LR-Visibility in Polygons

(Extended Abstract)

Gautam Das Paul J. Heffernan
Giri Narasimhan
Dept. of Mathematical Sciences, Memphis State University, Memphis, TN 38152

Abstract

We give a linear-time algorithm which, for a simple polygon P, computes all pairs of points s and t on P that admit LR-visibility. The points s and t partition P into two subchains, and we say that P is LR-visible with respect to s and t if each point of P is visible from some point of the other subchain.

1 Introduction

We consider here the *LR-visibility* problem for simple polygons. Any two points s and t of a simple polygon P partition P into two subchains, which we call L and R, for left and right chains. The LR-visibility question asks whether each point of L can see a point of R, and each point of R can see a point of L. If the answer is yes, we say that P is *LR-visible with respect to s and t*. Figure 1 shows a polygon that is LR-visible for various point pairs s and t, and Figure 2 one that is not LR-visible for any pair s and t.

We state four versions of the LR-visibility problem for a polygon P:

1. determine whether a given pair s and t admits LR-visibility;
2. determine whether there exists a pair s and t which admits LR-visibility;
3. return a pair s and t which admits LR-visibility, if indeed such a pair exists;
4. return *all* pairs s and t which admit LR-visibility.

Version (4) is the strongest, and an algorithm for it also solves the first three versions. In this paper we solve to optimality the strongest version: we give a $\Theta(n)$-time algorithm that computes all pairs of points s and t that admit LR-visibility for a simple polygon P with n vertices. The output is in the form of $O(n)$ pairs of subchains S_i and T_i, such that any pair of points $s \in S_i$ and $t \in T_i$ is a valid pair s and t. In Figure 1, the output subchains $(S_1, T_1), \ldots, (S_4, T_4)$ are shown. Version (4) has been solved previously in [TL] in $O(n \log n)$ time.

The question of LR-visibility falls in the larger area of weak visibility in polygons, which has received much attention by researchers. To say that two sets are weakly-visible means that *every* point in either set is visible from *some* point in the other set. A simple polygon P is weakly-visible from an edge e if e and $P \setminus e$ are weakly-visible. A weakly-visible chord c of P is one such that c and P are weakly-visible. A polygon is LR-visible if its left chain L and right chain R are weakly-visible. Weak-visibility of a polygon from an edge was first studied in [AT], and Sack and Suri [SS] subsequently gave a linear-time algorithm which computes all weakly-visible edges of a simple polygon.

This paper is of interest not only because we

present an optimal result for an intriguing problem in polygonal visibility, but also on account of the techniques we employ, and because of the relationship between LR-visibility and other problems in polygonal visibility, such as the weakly-visible chord and the two-guard problems. While the two-guard problem has many formulations, we will state just one for the sake of illustration: a polygon P is walkable from point s to point t if one "guard" can traverse the left chain L and the other the right chain R from s to t while always remaining co-visible. Other formulations require the guards to move monotonically or that one guard traverses from t to s. LR-visibility is a subproblem of two-guard, in the sense that a polygon must be LR-visible with respect to s and t in order to be walkable for that pair. LR-visibility is also a subproblem of weakly-visible chord, for it can be shown that two points s and t of P are the endpoints of a weakly-visible chord of P if and only if $\overline{st}$ is a chord of P and P is LR-visible with respect to s and t.

As for LR-visibility, the four problem versions listed above, namely testing a fixed pair s and t, determining if a pair s and t exists, and finding one or all pairs s and t, exist for the weakly-visible chord problem and for each formulation of the two-guard problem. The authors have recently developed a linear-time algorithm which computes all weakly-visible chords of a simple polygon (problem version (4)) [DHN], where the result of the current paper is used as a subprocedure. The previous best result [K] determines whether there exists a weakly-visible chord and if so constructs one (versions (2) and (3)) in $O(n \log n)$ time. For the two-guard problem, currently there exist optimal linear-time algorithms for various formulations for fixed s and t (version (1)) [H], and $O(n \log n)$-time algorithms which find all pairs s and t (version (4)) for various formulations [TL]. The authors are currently working to develop optimal solutions for the all-pairs version (version (4)) of various formulations of the two-guard problem, and we feel that the present work is an important step towards this goal.

2 Preliminaries

We define notation for this paper. A *polygonal chain* in the plane is a concatenation of line segments. The endpoints of the segments are called *vertices*, and the segments themselves are *edges*. If the segments intersect only at the endpoints of adjacent segments, then the chain is *simple*, and if a polygonal chain is closed we call it a *polygon*. In this paper, we deal with a simple polygon P, and its interior, $int(P)$. Two points $x, y \in P$ are *visible* (or *co-visible*) if $\overline{xy} \subset P \cup int(P)$. We assume that the input is in general position, which means that no three vertices are collinear, and no three lines defined by edges intersect in a common point.

If x and y are points of P, then $P_{CW}(x, y)$ ($P_{CCW}(x, y)$) is the subchain obtained by traversing P clockwise (counterclockwise) from x to y.

The *ray shot* from a vertex v in direction d consists of "shooting" a "bullet" from v in direction d which travels until it hits a point of P. Formally, if $\vec{r}$ is the ray rooted at v in direction d, then the *hit point* of this ray shot is the point of $(P \setminus \{v\}) \cap \vec{r}$ closest to v. Each reflex vertex defines two special ray shots as follows. Let v be a reflex vertex and v'' the vertex adjacent to v in the clockwise direction. Then the ray shot from v in the direction from v'' to v is called the *clockwise ray shot* of v. If v' is the hit point of the clockwise ray shot, then the subchain $P_{CW}(v, v')$ is the *clockwise component* of v (see Figure 3). Counterclockwise ray shots and components are defined in the same way. A component is *redundant* if it is a superset of another component.

As noted in [IK], the family of components completely determines LR-visibility of P, since a pair of points s and t admits LR-visibility if and only if each component of P contains either s or t. The definition of redundant gives the following.

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

LR-visibility can be reduced to a problem

known as *two-cut*. Given a circle and a collection of (closed) arcs on the circle, we say that two points of the circle form a two-cut if every arc of the collection contains at least one of the two points. To reduce LR-visibility for a polygon P to an instance of two-cut, we parameterize both P and a circle C, and map each point of P to its corresponding point on C. This maps a component of P to an arc of C. If we map each non-redundant component of P to its corresponding arc of C, then the problem of finding points s and t on P that admit LR-visibility is equivalent to finding a two-cut on C. Note that the arcs of C are non-redundant in the sense that their beginning and ending points appear on C in the same order.

In [TL], an $O(n \log n)$-time algorithm is given which computes all two-cuts for a collection of n non-redundant arcs which are given in sorted order. It is not difficult to modify this algorithm to run in linear time. The output requires $O(n)$ space, because it is in the form of $O(n)$ pairs of intervals, such that any pair of points on a pair of intervals is a two-cut. A polygon P with n vertices has $O(n)$ components. This means that to solve the LR-visibility problem, it suffices to give a linear-time method to construct in sorted order all non-redundant components. We address this problem in the next section.

3 Non-redundant components

We discuss here our method for constructing all non-redundant components of a polygon P. The main tool is a procedure which produces a superset of all non-redundant clockwise components. A symmetric procedure does the same for counterclockwise components. This yields a superset of all non-redundant components, from which the non-redundant components in sorted order can be extracted, as we will now show.

Suppose we have a set of clockwise components which contains all the non-redundant ones. As we traverse P in clockwise order, we encounter a beginning point and an ending point of each component. Since the beginning points are vertices of P, they can be bucket-sorted in linear time. Suppose we traverse P twice counterclockwise. Each time we encounter a beginning point, we compare the ending point of the component to the ending point of the previous component; if the current component contains the previous component, then the current component is redundant and therefore is deleted from the list of components. We must traverse P twice since one of the first components considered may be redundant with respect to one of the last ones. After an analogous procedure is performed for counterclockwise components, we have two lists of components, each in sorted order, which can be merged and pruned of redundant components in linear time to obtain a sorted list of all non-redundant components.

We now turn our attention to the problem of computing a collection of clockwise components that contains all non-redundant ones. The discussion of this algorithm in this extended abstract is incomplete, but we attempt to express the major ideas behind it. Throughout this section, if v is a reflex vertex, then v' represents the hit point of the clockwise ray shot from v, so that $P_{CW}(v, v')$ is the clockwise component from v.

The outline of the procedure is as follows. We fix a vertex x of P. We traverse P counterclockwise from x, computing clockwise components as we go. Each time we encounter a reflex vertex v, we determine whether the clockwise component of v is redundant with respect to any of the previously computed components. Let v represent the current reflex vertex and $P_{CW}(w, w')$ the most recently computed clockwise component. To process v, we perform two steps:

1. we determine whether v' lies on $P_{CW}(v, w')$ (as in Figure 4(a)) or $P_{CCW}(v, w')$ (as in Figure 4(b));

2. if v' lies on $P_{CW}(v, w')$ then we compute v'.

Because step (2) is performed only for certain components, the hit points v' that are actually computed by step (2) occur in a counterclockwise order on P (although they may wrap around P twice).

We show that the components constructed by the above procedure include all non-redundant

clockwise components. It suffices to show that the clockwise component from the current reflex vertex v is redundant if its hit point v' lies on $P_{CCW}(v, w')$. Consider the relationship between the clockwise components $P_{CW}(v, v')$ and $P_{CW}(w, w')$. If v' lies on $P_{CCW}(v, w')$ (as in Figure 4(b)), then $P_{CW}(v, v')$ contains the component $P_{CW}(w, w')$, implying that the clockwise component from v is redundant.

There are two reasons why the above procedure may compute a component that is later found to be redundant. One is that a clockwise component may contain a counterclockwise component, or vice versa. The other is that the first clockwise components computed by the counterclockwise traversal of P may contain the last component computed.

Having seen that the procedure is correct, we now proceed to discuss how it is that steps (1) and (2) can be performed in $O(n)$ time. Constructing the non-redundant components with standard ray shooting techniques would yield a time bound of $O(n \log n)$, since each shot requires $O(\log n)$ time [CG]. By strategically choosing not to construct certain (redundant) components, our algorithm is able to perform faster. The key observation, noted earlier, is that the hit points of the constructed components appear in counterclockwise order. Thus, while the traversal for reflex vertices v is going on, a simultaneous traversal for (constructed) hit points v' is also taking place. The fact that this second traversal proceeds monotonically around P is crucial to the linear run-time.

The manner in which steps (1) and (2) are actually performed in $O(n)$ time is a matter of great detail. We will try to convey a few of the main ideas that are used in the methods of execution.

The algorithm of this paper resembles those of [H] both in spirit and in many of the techniques. Our techniques must be stronger, because we compute all pairs s and t, while [H] deals with a fixed pair. A key question in both papers is asking where a ray shot from a vertex y hits. If y and z are points of P, let $FE(y, z)$ be the edge of the shortest path from y to z incident to y (the *first edge*), and let $dFE(y, z)$ be the direction of $FE(y, z)$. A ray shot from y has a direction that is either "left" or "right" of $dFE(y, z)$. A version of the lemma below is given in [H] (see Figure 5).

Lemma 2 *A ray shot from y hits $P_{CW}(y, z)$ if and only if the shot is "left" of $dFE(y, z)$ (equivalently, the shot hits $P_{CCW}(y, z)$ if and only if the shot is "right" of $dFE(y, z)$).*

A *shortest path tree* from a point x of P, denoted $SPT(x)$, is the union of all shortest paths from x to a vertex of P. Such a structure can be constructed for x in $O(n)$ time [GHLST] (given a triangulation of P, which can be constructed in $O(n)$ time [C]), and $SPT(x)$ can be used to provide each point of P with constant-time access to the first edge of its shortest path to x. By Lemma 2, this means that given a ray shot from a vertex y, we can determine in constant time whether the hit point lies on $P_{CW}(y, x)$ or $P_{CCW}(y, x)$, if x has been preprocessed. In the clockwise component algorithm, we compute $SPT(x)$ from the fixed point x. We also compute shortest path trees from other vertices at various stages of the algorithm, both with respect to the entire polygon P and within certain subpolygons. These structures are used in the efficient execution of step (1), where we must determine (in amortized constant time) where a hit point lies on P with relation to two other points.

The first edges of shortest paths are also used to answer the visibility question of step (2), which asks that a hit point v' actually be returned. Imagine two points a and b of P, such that $\overline{ab}$ is a chord, and suppose that we have constructed $SPT(a)$ and $SPT(b)$. Let y and z be points of P on opposite sides of the chord $\overline{ab}$ (e.g. y is on $P_{CW}(a, b)$ and z on $P_{CCW}(a, b)$). Define a *cone* around y, that is defined by the two rays rooted at y and in the directions $dFE(x, a)$ and $dFE(x, b)$. By Lemma 2, a ray shot from y crosses $\overline{ab}$ if and only if the direction of the ray shot lies in (the interior of) the cone of y. If we define a cone of z in the same manner, we see that y and z are visible if and only if each is in (the interior of) the cone of the other. Step (2) consists of finding a hit point v', and it is

performed basically by traversing counterclockwise from a starting point, where we know that v' lies in the counterclockwise direction from the starting point (a condition guaranteed by an inductive property and by the answer returned by step (1)). The algorithm dynamically maintains a chord $\overline{ab}$ such that v and v' lie on opposite sides of $\overline{ab}$ whenever step (2) is called (in [H], a chord $\overline{ab}$ is used, but it remains constant throughout a procedure). When we encounter a point z that intersects the ray rooted at v, we determine whether z and v are visible; if they are we have found our hit point v', and if not we continue our traversal in search of v'.

Another observation is worth stating: a polygon P that is LR-visible for some pair of points s and t cannot have three disjoint components. This is true because this situation results in three disjoint arcs in the corresponding two-cut problem. The observation is important for our algorithm because the discovery of a pair of disjoint components is a special case that results in some extra work—it can be roughly thought of as having to "start over." However, the observation guarantees that this special case can occur only a constant number of times in an LR-visible polygon; if it occurs too often the algorithm halts and answers that P is not LR-visible.

References

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

[C] B. Chazelle, "Triangulating a simple polygon in linear time," *Discrete and Computational Geometry*, **6** (1991), pp. 485-524.

[CG] B. Chazelle and L. Guibas, "Visibility and intersection problems in plane geometry," *Discrete and Computational Geometry*, **4** (1989), pp. 551-581.

[DHN] G. Das, P.J. Heffernan, and G. Narasimhan, "Finding all weakly-visible chords of a polygon in linear time," manuscript, 1993.

[GHLST] 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** (1987), pp. 209-233.

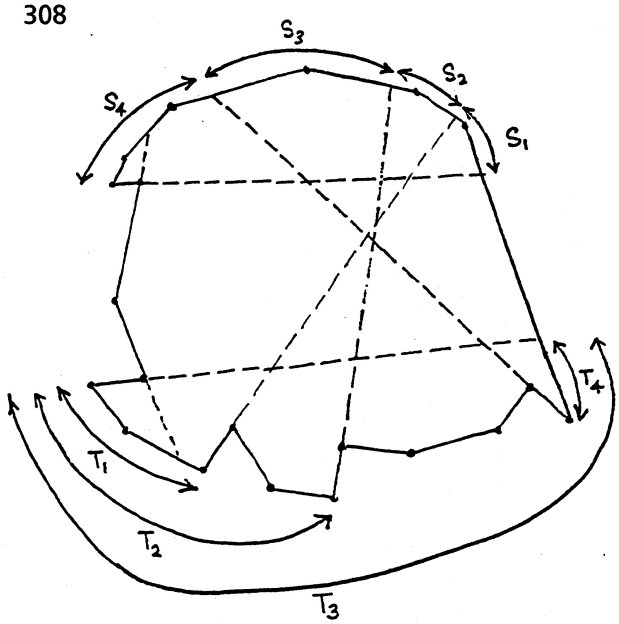
[H] P.J. Heffernan, "An optimal algorithm for the two-guard problem," to appear in *Proc. 9th Annual ACM Symp. on Computational Geometry*, 1993.

[IK] C. Icking and R. Klein, "The two guards problem," *Proc. 7th Annual ACM Symp. on Computational Geometry*, 1991, pp. 166-175.

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

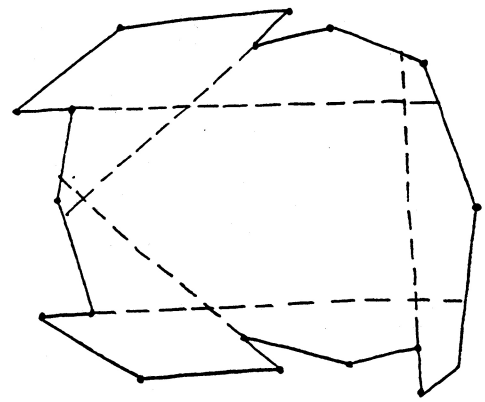
[SS] J.-R. Sack and S. Suri, "An optimal algorithm for detecting weak visibility," *IEEE Transactions on Computers*, **39** (1990), pp. 1213-1219.

[TL] L.H. Tseng and D.T. Lee, "Two-guard walkability of simple polygons," manuscript, 1993.



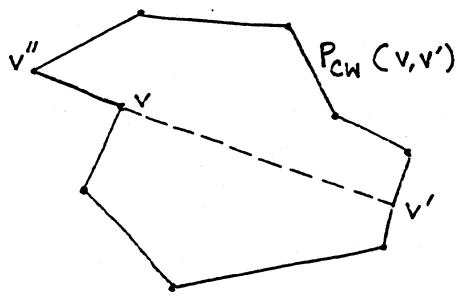
A LR-Visible Polygon

FIGURE 1



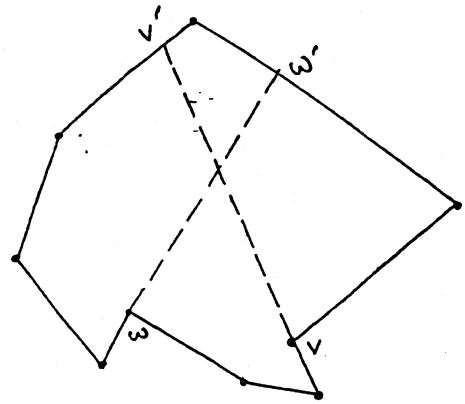
A Polygon which is not LR-Visible

FIGURE 2

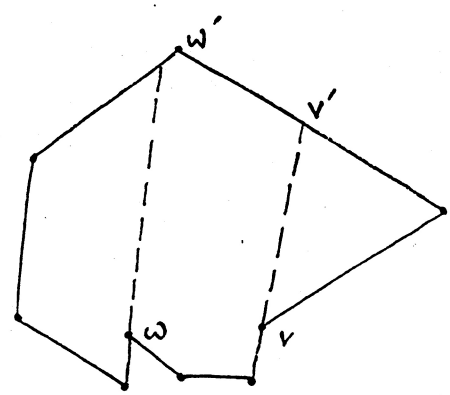


A Clockwise Component

FIGURE 3



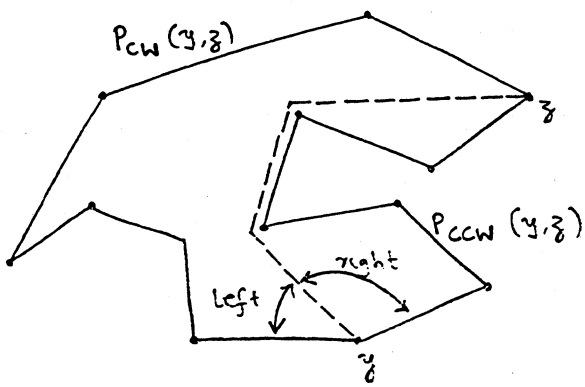
(a)



(b)

Computing Clockwise Components

FIGURE 4



Lemma 2

FIGURE 5