,w_1)$. Since $u'_1w'_1$ intersects u_1w_1, w_1 is visible only from $bd_{cc}(u'_1,u_1)$. Therefore, u_1 and u'_1 must belong to *opposite chain* because P is a LR-visibility polygon. So, s or t (say, t) belongs to $bd_{cc}(u'_1,u_1)$ and s belongs to $bd_c(u'_1,u_1)$. If $t \in bd_c(w_1,u_2)$ and any edge $u'_2w'_2$ of $bd_{cc}(y,x)$ intersects u_2w_2, then w_2 is not visible from any point of the opposite chain $bd_c(s,t)$. Therefore, either $u'_2w'_2$ does not intersect u_2w_2 or t belongs to $bd_c(u'_1,w_2)$. So, we assume that $t \in bd_c(u'_1,w_2)$ and $u'_2w'_2$ intersects u_2w_2. If $u'_2w'_2 \in bd_{cc}(y,w'_1)$, it means that both segments u'_1u_1 and u'_1w_1 are intersected by $bd_{cc}(y,w'_1)$ and therefore, u'_1 or w_1 is not visible from its *opposite chain*. Hence, $u'_2w'_2$ belongs to $bd_{cc}(u'_1,x)$. □

Lemma 3.6. *Let P be a LR-visibility polygon with respect to boundary points s and t (Fig. 12). Let yy' be a segment inside P connecting two boundary points y and y'. Let $y'u_1$ and u_1w_1 be two consecutive edges of $SP_c(y',x)$ for some vertex $x \in bd_{cc}(y,y')$. Let z be the first point of intersection between the ray drawn from y' through u_1 and $bd_{cc}(y,x)$, where $bd_{cc}(y,x)$ is traversed from y to x. Then no edge of $bd_{cc}(z,x)$ intersects $y'u_1$.*

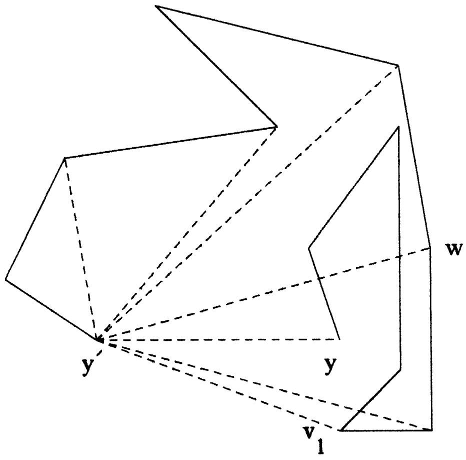

Fig. 13. The vertex y lies inside the clockwise pocket induced by the segment y' and its last child in $SPT_{\mathrm{c}}(y', v_l)$.

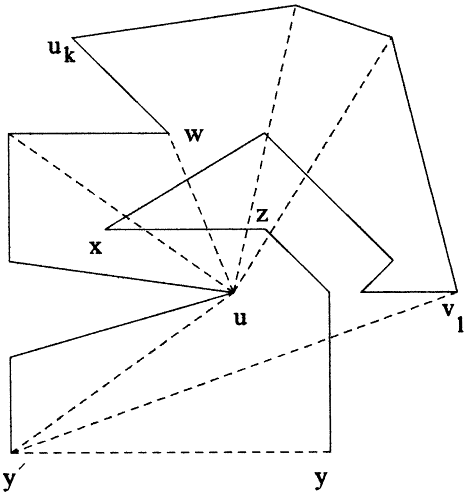

Fig. 14. The inward edge zx from $bd_{\mathrm{cc}}(y, u_k)$ is entering in the clockwise pocket induced by uw.

Proof. If any edge of $bd_{\mathrm{cc}}(z, x)$ intersects $y'u_1$, then some edge of $bd_{\mathrm{cc}}(z, x)$ also intersects zu_1. It means that z cannot be visible from any point of the *opposite chain* of z. Therefore, the given polygon is not a LR-visibility polygon, a contradiction. □

The above lemmas suggest a method for locating all edges of $bc_{\mathrm{cc}}(y, x)$ intersecting $SP_{\mathrm{c}}(y', x)$ as follows. Scan $bd_{\mathrm{cc}}(y, x)$ from y till an edge intersects $y'u_1$ or the ray drawn from y' through u_1. If the ray is intersected, repeat the process of checking the intersection treating u_1 as y'. If an edge of $bd_{\mathrm{cc}}(y, x)$ intersects $y'u_1$, then continue the scan till another edge intersects $y'u_1$. Once another edge intersecting $y'u_1$ is found, continue the scan as before to check for intersection with $y'u_1$ or the ray drawn from y' through u_1. We use this method in our merging procedure for locating every edge of $bd_{\mathrm{cc}}(y, u_k)$ intersecting any edge of $SPT_{\mathrm{c}}(y', v_l)$. 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 $bd_{\mathrm{cc}}(y, u_k)$ as an *inward edge* zx and an edge uw of $SPT_{\mathrm{c}}(y', v_l)$ as follows. If y lies inside the clockwise pocket induced by the lid connecting y' and the last child of y' in $SPT_{\mathrm{c}}(y', v_l)$ (Fig. 13), traverse the children of y' in $SPT_{\mathrm{c}}(y', v_l)$ 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 $SPT_{\mathrm{c}}(y', v_l)$ 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 zx is entering into the pocket induced by uw. So, in general, there are two situations: either the inward edge zx from $bd_{\mathrm{cc}}(y, u_k)$ is entering in the clockwise pocket induced by uw in $bd_{\mathrm{c}}(y', v_l)$ (Fig. 14) or the inward edge zx from $bd_{\mathrm{c}}(y', v_l)$ is entering in the counterclockwise pocket induced by uw in $bd_{\mathrm{cc}}(y, u_k)$ (Fig. 15). Since two

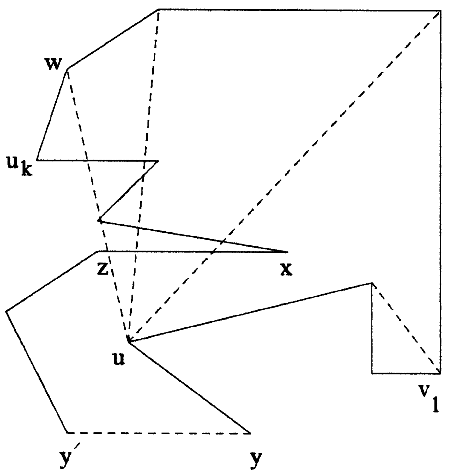

Fig. 15. The inward edge zx from $bd_{\mathrm{c}}(y', v_l)$ is entering in the counterclockwise pocket induced by uw.

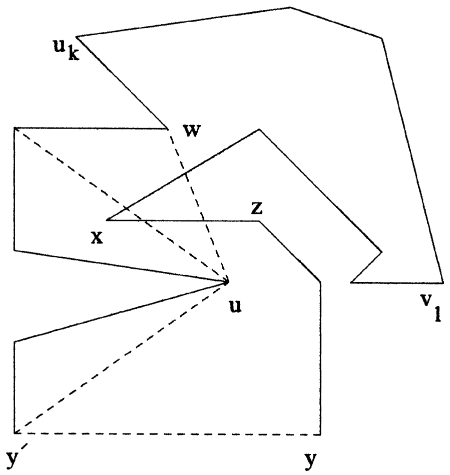

Fig. 16. The inward edge zx intersects uw.

cases are analogous, we present here only the case when inward edge zx from $bd_{\mathrm{cc}}(y, u_k)$ is entering the clockwise pocket induced by uw in $bd_{\mathrm{c}}(y', v_l)$ (Fig. 14).

Step 1. While zx intersects uw, remove the edge uw from $SPT_{\mathrm{c}}(y', v_l)$ and $w :=$ the last child of u in $SPT_{\mathrm{c}}(y', v_l)$ (Fig. 16);

Step 2. If zx intersects the extension ww' (if it exists) of uw (Fig. 17), *then begin* $u := w$; $w :=$ the last child of w in $SPT_{\mathrm{c}}(y', v_l)$; *goto Step 1 end*;

Step 3. If uw is an edge of $SP_{\mathrm{c}}(y', v_l)$ and zx intersects the ray drawn from u through w (Fig. 18), *then begin* $u := w$; $w :=$ the last child of w in $SPT_{\mathrm{c}}(y', v_l)$; *goto Step 1 end*;

Step 4. Let x' be the next counterclockwise vertex of x. *If* (u, x, x') is a left turn (Fig. 19), *then begin* $z := x$; $x := x'$; *goto Step 1 end*;

Step 5. If (z, x, x') is a left turn (Fig. 20), *then begin* scan the boundary in counterclockwise order starting from x' till an edge bc intersecting ux is found; assign bc as the inward edge zx; *goto Step 1 end*;

Step 6. [(z, x, x') is not a left turn.] Let q be the point obtained by extending ux from x to the boundary. *If* q belongs to an edge of $bd_{\mathrm{cc}}(y, v_l)$ (Fig. 21), *then begin* assign u as the parent of x in $SPT_{\mathrm{c}}(y', v_l)$; by scanning $bd_{\mathrm{cc}}(x, q)$, 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 zx; *goto Step 1 end*;

Step 7. If q belongs to an edge of $bd_{\mathrm{c}}(u_k, v_l)$ (Fig. 22), *then begin* assign u as the parent of x in $SPT_{\mathrm{c}}(y', v_l)$; by scanning $bd_{\mathrm{cc}}(x, q)$, compute the descendants of x by using the procedure stated earlier as Case 1 and Case 2; *goto Step 9 end*;

Step 8. If q belongs to an edge of $bd_{\mathrm{c}}(y', u_k)$ (Fig. 23), *then begin* assign u as the parent of x in $SPT_{\mathrm{c}}(y', v_l)$; locate the child x'' of x in $SPT_{\mathrm{cc}}(y, u_k)$ 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 zx; call the analogous procedure for checking the intersection of inward edge zx entering into the counterclockwise pocket induced by uw *end*;

Step 9. STOP.

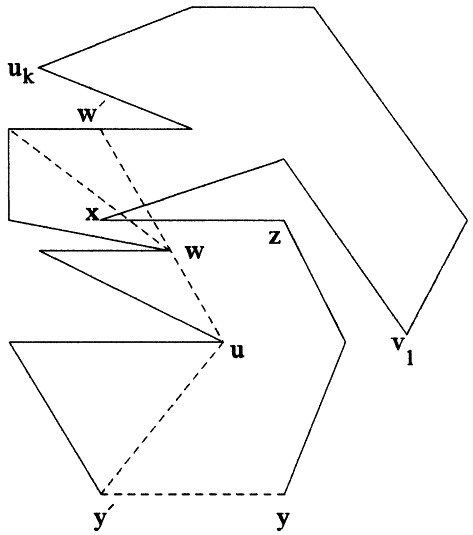

Fig. 17. The inward edge zx intersects the extension ww'.

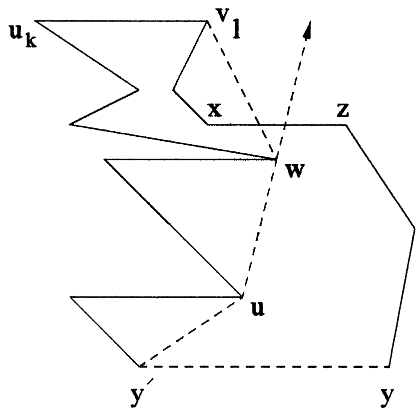

Fig. 18. The inward edge zx intersects the ray drawn from u through w.

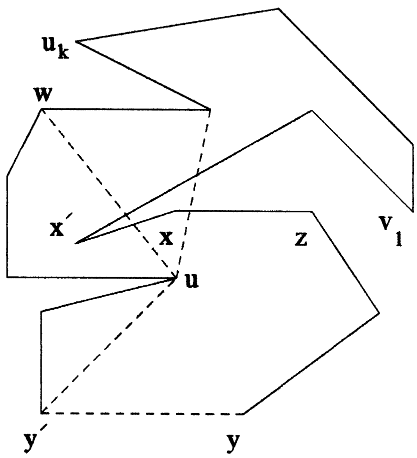

Fig. 19. The inward edge zx is updated.

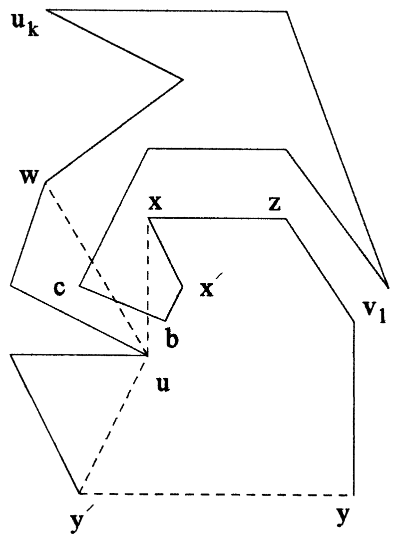

Fig. 20. The inward edge zx is updated to an edge bc.

Observe that after the above procedure is executed, 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. Let $z_1z_2, z_2z_3, \ldots, z_{j-1}z_j$ 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 (Fig. 24). We wish to compute the parents of $z_2, z_3, \ldots, z_{j-1}$ 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

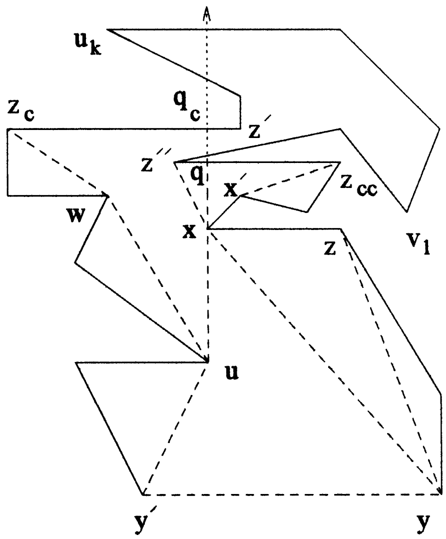

Fig. 21. The extension point q belongs to an edge of $bd_{\text{cc}}(y, v_l)$.

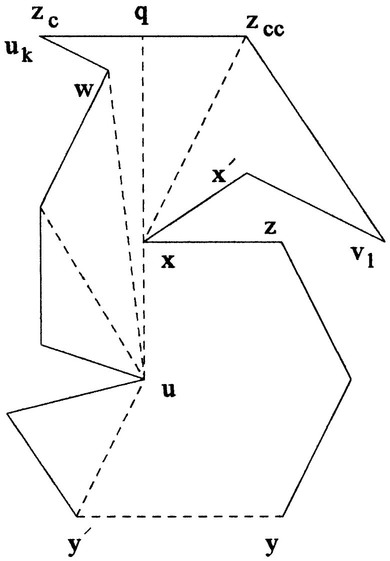

Fig. 22. The extension point q belongs to an edge of $bd_{\text{c}}(u_k, v_l)$.

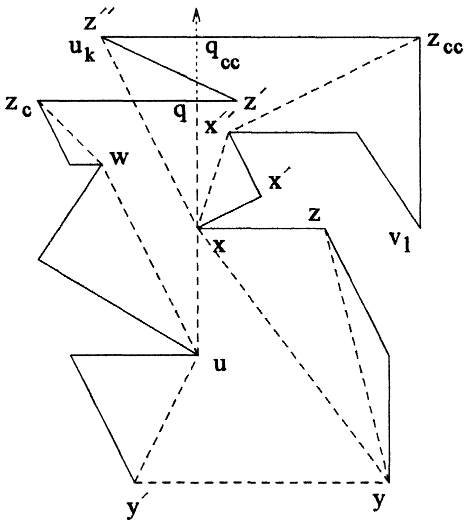

Fig. 23. The extension point q belongs to an edge of $bd_{\text{c}}(y', u_k)$.

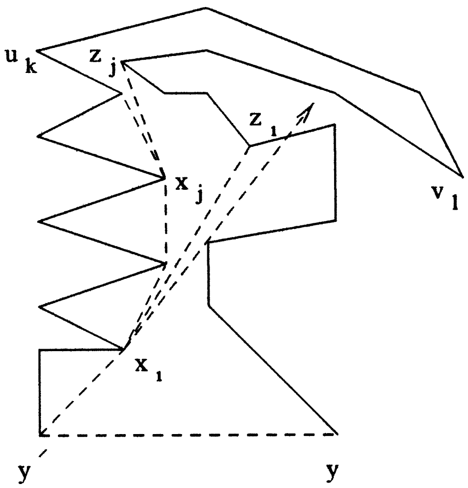

Fig. 24. Computing the parents of vertices of inward edges.

a descendant of x_1. Consider the polygon consisting of $SP(x_1, x_j)$, $x_j z_j, z_{j-1} z_j, \dots, z_2 z_1, z_1 x_1$. 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 $bd_{\text{c}}(y', u_k)$ or by the analogous procedure of Case 1 and Case 2 if the chain of inward edges belongs to $bd_{\text{cc}}(y, v_l)$. Observe that if Case 3 (i.e., a reverse turn) arises, it contradicts Lemma 3.1. Hence, the procedure mentioned as Case 1 and Case 2 can be used to compute parents of vertices of all inward edges. Once the parent of each vertex of inward edges is located, entire $SPT_{\mathrm{c}}(y')$ has been computed. Since all edges of $SPT_{\mathrm{c}}(y')$ lie inside the polygon and paths in $SPT_{\mathrm{c}}(y')$ are outward convex, $SPT_{\mathrm{c}}(y')$ has become $SPT(y')$. Once $SPT(y')$ is computed, $SPT(y)$ can be computed from $SPT(y')$ as stated in Case 3.1.

Let us now explain how the extension point q in Step 6 (Fig. 21) or Step 7 (Fig. 22) or Step 8 (Fig. 23) can be located on an edge. Let z_{c} be a leaf in $SPT_{\mathrm{c}}(y', v_l)$ such that $SP_{\mathrm{c}}(u, z_{\mathrm{c}})$ 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 $z'z_{\mathrm{c}}$. Now, the child x'' of x is located in $SPT_{\mathrm{cc}}(y, u_k)$ such that the ray drawn from u through x passes between x'' and its next sibling. Let z_{cc} be a leaf in $SPT_{\mathrm{cc}}(y, u_k)$ such that every vertex in $SP_{\mathrm{cc}}(x'', z_{\mathrm{cc}})$ is the last child of its parent. Let z'' be the next counterclockwise vertex of z_{cc}. Let q_{cc} be the point of intersection of the ray drawn from u through x and the polygonal edge $z''z_{\mathrm{c}}$. If q_{c} lies on the segment xq_{cc} then q is assigned to q_{c} (Fig. 23). Otherwise, q is assigned to q_{cc} (Fig. 21).

3.3. *Procedure for computing $SPT(y)$ in Case 3.3*

In this subsection, we consider Case 3.3. $SPT_{\mathrm{c}}(y', v_l)$ and $SPT_{\mathrm{cc}}(y, u_k)$ have been computed and there is a reverse turn at v_l as well as at u_k where $v_l \in bd_{\mathrm{c}}(y', u_k)$ (Fig. 8). We have the following lemma.

Lemma 3.7. *If P is a LR-visibility polygon with respect to two boundary points s and t, then the parent of v_l in $SPT(y')$ is a vertex of $bd_{\mathrm{cc}}(y, u_k)$ or the parent of u_k in $SPT(y)$ is a vertex of $bd_{\mathrm{c}}(y', v_l)$.*

Proof. Let b_l and c_k be the parents of v_l and u_k in $SPT(y')$ and $SPT(y)$, respectively (Fig. 26). If $b_l \in bd_{\mathrm{c}}(y', v_l)$, then one of s and t, say s, must lie on $bd_{\mathrm{c}}(b_l, v_l)$. If $b_l \in bd_{\mathrm{c}}(v_l, u_k)$, then one of s and t, say s, must lie on $bd_{\mathrm{c}}(v_l, b_l)$. So, if the parent of v_l in $SPT(y')$ is not a vertex of $bd_{\mathrm{cc}}(y, u_k)$, then s must lie on $bd_{\mathrm{c}}(y', v_l)$. Analogously, if the parent of u_k in $SPT(y)$ is not a vertex of $bd_{\mathrm{c}}(y', v_l)$, then t must lie on $bd_{\mathrm{cc}}(y, u_k)$. Since both s and t cannot lie on $bd_{\mathrm{c}}(y', y)$ by Lemma 3.1, the parent of v_l in $SPT(y')$ is a vertex of $bd_{\mathrm{cc}}(y, u_k)$ or the parent of u_k in $SPT(y)$ is a vertex of $bd_{\mathrm{c}}(y', v_l)$. □

The above lemma suggests that if P is a LR-visibility polygon, then $bd_{\mathrm{cc}}(y, u_k)$ has intersected the edge $b_l v_l$ or $bd_{\mathrm{c}}(y', v_l)$ has intersected the edge $c_k u_k$. Since these two situations are analogous, we state the procedure when $bd_{\mathrm{cc}}(y, u_k)$ has intersected $b_l v_l$ and $bd_{\mathrm{c}}(y', v_l)$ has not intersected the edge $c_k u_k$. Since the order of intersections satisfies Lemmas 3.5 and 3.6, intersections can be checked by the merging procedure stated as Steps 1–9 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 (Fig. 25), then the parent of b_l in $SPT(y')$ belongs to $bd_{\mathrm{c}}(v_l, u_k)$. In that case P is not a LR-visibility polygon by Lemma 3.7. So, we assume that (b_l, v_l, x) is a right turn (Fig. 26). 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 Cases 1–3 till another reverse turn is encountered at some vertex v_m in Case 3. Now there are reverse turns at v_m and u_k which is Case 3.3 itself. The entire process of scanning and merging is repeated till it becomes Case 3.2 or P is found not to be a LR-visibility polygon.

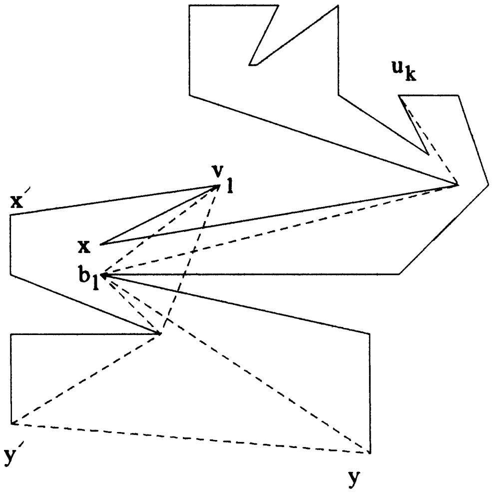

Fig. 25. The reverse turn at v_l remains.

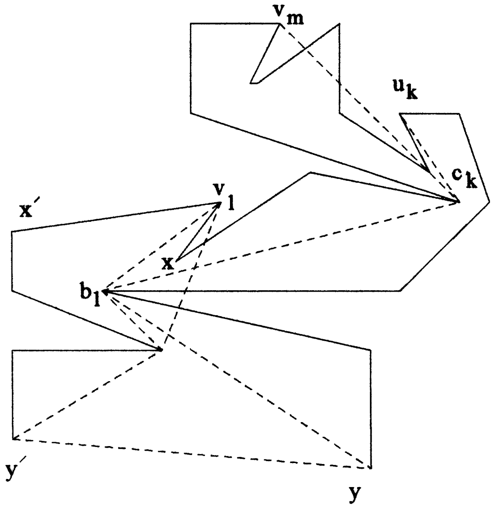

Fig. 26. Figure shows that $SP(y', v_l)$ passes through a vertex of $bd_{cc}(y, u_k)$.

3.4. *Analysis 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 a 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 any 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 tree. 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 number of LR-visibility polygons in linear time.

Acknowledgements

We gratefully acknowledge the helpful comments of Piyush Kumar, Sudeb Pal and Tom Shermer. The authors thank anonymous referees for suggesting improvements to the original paper.

References

[1] T. Asano, S.K. Ghosh, T.C. Shermer, Visibility in the plane, in: J.-R. Sack, J. Urrutia (Eds.), Handbook in Computational Geometry, Elsevier Science, 1999, Chapter 19, pp. 829–876.
[2] D. Avis, G.T. Toussaint, An optimal algorithm for determining the visibility of a polygon from an edge, IEEE Trans. Comput. C-30 (1981) 910–914.
[3] B. Chazelle, Triangulating a simple polygon in linear time, Technical Report No. CS-TR-264-90, Department of Computer Science, Princeton University, 1990.
[4] B. Chazelle, Triangulating a simple polygon in linear time, Discrete Comput. Geom. 6 (1991) 485–529.
[5] D. Chen, Optimally computing the shortest weakly visible subedge of a simple polygon, in: Proceedings of the 4th International Symposium on Algorithms and Computation, 1993, pp. 323–332.
[6] G. Das, P.J. Heffernan, G. Narasimhan, Finding all weakly-visible chords of a polygon in linear time, in: Scandinavian Workshop on Algorithm Theory, 1994, pp. 119–130.
[7] G. Das, P.J. Heffernan, G. Narasimhan, LR-visibility in polygons, Computational Geometry 7 (1997) 37–57.
[8] J. Doh, K. Chwa, An algorithm for determining internal line visibility of a simple polygon, J. Algorithms 14 (1993) 139–168.
[9] 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, Vol. 560, Springer, Berlin, 1991, pp. 369–389.
[10] S.K. Ghosh, A. Maheshwari, S.P. Pal, S. Saluja, C.E. Veni Madhavan, Characterizing and recognizing weak visibility polygons, Computational Geometry 3 (1993) 213–233.
[11] L. Guibas, J. Hershberger, D. Leven, M. Sharir, R. Tarjan, Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons, Algorithmica 2 (1987) 209–233.
[12] P.J. Heffernan, An optimal algorithm for the two-guard problems, in: Proceedings of the ACM Symposium on Computational Geometry, 1993, pp. 139–168.
[13] J. Hershberger, An optimal visibility graph algorithm for triangulated simple polygons, Algorithmica 4 (1989) 141–155.
[14] S.-H. Kim, S.Y. Shin, K.-Y. Chwa, Efficient algorithms for solving diagonal visibility problems in a simple polygon, Internat. J. Comput. Geom. Appl. 5 (1995) 433–458.
[15] D.T. Lee, Visibility of a simple polygon, Comput. Vision Graphics Image Process. 22 (1983) 207–221.
[16] 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.
[17] L.H. Tseng, P. Heffernan, D.T. Lee, Two-guard walkability of simple polygons, Internat. J. Comput. Geom. Appl. 8 (1998) 85–116.