Available online at www.sciencedirect.com

SCIENCE DIRECT®

Computational Geometry 33 (2006) 165–173

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

A linear time algorithm to remove winding of a simple polygon

Binay Kumar Bhattacharya [a], Subir Kumar Ghosh [b,*,1], Thomas Caton Shermer [a]

[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*

Received 29 November 2004; accepted 3 May 2005

Available online 11 August 2005

Communicated by T. Asano

Abstract

In this paper, we present a linear time algorithm to remove winding of a simple polygon P with respect to a given point q inside P. The algorithm removes winding by locating a subset of *Jordan sequence* that is in the proper order and uses only one stack.

Keywords: Algorithm; Pruning; Revolution; Visibility polygon; Winding

1. Introduction

Determining the visible region of a geometric object from a given source under various constraints is a well-studied problem in computational geometry [1]. Two points of a simple polygon P is said to be *visible* if the line segment joining them lies inside P. The *visibility polygon* of a point q in P is the set of all points of P visible to q (see Fig. 1(a)). A similar definition holds in a polygon with holes (see Fig. 1(b)). This problem of computing the visibility polygon $V(q)$ from a point q is an integral part of the rendering process in computer graphics, where it is called *hidden line elimination* or *hidden surface elimination* [5].

The problem of computing $V(q)$ inside a simple polygon P of n vertices was first taken up in a theoretical setting by Davis and Benedikt [4], who presented an algorithm that takes $O(n^2)$ time. Soon thereafter, ElGindy and Avis [6] and Lee [12] gave linear-time algorithms for this problem. For a polygon with h holes of total n vertices, Asano [3] presented $O(n \log h)$ algorithms for computing the visibility polygon of a point. Around the same time, $O(n \log n)$ time algorithm for this problem was proposed by Suri and O'Rourke [13], and Asano et al. [2]. Later, Heffernan and Mitchell [8] presented an $O(n + h \log h)$ time algorithm for this problem.

It has been shown in Joe [10] and Joe and Simpson [11] that both algorithms of ElGindy and Avis, and Lee may fail on some polygons with sufficient winding, i.e., if the revolution number is at least two. For any point $z \in P$, the *revolution number* of P with respect to z is the number of revolutions that the boundary of P makes about z. Joe and

* Corresponding author.
E-mail addresses: binay@cs.sfu.ca (B.K. Bhattacharya), ghosh@tifr.res.in (S.K. Ghosh), shermer@cs.sfu.ca (T.C. Shermer).
1 A part of this work was done when the author visited Simon Fraser University and was supported by NSERC grants.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2005.05.001

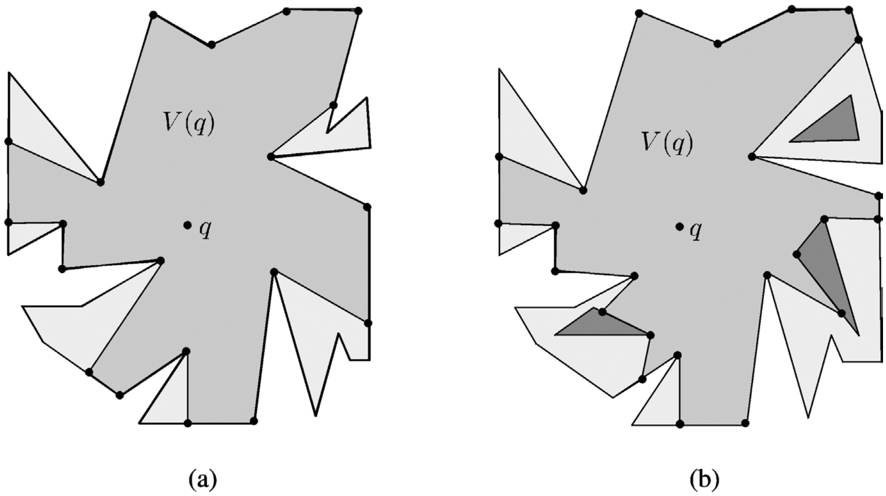


Fig. 1. The visibility polygons of q in a simple polygon and in a polygon with holes.

Simpson [11] suggested a linear time algorithm for computing $V(q)$ which correctly handles winding in the polygon by keeping the count of the number of revolutions around q.

It can be seen that the portion of the boundary of P that makes the revolution number of q more than one is not visible from q. So, it is better to prune P before using the algorithm of ElGindy and Avis or Lee so that (i) the revolution number of the pruned polygon of P with respect to q is one and (ii) the pruned polygon of P contains both q and $V(q)$. In the next section, we discuss in details the need for such pruning in the context of computing $V(q)$. In Section 3, we present our O(n) time algorithm for pruning P. In Section 4, we conclude the paper with a few remarks.

2. Background

As stated earlier, Lee's algorithm works in general but it may fail on some polygons with sufficient winding as pointed out in Joe [10] and Joe and Simpson [11]. The polygon in Fig. 2(a) is one such polygon. While scanning the

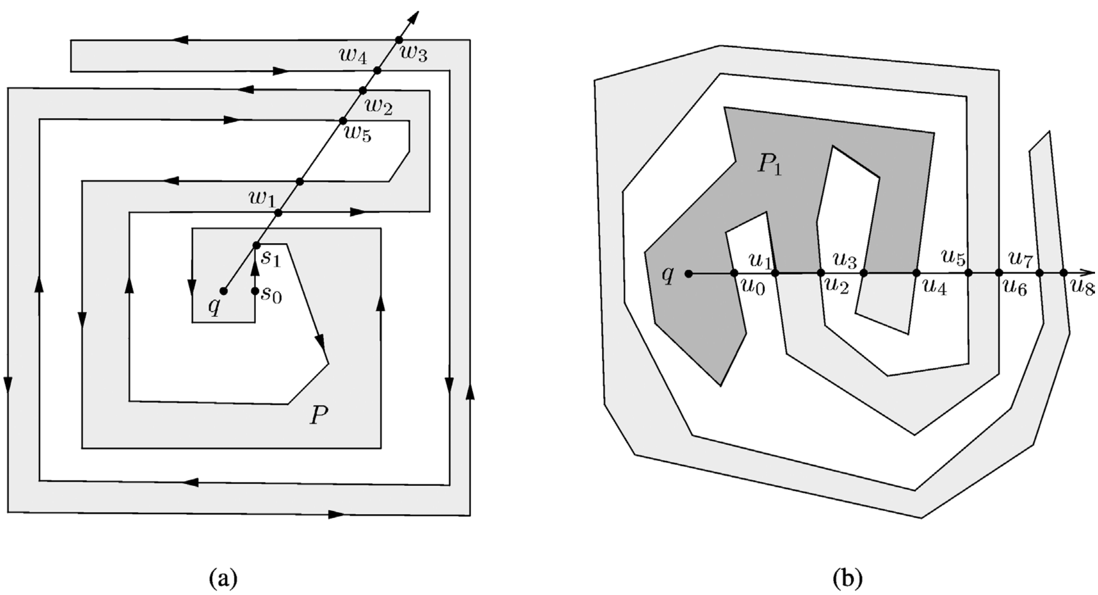


Fig. 2. (a) The algorithms of ElGindy and Avis, and Lee fail for this polygon P. (b) The polygon P can be divided by segments u_1u_2, u_3u_4, u_5u_6 and u_7u_8. The shaded region P_1 contains both q and $V(q)$.

boundary of P (denoted as $bd(P)$) in counterclockwise order starting from s_0, Lee's algorithm pushes s_0 and s_1 on the stack. Then it looks for the intersection of $bd(P)$ with the ray drawn from q through s_1 (denoted as $\overrightarrow{qs_1}$) and locates the intersection point w_1. Since $bd(P)$ has intersected $\overrightarrow{qs_1}$ at w_1 in the opposite direction, w_1 and the next intersection point w_2 are ignored. The algorithm correctly accepts the next intersection point w_3. It again returns to $\overrightarrow{qs_1}$ and locates the intersection point w_4 on s_1w_3. Then it locates the next intersection point w_5 by checking intersection of $bd(P)$ with s_1w_4 and pushes w_5 on the stack. Since then, the algorithm does not compute $V(q)$ correctly. Observe that $bd(P)$ has intersected s_1w_4 at w_5 from the opposite direction due to winding.

It can be seen that if the winding part of the input polygon P is removed before using Lee's algorithm, it correctly compute $V(q)$. Let $P_1 \subseteq P$ be a pruned polygon (see Fig. 2(b)) such that P_1 contains both q and $V(q)$, and the angle subtended at q is no more than 2π while scanning the boundary of P_1. Then P_1 and q can be given as inputs to Lee's algorithm to compute $V(q)$. We start the discussion on pruning with the following lemma.

Lemma 2.1. *Let $(u_0, u_1, \ldots, u_k)$ be the intersection points of $bd(P)$ with the half-line drawn from q to the right of q such that for all i, $u_i \in qu_{i+1}$. Then, the segments $u_1u_2, u_3u_4, \ldots, u_{k-1}u_k$ lie inside P.*

Lemma 2.1 suggests that since $(u_0, u_1, \ldots, u_k)$ is in sorted order along the half-line (Fig. 2(b)), P can be partitioned into several parts by adding the segments $u_1u_2, u_3u_4, \ldots, u_{k-1}u_k$. Observe that the part containing u_0 is a pruned polygon P_1 which contains q as well as $V(q)$. Analogously, remove winding by drawing a horizontal line from q to the left of q by treating P_1 as P. Since there is no winding now in the new P_1, the angle subtended at q cannot be more than 2π while scanning the boundary of the new P_1 by Lee's algorithm.

Intersection points $u_0, u_1, \ldots, u_k$ can be computed in $O(n)$ time by checking the intersection of the half-line with every edge of P. Then, intersection points can be sorted along the half-line in $O(n)$ time using the algorithm of Hoffmann et al. [9]. Note that sorting of n numbers in general is different from sorting of these intersection points $(u_0, u_1, \ldots, u_k)$ lying on $bd(P)$. Hence, the overall time complexity of the algorithm for computing $V(q)$ remains $O(n)$. However, the algorithm of Hoffmann et al. [9] uses involved data structures called *L*evel-linked search trees, which are not easy to implement. In our pruning algorithm, we adopt a different method for computing P_1, which uses only one stack.

Observe that if only the segment u_1u_2 or u_5u_6 is added to the polygon in Fig. 2(b), it still removes winding from P. It suggests that winding can be removed by introducing a few selected segments in P. Our pruning algorithm shows that such segments can be identified without sorting all intersection points of $bd(P)$ with the horizontal line (called Jordan sequence).

3. Pruning algorithm

Pruning algorithm starts by drawing the horizontal line L through q. Let L_r and L_l denote the portion of L to the right and left of q respectively (see Fig. 3(a)). Let q_r (or q_l) be the closest point of q among the intersection points of $bd(P)$ with L_r (respectively, L_l). Add the segment q_lq_r to partition P into polygons P_a and P_b, where the

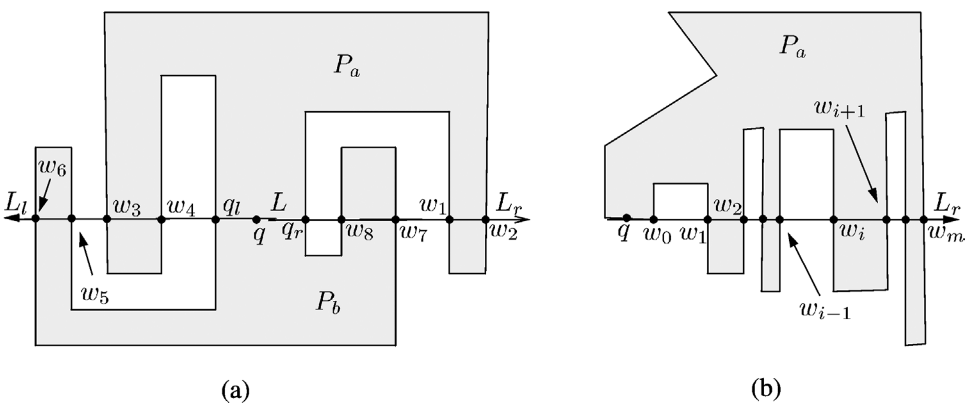


Fig. 3. (a) Four procedures identify one subsegment each on L. (b) All pairs of consecutive intersection points on L_r are of opposite type.

boundary of P_a (or P_b) consists of the segment $q_l q_r$ and the counterclockwise (respectively, clockwise) boundary from q_r to q_l. There are four types of subsegments of L that are lying inside P: the subsegments formed by pairs of intersection points of (i) L_r with $bd(P_a)$, (ii) L_l with $bd(P_a)$, (iii) L_r with $bd(P_b)$ and (iv) L_l with $bd(P_b)$. Our procedure identifies some of these subsegments such that after splitting P_a and P_b by adding these subsegments, the portion of P_a (or P_b), whose boundary contains $q_l q_r$, is above (respectively, below) L. Union of these two portions, one from P_a and another from P_b, form the polygon P_1 and it contains both q and $V(q)$.

The subsegments of type (i) on L_r can be identified by scanning $bd(P_a)$ in counterclockwise order from q_r to q_l. This procedure is denoted as $CCS(P_a, q_r, q_l, L_r)$. Analogously, procedures for identifying the subsegments of types (ii), (iii) and (iv) are denoted as $CS(P_a, q_l, q_r, L_l)$, $CS(P_b, q_r, q_l, L_r)$ and $CCS(P_b, q_l, q_r, L_l)$ respectively. For the simple polygon in Fig. 3(a), $CCS(P_a, q_r, q_l, L_r)$ identifies the subsegment $w_1 w_2$, $CS(P_a, q_l, q_r, L_l)$ identifies $w_3 w_4$, $CS(P_b, q_r, q_l, L_r)$ identifies $w_7 w_8$ and $CCS(P_b, q_l, q_r, L_l)$ identifies $w_5 w_6$. Since these procedures are analogous, we present here only the procedure $CCS(P_a, q_r, q_l, L_r)$.

As stated above, $CCS(P_a, q_r, q_l, L_r)$ scans $bd(P_a)$ in counterclockwise order from q_r to q_l and locate subsegments on L_r lying inside P_a. Let w be an intersection point of L_r with $bd(P_a)$. If the next counterclockwise vertex of w on $bd(P_a)$ is below (or above) L_r, then w is called a *downward* (respectively, *upward*) intersection point. Note that q_r is an upward intersection point by definition. If two intersection points are both downward or upward, they are called the *same type* of intersection points. Otherwise, they are called the *opposite type* of intersection points. In Fig. 3(a), (w_1, w_2) is a pair of opposite type as w_1 and w_2 are downward and upward intersection points respectively. We have the following properties on the pairs of intersection points of L_r with $bd(P_a)$.

Lemma 3.1. *Let u and w be two intersection points of L_r with $bd(P_a)$. If u and w are same type of intersection points, the segment uw does not lie inside P_a.*

Proof. Since u and w are same type of intersection points and P_a is a closed and bounded region, there are odd number of intersection points of $bd(P_a)$ with L_r that are lying on the segment uw. Hence, the segment uw does not lie inside P_a. □

Lemma 3.2. *Let u and w be two intersection points of L_r with $bd(P_a)$. If the segment uw lies inside P_a, then u and w are opposite type of intersection points.*

Proof. If u and w are the same type of intersection points, then the segment uw does not lie inside P_a by Lemma 3.1, a contradiction. □

Corollary 3.1. *If u is a downward* (*or upward*) *intersection point, w is an upward* (*respectively, downward*) *intersection point and uw lies inside P_a, then $u \in qw$* (*respectively, $w \in qu$*).

Lemma 3.3. *Let u and w be two intersection points of L_r with $bd(P_a)$. Assume that u and w are downward and upward intersection points respectively and $u \in qw$, or vice versa. If the segment uw does not lie inside P_a, then there exists another pair of intersection points (u', w') of opposite type lying on the segment uw.*

Proof. Since u and w are intersection points of opposite type and the segment uw does not lie inside P_a, there are even number of intersection points of $bd(P_a)$ with the segment uw excluding the points u and w. So, there exists at least a pair of intersection points (u', w') of opposite type lying on the segment uw. □

Above lemma suggests that in order to locate subsegments of L_r that are lying inside P_a, it is necessary to locate the pairs of intersection points of opposite type on L_r and then test whether the segment formed by any such pair contains another pair of opposite type. Let $W = (w_0, w_1, \ldots, w_m)$ be the order of intersection points of $bd(P_a)$ with L_r while $bd(P_a)$ is traversed in counterclockwise order starting from q_r, where $q_r = w_0$ (see Fig. 3(b)). Let w_{i-1} be a point of W such that for any two consecutive points w_k and w_{k+1} in $(w_0, w_1, \ldots, w_{i-1})$, (w_k, w_{k+1}) form a pair of opposite type and $w_k \in q w_{k+1}$. We say that points in $(w_0, w_1, \ldots, w_{i-1})$ are in the *proper order* up to w_{i-1}. It can be seen that if the points in W are in the proper order up to w_m, then the segments connecting alternate pairs of points in W lie inside P_a. Note that if there is winding in P_a, points in W are not in the proper order.

The procedure $CCS(P_a, q_r, q_l, L_r)$ tests whether the points in W are in the proper order starting from w_1. If it encounters a point that violates the proper order up to the last point tested, it discards some points of W and restores the proper order. In this process, the procedure $CCS(P_a, q_r, q_l, L_r)$ identify the subsegments of L_r that are lying inside P_a. In the following lemmas, we explicitly state the properties of the proper order on a subset of points in W.

Lemma 3.4. *Assume that the points in W are in the proper order up to w_{i-1}. If w_i preserves the order, then $w_i \notin qw_{i-1}$ and (w_{i-1}, w_i) is a pair of opposite type.*

Lemma 3.5. *Assume that the points in W are in the proper order up to w_{i-1}. If w_i violates the order, then $w_i \in qw_{i-1}$ or (w_{i-1}, w_i) is a pair of same type.*

Lemma 3.6. *Assume that the points in W are in the proper order up to w_{i-1}. If there is a point w_j of W lies on the segment $w_k w_{k+1}$, where $k < i-1$, then w_j is a subsequent point of w_{i-1} in W.*

Assume that the procedure $CCS(P_a, q_r, q_l, L_r)$ has tested points in W up to w_{i-1} and they are in the proper order (see Fig. 3(b)). It means that $w_0, w_2, w_4, \ldots, w_{i-1}$ are upward intersection points and $w_1, w_3, w_5, \ldots, w_{i-2}$ are downward intersection points. We also assume that the procedure has pushed alternate pairs of opposite type (w_1, w_2), $(w_3, w_4), \ldots, (w_{i-2}, w_{i-1})$ on the stack, where (w_{i-2}, w_{i-1}) is on the top of the stack. Note that the segments w_0w_1, $w_2w_3, \ldots, w_{i-3}w_{i-2}$ do not lie inside P_a. The procedure checks whether the next point w_i satisfies the order. We have the following cases.

Case 1. The point w_i is a downward intersection point and $w_i \notin qw_{i-1}$ (see Fig. 3(b)).
Case 2. The point w_i is a downward intersection point and $w_i \in qw_{i-1}$ (see Fig. 5(a)).
Case 3. The point w_i is an upward intersection point and $w_i \in qw_{i-1}$ (see Fig. 5(b)).
Case 4. The point w_i is an upward intersection point and $w_i \notin qw_{i-1}$ (see Fig. 7(b)).

Consider Case 1. Since w_i is a downward intersection point and $w_i \notin qw_{i-1}$, w_i is in the proper order by Lemma 3.4. The procedure checks whether (w_i, w_{i+1}) is the next pair of opposite type. If $w_i \in qw_{i+1}$, then (w_i, w_{i+1}) is the next pair (see Fig. 3(b)). If $w_{i+1} \in qw_i$ (see Fig. 4(a)), then w_{i+1} violates the proper order by Lemma 3.5. Scan W starting from w_{i+2} till a point w_k is found such that $w_i \in qw_k$. So, (w_i, w_k) is the next pair and points $(w_{i+1}, \ldots, w_{k-1})$ are removed. Without loss of generality, we assume that (w_i, w_{i+1}) is the next pair. If w_{i+1} is an upward intersection point (see Fig. 3(b)), then (w_i, w_{i+1}) is the next pair of opposite type, and the points in $(w_0, w_1, \ldots, w_i, w_{i+1})$ are in the proper order by Lemma 3.4. Therefore, (w_i, w_{i+1}) is pushed on the stack. Otherwise, (w_i, w_{i+1}) is the first pair of same type because both w_i and w_{i+1} are downward intersection points (see Fig. 4(b)). By Lemma 3.5, w_{i+1} has violated the proper order. It can be seen that the counterclockwise boundary of

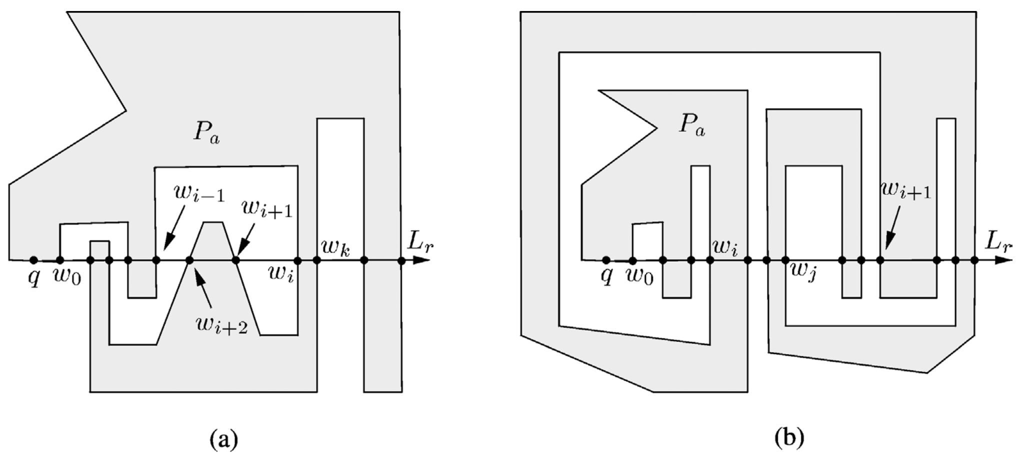


Fig. 4. (a) The points w_i and w_k form the next pair. (b) The points w_i and w_j form the next pair.

P_a from w_i to w_{i+1} (denoted as $bd(w_i, w_{i+1})$) has winded around q. Scan W starting from w_{i+2} till a point w_j is found such that $w_j \in w_i w_{i+1}$. Since w_j is an upward intersection point, (w_i, w_j) becomes the next pair of opposite type by Lemma 3.4. Remove all points of W that do not belong to the segment qw_{i+1} as L_r is now restricted to qw_{i+1}. The pair (w_i, w_j) is pushed on the stack. Note that if the segment $w_i w_j$ lies inside P_a, the winding in $bd(w_i, w_{i+1})$ can be removed from P_a by adding the segment $w_i w_j$ to P_a. Otherwise, there exists another pair of opposite type in W by Lemma 3.3 (see Fig. 4(b)) that lies on the segment $w_i w_j$, which will be detected subsequently as stated in Lemma 3.6.

Consider Case 2. Since w_i is a downward intersection point and $w_i \in qw_{i-1}$, w_i has violated the proper order by Lemma 3.5 (see Fig. 5(a)). It can be seen that w_i lies on a segment formed by a pair (say, (w_k, w_{k+1})) which is already in the stack. Pop the stack till (w_k, w_{k+1}) is on the top of the stack. We know from Lemma 3.3 that there exists another pair of opposite type in W that lies on the segment $w_k w_{k+1}$. Scan W from w_{i+1} till a point w_j is found such that $w_j \in w_k w_i$. Observe that w_j is an upward intersection point and (w_k, w_j) is a pair of opposite type. Hence, the points in $(w_0, w_1, \ldots, w_k, w_j)$ are in the proper order by Lemma 3.4. Pop (w_k, w_{k+1}) from the stack and push (w_k, w_j) on the stack.

Consider Case 3. Since w_i is an upward intersection point and $w_i \in qw_{i-1}$, w_i has violated the proper order by Lemma 3.5 (see Figs. 5(b) and 6(a)). It can be seen that w_i belongs to the subsegment of L_r whose corresponding pair is not in the stack. Scan W from w_i till two consecutive points $w_{j-1} \in qw_{i-1}$ and $w_j \in qw_{i-1}$ are found such that they are both downward intersection points (see Fig. 5(b)). Remove all points $(w_i, \ldots, w_{j-1})$ from W. Treating w_j as new w_i, Case 2 is executed to update the stack. If no such vertices w_{j-1} and w_j exist (see Fig. 6(a)), it means that $bd(w_i, w_m)$ has not intersected any segment formed by a pair in the stack and therefore, these segments are added to partition P_a. In the process, the stack becomes empty.

It can be seen that P_a still has winding in $bd(w_{i-1}, w_i)$ (see Fig. 6(a)) which is to be removed. The procedure $CCS(P_a, q_r, q_l, L_r)$ now locates the subsegments of qw_i (from w_i towards q) using the same stack that are lying inside P_a. Let $U = (u_0, u_1, \ldots, u_p)$ be the order of intersection points of $bd(w_i, q)$ with qw_i while $bd(P_a)$ is traversed in counterclockwise order starting from w_i, where $w_i = u_0$ (see Fig. 6(a)). Observe that any two consecutive points u_{k-1} and u_k in U are of opposite type though there may be winding in P_a. However, u_k may not always lie on qu_{k-1} for all k and therefore, the points in U may not be in the proper order in the direction from u_0 towards q. We have the following lemmas on the proper order of U, which are analogous to Lemmas 3.4, 3.5 and 3.6.

Lemma 3.7. *Assume that the points in U are in the proper order from u_0 to u_{k-1}. If u_k preserves the order, then $u_k \notin u_0 u_{k-1}$.*

Lemma 3.8. *Assume that the points in U are in the proper order from u_0 to u_{k-1}. If u_k violates the order, then $u_k \in u_0 u_{k-1}$.*

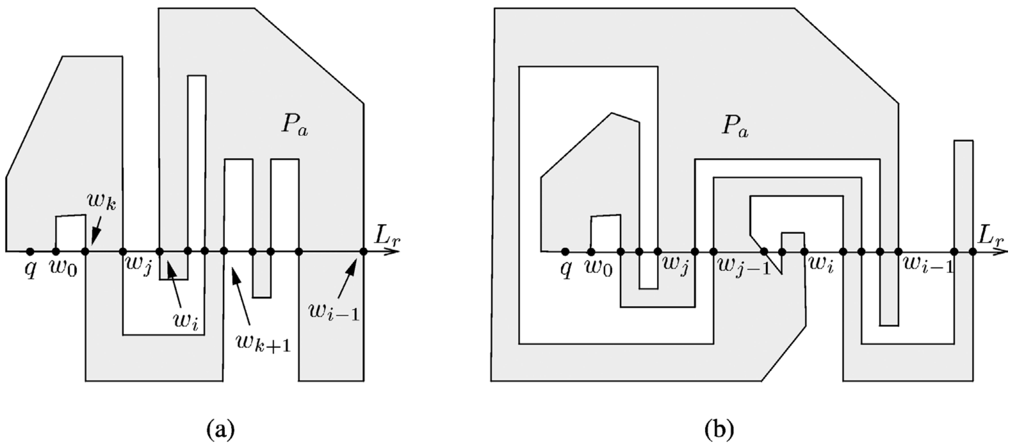
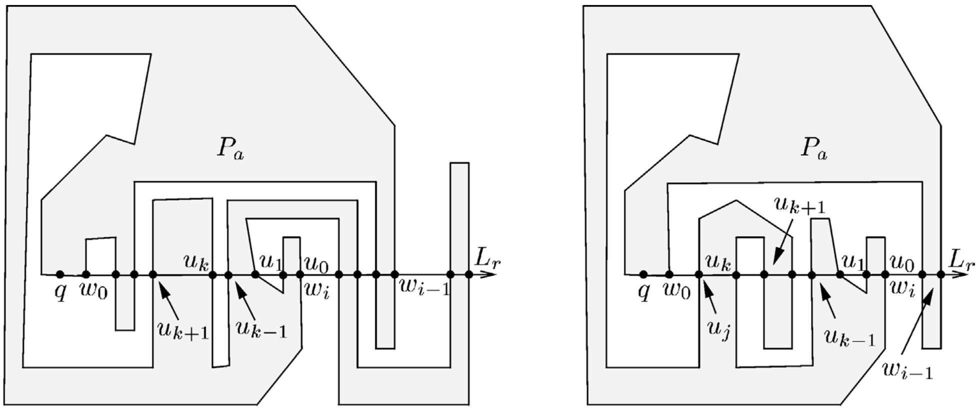
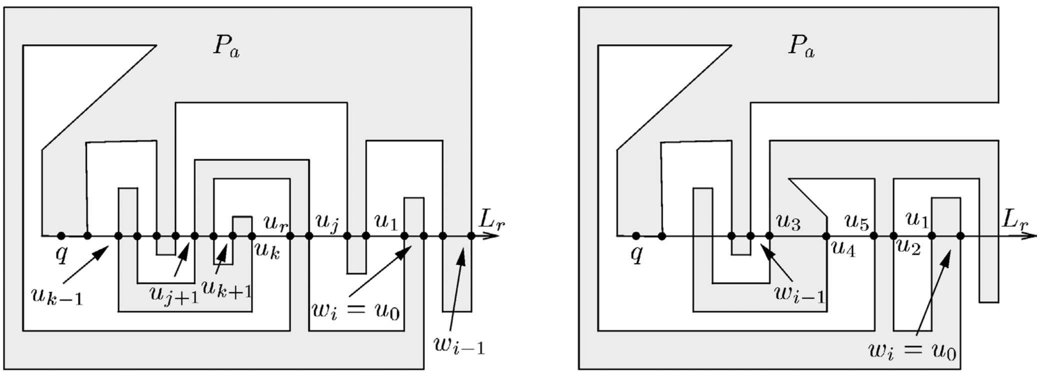


Fig. 5. (a) The downward intersection point w_i lies on the segment $w_k w_{k+1}$. (b) The upward intersection point w_i belongs to the segment whose corresponding pair is not in the stack.



(a) (b)

Fig. 6. (a) The next pair (u_k, u_{k+1}) is in the proper order. (b) The next pair in the proper order is (u_k, u_j).



(a) (b)

Fig. 7. (a) The next pair in the proper order is (u_j, u_r). (b) P_a is partitioned using the segments corresponding to the pairs (u_0, u_1) and (u_2, u_5).

Lemma 3.9. *Assume that the points in U are in the proper order from u_0 to u_{k-1}. If there is a point u_j of U lies on the segment $u_t u_{t+1}$, where $t < k - 1$, then u_j is a subsequent point of u_{k-1} in U.*

Assume that the procedure $CCS(P_a, q_r, q_l, L_r)$ has tested points in U up to u_{k-1} and they are in proper order (see Fig. 6(a)). We also assume that the procedure has pushed alternate pairs of opposite type (u_0, u_1), $(u_2, u_3), \ldots, (u_{k-2}, u_{k-1})$ on the stack. Recall that u_0 is an upward intersection point. If $u_k \notin u_0 u_{k-1}$, then the point u_k is in the proper order by Lemma 3.7. The procedure checks whether (u_k, u_{k+1}) is the next pair of opposite type. If $u_{k+1} \notin u_0 u_k$ (see Fig. 6(a)), then u_{k+1} is also in the proper order by Lemma 3.7. So, (u_k, u_{k+1}) is the next pair of opposite type and (u_k, u_{k+1}) is pushed on the stack. Otherwise, u_{k+1} belongs to $u_0 u_k$ (see Fig. 6(b)) and u_{k+1} has violated the proper order by Lemma 3.8. Scan U starting from u_{k+2} till a point u_j is found such that $u_j \notin u_0 u_k$. So, points in $(u_0, u_1, \ldots, u_k, u_j)$ are in the proper order by Lemma 3.7. Therefore, (u_k, u_j) is the next pair of opposite type and (u_k, u_j) is pushed on the stack. Consider the other situation when $u_k \in u_0 u_{k-1}$ (see Fig. 7(a)). So, u_k has violated the proper order by Lemma 3.8. It can be seen that u_k lies on the segment formed by a pair (say, (u_j, u_{j+1})) which is already in the stack. Pop the stack till (u_j, u_{j+1}) is on the top of the stack. We know from Lemma 3.3 that there exists another pair of opposite type in U that lies on the segment $u_j u_{j+1}$. Scan U from u_{k+1} till a point u_r is found such that $u_r \in u_j u_k$. Observe that u_r is a downward intersection point and points $(u_0, u_1, \ldots, u_j, u_r)$ are in the proper order by Lemma 3.7. Hence, (u_j, u_r) is a pair of opposite type. Pop (u_j, u_{j+1}) from the stack and push (u_j, u_r) on the stack.

Consider Case 4. Since w_i is an upward intersection point and $w_i \notin qw_{i-1}$, w_i has violated the proper order by Lemma 3.5 (see Fig. 7(b)). It can be seen that $bd(w_{i-1}, w_i)$ has winded around q. Let $U = (u_0, u_1, \ldots, u_p)$ be the order of intersection points of $bd(w_i, q)$ with the segment $w_{i-1}w_i$ while $bd(P_a)$ is traversed in counterclockwise order starting from w_i, where $w_i = u_0$. Pop the stack till the stack becomes empty. Using the same method stated above for U, locate all pairs of opposite type in U (from w_i towards w_{i-1}) that are in proper order. Add the segments corresponding to the pairs in the stack to partition P_a. After partition, the portion of P_a that contains q on its boundary becomes new P_a. Let W denote only the intersection points of the boundary of new P_a and qw_{i-1}. With new P_a and new W, $CCS(P_a, q_r, q_l, L_r)$ is executed again. Note that Case 4 cannot occur again and therefore, $CCS(P_a, q_r, q_l, L_r)$ terminates after the second round. In the following steps, we formally present the procedure $CCS(P_a, q_r, q_l, L_r)$.

Step 1. Traverse $bd(q_r, q_l)$ in counterclockwise order starting from q_r and compute the intersection points $W = (w_0, w_1, \ldots, w_m)$ of L_r with $bd(P_a)$ where $w_0 = q_r$; $h := 0$; $i := 1$;
Step 2. *If* w_i is a downward intersection point *and* $w_i \notin qw_h$ (see Case 1) *then*
  Step 2a. Assign $i + 1$ to k; *while* $w_k \in qw_i$, $k := k + 1$;
  Step 2b. *If* w_k is an upward intersection point *then begin* push (w_i, w_k) on the stack; $i := k + 1$ *end else begin* $j := k + 1$; *while* $w_j \notin w_i w_k$, $j := j + 1$; push (w_i, w_j) on the stack; $i := j + 1$ *end*;
  Step 2c. Assign $i - 1$ to h; *if* $i \neq m + 1$ *then goto* Step 2 *else goto* Step 10;
Step 3. *If* w_i is a downward intersection point *and* $w_i \in qw_h$ (see Case 2) *then*
  Step 3a. Let (w_k, w_r) denote the pair on the top of the stack. *While* $w_i \notin w_k w_r$ pop the stack; $j := i + 1$; *while* $w_j \notin w_k w_i$, $j := j + 1$; pop the stack and push (w_k, w_j) on the stack;
  Step 3b. Assign $j + 1$ to i; $h := i - 1$; *if* $i \neq m + 1$ *then goto* Step 2 *else goto* Step 10;
Step 4. *If* w_i is an upward intersection point *and* $w_i \in qw_h$ (see Case 3) *then*
  Step 4a. Assign $i + 1$ to j;
  Step 4b. *If* $j = m$ *then goto* Step 4d;
  Step 4c. *If* w_{j+1} and w_j are downward intersection points and both of them belong to qw_h then $i := j + 1$ and *goto* Step 3 *else* $j := j + 1$ *and goto* Step 4b;
  Step 4d. *While* stack is not empty, add the segment corresponding to the pair on the top of the stack to P_a and pop the stack;
  Step 4e. Traverse $bd(w_i, q_l)$ in counterclockwise order starting from w_i and locate the intersection points $U = (u_0, u_1, \ldots, u_p)$ of qw_i with $bd(w_i, q_l)$ where $u_0 = w_i$; *goto* Step 6;
Step 5. *If* w_i is an upward intersection point *and* $w_i \notin qw_h$ (see Case 4) *then*
  Step 5a. Traverse $bd(w_i, q_l)$ in counterclockwise order starting from w_i and compute the intersection points $U = (u_0, u_1, \ldots, u_p)$ of $w_h w_i$ with $bd(w_i, q_l)$ where $u_0 = w_i$;
  Step 5b. Clear the stack;
Step 6. Push (u_0, u_1) on the stack; $k := 2$;
Step 7. *If* $u_k \notin u_0 u_{k-1}$ *then begin* $j := k + 1$; *while* $u_j \in u_0 u_k$, $j := j + 1$; push (u_k, u_j) on the stack; $k := j + 1$; *goto* Step 9 *end*;
Step 8. *If* $u_k \in u_0 u_{k-1}$ *then*
  Step 8a. Let (u_j, u_t) denote the pair on the top of the stack. *While* $u_k \notin u_j u_t$, pop the stack; $r := k + 1$;
  Step 8b. *While* $u_r \notin u_j u_k$, $r := r + 1$; pop the stack and push (u_j, u_r) on the stack; $k := r + 1$;
Step 9. *If* $k \neq p + 1$ *then goto* Step 7;
Step 10. *While* stack is not empty, add the segment corresponding to the pair on the top of the stack to P_a and pop the stack;
Step 11. STOP.

Lemma 3.10. *The procedure $CCS(P_a, q_r, q_l, L_r)$ correctly removes winding from P_a.*

Proof. Consider any pair of intersection points (w_i, w_j) of W in the stack at the time of executing Step 4d or Step 10. We show that the segment $w_i w_j$ lies inside P_a. We know that (w_i, w_j) is pushed on the stack either in Step 2b or in Step 3a. It can be seen from Steps 2 and 3 that (i) w_i is a downward intersection point, (ii) w_j is an upward intersection point, (iii) $w_i \in qw_j$ and (iv) points of W belonging to the pairs in the stack are in the proper order up to w_j. So, the segment $w_i w_j$ lies inside P_a provided no subsequent point w_k of w_j in W belongs to $w_i w_j$. If such point w_k

exists, then Steps 3 and 4c ensure that (w_i, w_j) cannot remain on the stack. Hence, the segment $w_i w_j$ lie inside P_a. Analogous arguments show that any segment, which corresponds to a pair of intersection points of U in the stack at the time of executing Step 10, lies inside P_a.

Consider any point z on $bd(P_a)$ such that $bd(w_0, z)$ has winded around q. We show that the procedure $CCS(P_a, q_r, q_l, L_r)$ has removed z from P_a. Assume on the contrary that z is not removed from P_a by the procedure $CCS(P_a, q_r, q_l, L_r)$. So, there is a polygonal path from q to z such that the path does not intersect the segment corresponding to any pair in the stack at the time of executing Step 4d or Step 10. So, there exists another pair (v', v'') of intersection points of W or U such that v' and v'' are in the proper order along with the points of the pairs in the stack. It means that $CCS(P_a, q_r, q_l, L_r)$ has not considered all points of W and U, which is not possible. Hence, z is removed from P_a. $\square$

After P_a is modified by $CCS(P_a, q_r, q_l, L_r)$, P_a is further modified by $CS(P_a, q_l, q_r, L_l)$ and the new P_a forms the portion of P_1 above the line L. Similarly, after the execution of procedures $CS(P_b, q_r, q_l, L_r)$ and $CCS(P_b, q_l, q_r, L_l)$, the new P_b forms the portion of P_1 below L. So, the union of P_a and P_b gives the pruned polygon P_1. It can be seen that the pruning algorithm runs in $O(n)$ time. We state the result in the following theorem.

Theorem 3.1. *Given a point q inside an n-sided simple polygon P, a polygon $P_1 \subseteq P$ can be constructed in* $O(n)$ *time such that* (i) *P_1 contains both q and the visibility polygon of P from q, and* (ii) *the boundary of P_1 does not wind around q.*

4. Concluding remarks

In Section 2, we have discussed the need for pruning algorithm in the context of computing $V(q)$. Consider the problem of computing the weak visibility polygon $V(pq)$ of P from a given internal segment pq. A point $z \in P$ is said to be *weakly visible* from pq if z is visible from any point of pq. Draw the line L passing through p and q, and remove the winding of P using our pruning algorithm. It can be seen that the pruned polygon contains p, q and $V(pq)$. Therefore, the pruned polygon can be used as the input polygon to the algorithm of Guibas et al. [7] for compute $V(pq)$. We feel that such pruning can reduce the size of the input polygon significantly for polygons with a large number of vertices.

Acknowledgements

The authors wish to thank Chinmoy Dutta and Partha Goswami for suggesting improvements to original paper.

References

[1] T. Asano, S.K. Ghosh, T. Shermer, Visibility in the plane, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, North-Holland, Amsterdam, 2000, pp. 829–876.
[2] Ta. Asano, Te. Asano, L.J. Guibas, J. Hershberger, H. Imai, Visibility of disjoint polygons, Algorithmica 1 (1986) 49–63.
[3] Te. Asano, Efficient algorithms for finding the visibility polygons for a polygonal region with holes, Trans. IECE Japan E68 (1985) 557–559.
[4] L. Davis, M. Benedikt, Computational models of space: Isovists and isovist fields, Comput. Graph. Image Process. 11 (1979) 49–72.
[5] S.E. Dorward, A servey of object-space hidden surface removal, Internat. J. Comput. Geom. Appl. 4 (1994) 325–362.
[6] H. ElGindy, D. Avis, A linear algorithm for computing the visibility polygon from a point, J. Algorithms 2 (1981) 186–197.
[7] L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, R.E. Tarjan, Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons, Algorithmica 2 (1987) 209–233.
[8] P.J. Heffernan, J.S.B. Mitchell, An optimal algorithm for computing visibility in the plane, SIAM J. Comput. 24 (1) (1995) 184–201.
[9] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, R.E. Tarjan, Sorting Jordan sequences in linear time using level-linked search trees, Inform. Control 68 (1986) 170–184.
[10] B. Joe, On the correctness of a linear-time visibility polygon algorithm, International J. Comput. Math. 32 (1990) 155–172.
[11] B. Joe, R.B. Simpson, Corrections to Lee's visibility polygon algorithm, BIT 27 (1987) 458–473.
[12] D.T. Lee, Visibility of a simple polygon, Comput. Vis. Graph. Image Process. 22 (1983) 207–221.
[13] S. Suri, J. O'Rourke, Worst-case optimal algorithms for constructing visibility polygons with holes, in: Proceedings of the 2nd Annual ACM Symposium on Computational Geometry, 1986, pp. 14–23.