FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master's Thesis:
Constant-Workspace Algorithms for Visibility Problems in the Plane

Mikkel Abrahamsen

Department of Computer Science

Supervisor: Jyrki Katajainen

12th of June, 2013

Abstract. In the constant-workspace model, the input is given as a read-only array which allows random access and the output is to be produced on a write-only array as a stream. In addition to that, only a constant number of variables are available, independent on the size of the input. Most ordinary algorithms for geometric problems make heavy use on the construction of smart data structures such as doubly-linked lists, heaps, and search trees which enable fast processing. In the constant-workspace model such data structures are not available due to the small amount of memory. Instead we need to access the input repeatedly. We try to minimize the number of accesses in order to make the algorithms as efficient as possible.

In this thesis, we present new algorithms for visibility problems in the plane using constant workspace. We devise an $O(n^2)$-time algorithm computing the circular visibility region of a polygon with n vertices from a given point within the polygon. Next, we present an $O(n)$-time algorithm to compute the visible part of one edge from another edge in a polygon. Using that algorithm, we describe an algorithm computing the weak visibility polygon from a segment in $O(mn)$ time, where m is the number of vertices of the weak visibility polygon. Finally, we show how the computation of weak visibility polygons makes it possible to compute a minimum link path between two points in a polygon in $O(n^2)$ time.

Acknowledgements

I would like to thank Jyrki Katajainen for suggesting numerous improvements on the thesis and for being extraordinarily enthusiastic and encouraging in all of our meetings. Thanks to Kristian Nygaard Mortensen for suggesting the investigation of parabolic visibility in Section 5.3.

Table of contents

1. Introduction

A *visibility problem* is the task of computing what is visible to some observer under given circumstances, or a problem derived from such a problem, for instance determining the circumstances where the visibility of the observer satisfies some properties. The core of a visibility problem is to find the principles determining what is visible and what is not, and to efficiently use those principles to compute the visible parts of the surroundings of the observer. A real-world observer does not have to bother about such principles, as it is indeed solved by nature what he can see and what he cannot. It is clear that visibility problems are important in many practical applications, for instance in computer graphics, computer-aided design, and robotics. Visibility problems belong to the field of computational geometry and there have been made extensive research on a plethora of different problems. We are only concerned with two-dimensional visibility problems. The book by Gosh [15] gives a good overview of the most important problems and results in the area. The perhaps most famous visibility problem is the *art gallery problem*. In its simplest form, it is the task of placing a minimum number of stationary guards in a room with the shape of a simple polygon so that each point in the room can be seen by at least one guard. The book by O'Rourke [21] is a good introduction to a lot of variants of the problem.

1.1. The constant-workspace random-access model

Most of the algorithms developed for visibility problems are for the random-access model with $\Omega(n)$ variables available, where n is the size of the input. Here, we exclusively use the *constant-workspace random-access model* (or, for brevity, the *constant-workspace model*). In this model, the input is stored in a read-only array, but with random access, so that we can look up any value in constant time given its index. The output is to be produced on a write-only array in a stream-like manner, so that the storage cells of the array receive their values in the order of the array. In addition to that, a workspace consisting of only a constant number of variables is available—independent on the size of the input, though the constant can be arbitrarily large. Many problems become much more difficult to solve since it is extremely limited what data structures one can make with $O(1)$ variables. Ordinary algorithms often construct linked lists, heaps, and search trees which facilitate fast processing. Such data structures are not available in the constant-workspace model. Instead, we need to be extremely careful to make a minimal number of accesses to the input.

We assume that each variable can contain a pointer to a value in the input or a real number. We are aware that this assumption can be misused, since arbitrarily much information can be coded in the digits of one real number. Even though we cannot rigorously define when an algorithm makes illegal use of the model, we are sure that the algorithms in this thesis are sound in that sense. We assume that we can make comparisons between numbers and have access the elementary operations addition, subtraction, multiplication, and division in constant time. In Chapter 5, we also need the square-root function in order to do various computations involving circles and circular arcs. Those operations enable us to perform all elementary computations on points, lines, and line segments such as finding the intersection point between two lines, deciding if a point is to the right or to the left of an oriented line etc., see for instance the book by Cormen et al. [12, Chapter 33].

When the result of some computation consists of more than $O(1)$ values, we report the result using several **report** statements, where the reported values are to be written to the output array in the order in which they are reported. Otherwise, we just use one **return** statement.

1.2. Motivation

Constant-workspace algorithms are known to complexity theorists as log-space algorithms, since each variable takes up $\Theta(\log n)$ bits, see for instance the book by Savage [23, Section 3.9.3]. We prefer the term "constant workspace" since it seems more intuitive to think of the number of available variables than the number of available bits. With less than $\lceil \log(n+1) \rceil$ bits in the workspace, one cannot make any computation involving all of the input, since we cannot even store an index of one of the input values. Therefore, the constant-workspace model is the random-access model with the asymptotically smallest amount of memory that enables one to make a computation where each value in the input is considered at some point during the computation. The model may seem unnecessarily restrictive and irrelevant since most of today's computers have memory in abundance. However, we think it is a healthy and interesting exercise to investigate what one can do with a minimum of memory. As we shall see in Chapter 2, it is indeed sometimes possible to make a constant-workspace algorithm solving a problem in optimal time when the best previously known algorithm uses $\Omega(n)$ variables. The constant-workspace model is in some sense a computer analogy of dementia—the computer can only remember its own program and a fixed number of variables, there is no room for new "impressions". We think that makes the model interesting in its own right.

It is common to consider the *space-time tradeoff* of a problem. That means to establish upper and lower bounds of the space-time product ST, where S is the amount of space allowed and T is the required computation time given S space [23, Chapter 10]. For many problems, bounds of the space-time product independent on S for a whole range of different values of

S have been established. For instance, Beame [8] showed that the problem of sorting n numbers using comparisons requires $ST = \Omega(n^2)$ for $S = \Omega(\log n)$, where S is measured in bits. Subsequently, Pagter and Rauhe [22] showed that $ST = O(n^2)$ is also an upper bound whenever $c_1 \log n \leq S \leq c_2 n/\log n$ for some constants c_1 and c_2. Notice that the range of S is maximal since any comparison-based sorting algorithm uses $\Omega(n \log n)$ time [19, Section 5.3.1]. Recently, Asano et al. [3] gave a simpler memory-adjustable algorithm reaching the same bound.

In order to understand the space-time tradeoff for some problem, we think it is natural to start by considering algorithms in the extreme cases. For the problems in this thesis, efficient algorithms using $\Theta(n)$ variables have long been known, but the other extreme where we only use $O(1)$ variables is still to be investigated. In the rest of the thesis, the space complexities always refer to the number of variables used, not the number of bits.

1.3. Related work

One of the best-known constant-workspace algorithms for a geometric problem is *Jarvis' march* [17] for computing the convex hull of n points in the plane in $O(hn)$ time, where h is the number of points on the hull. Recently, Asano et al. [2] and Asano et al. [4] gave constant-workspace algorithms solving many elementary tasks in planar computational geometry like reporting triangulations of point sets and polygons, trapezoidations of polygons, finding Euclidean minimum spanning trees of point sets, and computing the shortest path between two points in a polygon. We give more detailed references to related work in each of the chapters.

1.4. Overview of the rest of the thesis

In this thesis, we address some planar visibility problems which were previously unsolved in the constant-workspace model.

In Chapters 2, 3, and 4, we consider "ordinary" visibility, where one can look from a point p to a point q if no object is on the line segment pq. In Chapter 2, we give a new algorithm to compute the visible part of one edge from another edge of a polygon. Imagine that a guard patrols back and forth along a wall in a very complex art gallery. Our algorithm can compute the part of another wall that the guard can see from his wall. The algorithm runs in $O(n)$ time and is thus optimal and superior to the already known algorithms for this problem, since they use $\Omega(n)$ space. We made a paper about the algorithm for The 25th Canadian Conference on Computational Geometry, which was accepted. The paper is included in Appendix A.

The *weak visibility polygon* from some segment in a simple polygon is all the points in the polygon which can be seen from some point on the segment, i.e. the region visible to a guard patrolling along the segment. The edge-to-edge-visibility algorithm of Chapter 2 enables us to devise an $O(mn)$-time algorithm to compute the weak visibility polygon in Chapter 3, where m is the number of vertices of the weak visibility polygon.

Imagine that one has to place a minimum number of mirrors in a complex room that enables an observer in a point s to see another point t. The positions of the mirrors define a *minimum link path*, which is a polygonal path contained in the polygon connecting the two points such that the number of line segments of the path is minimal. The algorithm for weak visibility polygons of Chapter 3 leads to an $O(n^2)$-time algorithm to compute a minimum link path given in Chapter 4.

In Chapter 5, we consider *circular visibility.* Imagine that light moves along circular arcs of all radii instead of straight lines, so that an object is visible to the observer if a circular arc from the object to the observer exists that does not interfere with other objects. How would the world look like? We answer that question partially by giving a constant-workspace algorithm to compute the circularly visible part of a polygon from a point. The algorithm runs in $O(n^2)$ time, where n is the number of vertices of the polygon. In Section 5.3, we round off the thesis by sketching how the methods used to deal with circular visiblity can be used to compute what one can hit with a canon in a two-dimensional vertical world with a boundary shaped as a simple polygon. In Chapter 6 we sum up the results and give some open problems and ideas for future work.

Whenever we do not state the origin of a result, we believe it to be our own original work.

1.5. Notation and some basic definitions

See Figure 1(a). Given two points a and b in the plane, the line segment with endpoints a and b is written ab. Both endpoints are included in segment ab. If s is a line segment, the line containing s which is inifinite in both directions is written $\overleftrightarrow{s}$. The *half line* $\overrightarrow{ab}$ is a line infinite in one direction, starting at a and passing through b. The *right half plane* RHP(ab) is the closed half plane with boundary $\overleftrightarrow{ab}$ lying to the right of ab. The *left half plane* LHP(ab) is just RHP(ba).

Let $V = \{v_0, v_1, \ldots, v_{n-1}\}$ be $n \geq 3$ points in the plane and let $E = \{v_0v_1, v_1v_2, \ldots, v_{n-1}v_0\}$ be the set of segments defined by neighbouring points in V. Assume that (i) no two points in V are equal, (ii) no three points in V are collinear, and (iii) if two segments in E do not share an endpoint, they are disjoint. In that case, V is the vertices and E is the edges of a *simple polygon* $\mathcal{P}$. The edges in E makes a simple, closed curve, which is the *boundary* of $\mathcal{P}$. The boundary is also written $\partial\mathcal{P}$. $\mathcal{P}$ is all the points on the boundary and the points that the boundary encloses in its interior. We might write *polygon* instead of *simple polygon* since we only consider simple polygons in this thesis. The requirement (ii) to the vertex set V is that no three vertices of $\mathcal{P}$ are collinear, which is a general position assumption. The assumption makes definitions and algorithms more elegant, but can be

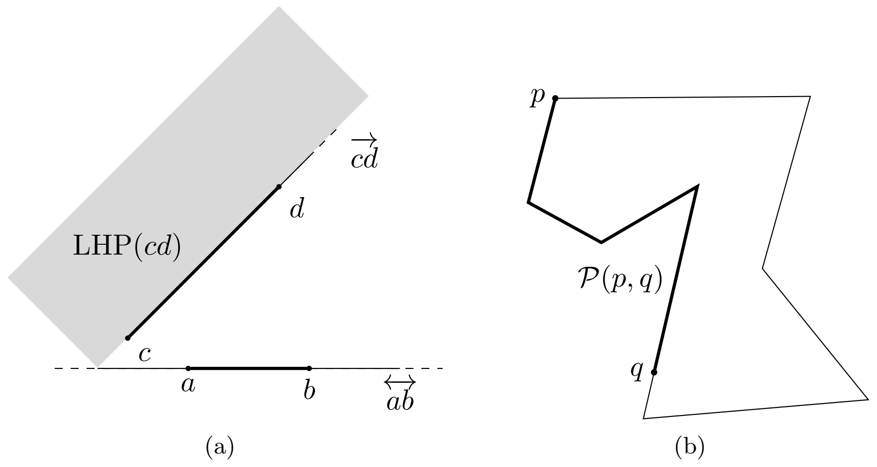


Figure 1. (a) A segment ab (thick), and the doubly infinite line $\overleftrightarrow{ab}$ through a and b. The half line $\overrightarrow{cd}$ and the half-plane LHP(cd) are also shown. Dashed lines indicate that the lines continue to the infinite. (b) A polygon with points p and q on its boundary and $\mathcal{P}(p,q)$ drawn with thick lines.

removed by introducing some special cases where it is used. The vertices of a polygon are always given in CCW (counterclockwise) order, so that the interior of $\mathcal{P}$ is on the left-hand side when traveling around $\partial\mathcal{P}$ in the order of the vertices. Vertices with indices congruent modulo n are the same, so that $v_0 = v_n$.

For two points p and q on $\partial\mathcal{P}$, we let $\mathcal{P}(p,q)$ be the set of points on $\partial\mathcal{P}$ we meet when traversing $\partial\mathcal{P}$ CCW from p to q, both included, see Figure 1(b). A *chain* of $\mathcal{P}$ is such a set $\mathcal{P}(p,q)$ for some $p,q \in \partial\mathcal{P}$. We see that $\mathcal{P}(p,q) \cup \mathcal{P}(q,p) = \partial\mathcal{P}$. A vertex v is a *reflex vertex* or a *concave vertex* if the interior angle at v is more than 180°. Otherwise, it is a *convex vertex.* A *chord* ab of $\mathcal{P}$ is a line segment contained in $\mathcal{P}$ such that the endpoints a and b are on $\partial\mathcal{P}$.

2. Visibility Between Two Edges

Consider the edge v_0v_1 of a simple polygon $\mathcal{P}$. A *beam* emanating from v_0v_1 is a segment pq where $p \in v_0v_1$ and pq is contained in $\mathcal{P}$. We also say that pq is emanating from p. A point q in $\mathcal{P}$ is *visible* from v_0v_1 if there exists a beam pq emanating from v_0v_1. A *right support* is a reflex vertex v of $\mathcal{P}$ such that $v \in pq$ and the edges meeting at v are both contained in RHP(pq). A *left support* is defined analogously. Since no beam emanates from a point to the left of v_0, we use the convention that v_0 is a left support of any beam v_0q. Likewise, v_1 is a right support of any beam v_1q. A *support* is a right support or a left support.

In this section, we present an $O(n)$-time algorithm to compute the visible part of one edge v_iv_{i+1} from the edge v_0v_1 using constant workspace. The algorithm is clearly asymptotically optimal even if we allow $O(n)$ workspace. Toussaint [27] devised a query algorithm to decide if one edge is visible from another edge. The algorithm first computes a triangulation of $\mathcal{P}$ as preproccessing and thus uses $\Omega(n)$ space. Hereafter, each query is answered in $O(n)$ time. We need to use Chazelle's extremely complicated triangulation algorithm [9] to make the algorithm of Toussaint run in $O(n)$ time which is very inefficient if only a few queries are made. Later, Avis et al. [5] described an $O(n)$-time algorithm computing the visible part of one edge from another needing no triangulation or other involved data structures, but using $\Omega(n)$ space. Our algorithm is clearly significantly simpler and saves a lot of memory. De et al. [13] claimed to present an $O(n)$-time algorithm using constant workspace. However, their algorithm has a fault, as we shall see.

The edge v_iv_{i+1} is *totally facing* the edge v_jv_{j+1} if both of the points v_j and v_{j+1} are in LHP(v_iv_{i+1}). Notice that v_iv_{i+1} can be totally facing v_jv_{j+1} even though v_jv_{j+1} is not visible from v_iv_{i+1}. Edge v_iv_{i+1} is *partially facing* v_jv_{j+1} if excactly one of the points v_j and v_{j+1} are in LHP(v_iv_{i+1}) and *not facing* v_jv_{j+1} if none of the points are in LHP(v_iv_{i+1}). We say that v_iv_{i+1} is *facing* v_jv_{j+1} if v_iv_{i+1} is partially or totally facing v_jv_{j+1}. It follows from the definitions that v_iv_{i+1} is either totally facing, partially facing or not facing v_jv_{j+1}. That gives 9 different combinations of how v_iv_{i+1} is facing v_jv_{j+1} and how v_jv_{j+1} is facing v_iv_{i+1}. However, only 8 of the cases are possible when v_iv_{i+1} and v_jv_{j+1} are edges of a simple polygon, since they cannot both partially face each other. That would imply that they intersect each other properly. All of the remaining 8 cases are possible, see Figure 1 for an example of each.

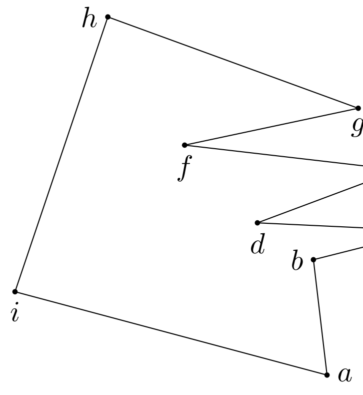


Figure 1. A polygon realizing each possible combination of types of facing between two edges: cd and fg are not facing each other. ab is partially facing de, de is not facing ab. bc is totally facing de, de is not facing bc. hi is totally facing ef, ef is partially facing hi. ia and ef are totally facing each other.

2.1. Point-to-point and point-to-edge visibility

If the edge v_iv_{i+1} is not facing edge v_0v_1, the only point on v_iv_{i+1} that can be visible from v_0v_1 is one of the endpoints v_i or v_{i+1}. Likewise, if v_0v_1 is not facing v_iv_{i+1}, the only point on v_0v_1 that can possibly see v_iv_{i+1} is one of the endpoints v_0 or v_1 by means of beams contained in $\mathrm{RHP}(v_0v_1)$. In such cases, the problem of computing the visible part of v_iv_{i+1} is reduced to point-to-point and point-to-edge visibility.

Point-to-point visibility is the problem of determining if ab is contained in $\mathcal{P}$ given two points a and b in $\mathcal{P}$. That can easily be tested in $O(n)$ time using constant workspace by traversing all of $\partial\mathcal{P}$, seeing if $\partial\mathcal{P}$ crosses ab somewhere.

Point-to-edge visibility is the slightly more complicated task of computing the visible part of an edge from a point $p \in \mathcal{P}$. Assume without loss of generality that the edge is v_0v_1. We assume that p is not collinear with two vertices of $\mathcal{P}$. Since $\mathcal{P}$ is a simple polygon, the visible part of v_0v_1 is the empty set, a segment, or possibly just a single point. If $p \notin \mathrm{LHP}(v_0v_1)$, it is only possible that v_0 or v_1 is visible from p, which can be determined with point-to-point visibility. Therefore, we assume that $p \in \mathrm{LHP}(v_0v_1)$. See Algorithm 1. The algorithm traverses $\partial\mathcal{P}$ once, finding the part of each of the ends of v_0v_1 that is in the shadow of some other edges. The algorithm has the invariant that at the end of iteration i, the visible part of v_0v_1 from p with respect of the chain $\mathcal{P}(v_1, v_{i+1})$ is ab. L and R are two indices of vertices of $\mathcal{P}$ such that $\overrightarrow{pv_L}$ exits $\mathcal{P}$ at a and $\overrightarrow{pv_R}$ exists $\mathcal{P}$ at b. In line 5, we have found an edge that makes more of the beginning of v_0v_1 invisible from p, so we update L and a. Similarly, in line 11, we have found an edge that makes more of the end of v_0v_1 invisible. Lines 6 and 12 test if the invisible parts cover all of v_0v_1, in which case nothing is visible.

Algorithm 1: `PointToEdgeVisibility`(p)

```
Input: A polygon $\mathcal{P}$ defined by its vertices $v_0, v_1, \ldots, v_{n-1}$ in CCW
       order and a point $p \in \mathcal{P}$. It is assumed that $p \in \text{LHP}(v_0v_1)$.
Output: If no point of $v_0v_1$ is visible from $p$, NULL is returned.
        Otherwise, $ab$ is returned, where $ab$ is the visible part of
        $v_0v_1$, $a \in v_0b$ and $b \in av_1$.
1  $L \leftarrow 0, \quad R \leftarrow 1$
2  $a \leftarrow v_0, \quad b \leftarrow v_1$
3  $i \leftarrow 1$
4  while $i < n$
5      if $v_iv_{i+1}$ crosses $ap$ from left to right
6          if $v_{i+1} \in \text{RHP}(bp)$
7              return NULL
8          else
9              $L \leftarrow i + 1$
10             Let $a$ be the intersection point between $v_0v_1$ and $\overrightarrow{pv_{i+1}}$
11     if $v_iv_{i+1}$ crosses $bp$ from right to left
12         if $v_{i+1} \in \text{LHP}(ap)$
13             return NULL
14         else
15             $R \leftarrow i + 1$
16             Let $b$ be the intersection point between $v_0v_1$ and $\overrightarrow{pv_{i+1}}$
17     $i \leftarrow i + 1$
18 return $ab$
```

We now turn our attention to the more interesting case of computing the visible part of v_iv_{i+1} from v_0v_1 if the edges are facing each other. We motivate the development of a new algorithm by giving a counterexample to the constant-workspace algorithm of De et al. [13]. The authors are aware of the error [20]. The reader who has not consulted their paper can skip this section.

2.2. Counterexample to the algorithm proposed by De et al. [13]

The textual description and the pseudocode in the paper of De et al. [13] does not agree. Figure 2 is an example of a polygon where the algorithm computes a wrong result in both cases. After *PASS1*(), the line L is still $p_{i+1}p_{j+1}$. After *PASS2*(), L is θp_{j+1}. The text says that *PASS3*() is to check if a vertex on $\mathcal{P}(p_{j+1}, p_i)$ is to the right of L. No one is, so the algorithm returns that the rightmost visible point on p_jp_{j+1} from p_ip_{i+1} is p_{j+1}, which is wrong. The pseudocode gives another definition of *PASS3*(), according to which we also check if a vertex on $\mathcal{P}(p_{i+1}, p_j)$ is to the left of L. Vertex v

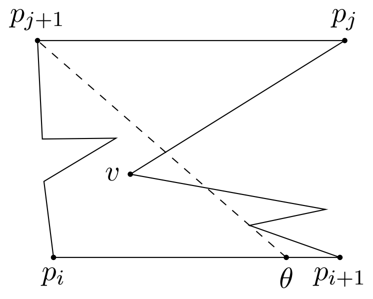


Figure 2. The algorithm from [13] reports the wrong visible part of p_jp_{j+1} from p_ip_{i+1} in this polygon.

is, so the algorithm reports that nothing of p_jp_{j+1} is visible. That is clearly also wrong.

2.3. Computing visibility between edges facing each other

Assume for the rest of this section that the edges v_0v_1 and v_iv_{i+1} are facing each other. We want to compute the part of v_iv_{i+1} containing v_{i+1} that is not visible from v_0v_1. The main idea is to consider the edges in the right side chain $\mathcal{P}(v_1, v_i)$ and the left side chain $\mathcal{P}(v_{i+1}, v_0)$ alternately, changing side after each edge. When an edge in one side is found that causes more of v_iv_{i+1} to be invisible from v_0v_1, we retract the search in the other chain to the last interfering edge in that chain. This will be made more precise in the following.

Let $\square = \square v_0v_1v_iv_{i+1}$ be the quadrilateral with vertices $v_0v_1v_iv_{i+1}$ in that order. The possible beams from v_0v_1 to v_iv_{i+1} are all contained in $\square$, so when computing the visible part of v_iv_{i+1}, we are only concerned about the edges of $\mathcal{P}$ that are (partially) in $\square$. A beam pq is a *proper beam* if $pq \subset \mathrm{LHP}(v_0v_1)$ and $pq \subset \mathrm{LHP}(v_iv_{i+1})$. An *improper beam* is a beam that is not proper. Each beam pq where p is an interior point on v_0v_1 and q is an interior point on v_iv_{i+1} is necessarily proper. Therefore, if pq is improper, $p = v_0$, $p = v_1$, $q = v_i$, or $q = v_{i+1}$. The visibility due to improper beams can be computed using point-to-edge visibility, so in this section, we focus on the visibility due to proper beams only.

Lemma 1. *Let $v_R \in \mathcal{P}(v_1, v_i) \cap \square$ and $v_L \in \mathcal{P}(v_{i+1}, v_0) \cap \square$. Every proper beam pq from v_0v_1 to v_iv_{i+1} satisfies $p \in LHP(v_Rv_L)$ and $q \in RHP(v_Rv_L)$. In particular, if $v_0v_1 \cap LHP(v_Rv_L) = \emptyset$ or $v_iv_{i+1} \cap RHP(v_Rv_L) = \emptyset$, then no proper beam from v_0v_1 to v_iv_{i+1} exists.*

Proof. Consider the continuous curve γ consisting of $\mathcal{P}(v_1, v_R)$, v_Rv_L and $\mathcal{P}(v_{i+1}, v_L)$, see Figure 3(a). γ is not necessarily simple, since the chains $\mathcal{P}(v_1, v_R)$ and $\mathcal{P}(v_{i+1}, v_L)$ might intersect v_Rv_L, but that doesn't matter for this proof. γ divides $\mathcal{P}$ into at least two regions, one to the left of v_Rv_L containing v_0v_1 and one to the right containing v_iv_{i+1}. Therefore, a proper

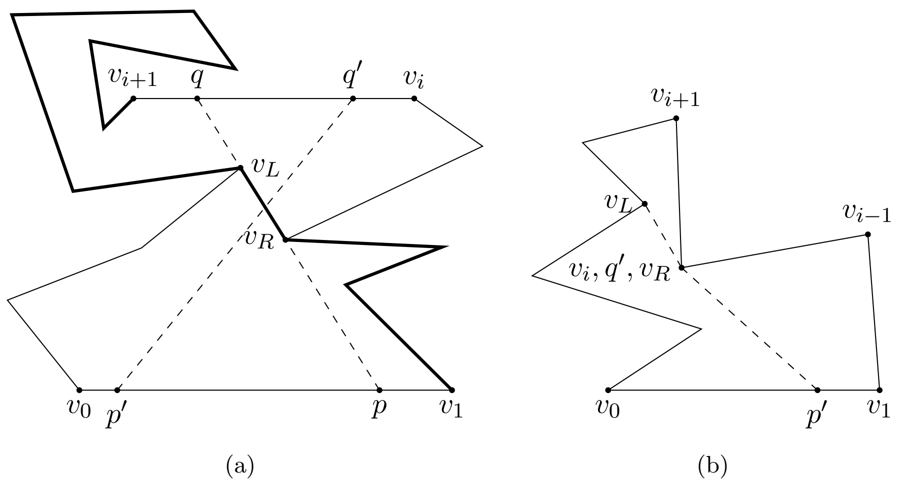


Figure 3. Illustrations for Lemma 1. (a) The curve γ is drawn with thick lines. (b) If $p' \notin \mathrm{LHP}(v_R v_L)$, the beam $p'q'$ is improper.

beam $p'q'$ must cross γ to get from v_0v_1 to v_iv_{i+1}. Since $p'q'$ does not intersect $\mathcal{P}(v_1, v_i)$ or $\mathcal{P}(v_{i+1}, v_0)$, it intersects γ only at a point on v_Rv_L. If $p' \notin \mathrm{LHP}(v_Rv_L)$, $p'q'$ must hit v_Rv_L at v_R or v_L. Assume it is v_R, the situation is showed in Figure 3(b). The line $\overrightarrow{p'q'}$ continues in $\mathrm{LHP}(v_Rv_L)$ after v_R and cannot hit v_iv_{i+1} there, so we must have $q' = v_i = v_R$. The four segments $q'p'$, v_iv_{i-1}, v_iv_{i+1}, and v_Rv_L all start at v_i, and they appear in that CCW order. Therefore, since $p'q' \in \mathrm{RHP}(v_Rv_L)$, also $v_iv_{i+1} \in \mathrm{RHP}(v_Rv_L)$, so $p' \notin \mathrm{LHP}(v_iv_{i+1})$. Hence, $p'q'$ is not a proper beam. The case where $p'q'$ hits v_L and the case where $q' \notin \mathrm{RHP}(v_Rv_L)$ can be handled similarly.

Therefore, if $p'q'$ is a proper beam, $p' \in \mathrm{LHP}(v_Rv_L)$ and $q' \in \mathrm{RHP}(v_Rv_L)$. $\square$

Assume that there are some proper beams from v_0v_1 to v_iv_{i+1}. We say that the beam pq is the *rightmost beam* from v_0v_1 to v_iv_{i+1} if p is as close to v_1 as possible and q is as close to v_{i+1} as possible among all proper beams. Similarly, pq is the *leftmost beam* from v_0v_1 to v_iv_{i+1} if p is as close to v_0 as possible and q is as close to v_i as possible. If v_0v_1 and v_iv_{i+1} are totally facing each other, all beams from v_0v_1 to v_iv_{i+1} are proper, so the visible part of v_iv_{i+1} is the points between the endpoints of the leftmost and rightmost beams. If one of the edges is only partially facing the other, the visible part of v_iv_{i+1} can be computed using the leftmost and rightmost beams in combination with point-to-edge visibility.

If pq is a beam from v_0v_1 to v_iv_{i+1}, a *generalized left support* of pq is v_{i+1} if $q = v_{i+1}$ or a left support of pq otherwise. The following lemma characterizes rightmost beams by means of their supports.

Lemma 2. *Let pq be a proper beam from v_0v_1 to v_iv_{i+1}. The beam pq is a rightmost beam if and only if pq has a right support v_R and a generalized* *left support v_L and $v_L \in v_R q$.*

Proof. If pq does not have any supports, we can choose p to be closer to v_1, so pq is no rightmost beam. If pq has a generalized left support v_L, but no right support on pv_L, we can turn pq CCW around v_L to get a proper beam from a point closer to v_1 to a point closer to v_{i+1}. The same is true if pq has a right support v_R but no generalized left support on $v_R q$.

It is also clear that if pq has a right support v_R and a generalized left support v_L, every other segment $p'q'$ from pv_1 to qv_{i+1} crosses $\partial\mathcal{P}$, so $p'q'$ is not a beam. Therefore, pq is a rightmost beam. □

If the edges v_0v_1 and v_iv_{i+1} are totally facing each other and no edge obstructs the visibility between the edges, then the rightmost beam is $pq = v_1v_{i+1}$ and it has supports $v_R = v_1$ and $v_L = v_{i+1}$.

Algorithm 2 returns the indices (R, L) of the supports of the rightmost beam if it exists. See Figure 7 for an example of how the variables change when running the algorithm on a concrete example. The algorithm iteratively computes the correct value of R and L, taking the edges into consideration one by one. Initially, R is set to 1 and L is set to $i+1$, as if no edges obstructs the visibility between the edges. The points p and q on v_0v_1 and v_iv_{i+1}, respectively, are always defined such that the segment pq contains v_R and v_L. The algorithm alternately traverses $\mathcal{P}(v_1, v_i)$ and $\mathcal{P}(v_{i+1}, v_n)$ one edge at a time using the index variables r and l. The variable *side* is 1 when an edge in $\mathcal{P}(v_1, v_i)$ is traversed and -1 when an edge in $\mathcal{P}(v_{i+1}, v_n)$ is traversed. Each time an edge $v_{r-1}v_r$ or $v_{l-1}v_l$ is found that crosses pq, the value of R or L is updated to r or l, respectively. If the value of R is updated, we reset l to L, since it is possible that there are some edges on $\mathcal{P}(v_L, v_n)$ that did not intersect the old segment pq, but intersect the updated one. Likewise, when L is updated, we reset r to R. Although segment pq is changed when R or L is updated, $\mathcal{P}(v_1, v_R)$ or $\mathcal{P}(v_{i+1}, v_L)$ does not cross pq after the update because pq is rotated CW away from the chains.

All our figures illustrate the case where v_0v_1 and v_iv_{i+1} are totally facing each other, but that assumption is not used in any of the proofs. If v_iv_{i+1} is partially facing v_0v_1 such that $v_0 \in \mathrm{LHP}(v_iv_{i+1})$, then v_i might be the right support of the rightmost beam from v_0v_1 to v_iv_{i+1}. Likewise, if v_0v_1 is partially facing v_iv_{i+1} such that $v_i \in \mathrm{LHP}(v_0v_1)$, v_0 can be the left support of the rightmost beam.

Lemma 3. *Assume that Algorithm 2 terminates after k iterations of the loop at line 5. Let R_j, L_j, p_j, and q_j be the values of R, L, p, and q, respectively, at the beginning of iteration j, $j = 1, 2, \ldots, k+1$, where the values when the algorithm terminates have index $k+1$. Then $R_j = R_{j+1}$ or $L_j = L_{j+1}$ for $j = 1, \ldots, k$. $p_1, p_2, \ldots, p_{k+1}$ is a sequence of points moving monotonically along v_0v_1 from v_1 towards v_0. Likewise, $q_1, q_2, \ldots, q_{k+1}$ is a sequence of points moving monotonically along v_iv_{i+1} from v_{i+1} towards v_i. Let a_j be the CW angle from $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ to $\overleftrightarrow{v_{R_j}v_{L_j}}$. Then $\sum_{j=2}^{k+1} a_j < 180°$. In particular, $a_j < 180°$ for each $2 = 1, \ldots, k+1$.*

Proof. See Figure 4. It is clear that at most one of R and L changes in

Algorithm 2: `FindRightmostBeam`(i)

```
Input: A polygon $\mathcal{P}$ defined by its vertices $v_0, v_1, \ldots, v_{n-1}$ in CCW
       order and an index $i$ such that $v_0v_1$ and $v_iv_{i+1}$ are facing each
       other.
Output: If no proper beam from $v_0v_1$ to $v_iv_{i+1}$ exists, NULL is
        returned. Otherwise, a pair of indices $(R, L)$ is returned such
        that the rightmost beam from $v_0v_1$ to $v_iv_{i+1}$ has right
        support $v_R$ and generalized left support $v_L$.
1  $R \leftarrow 1, \quad L \leftarrow i+1$
2  $r \leftarrow R, \quad l \leftarrow L$
3  $p \leftarrow v_1, \quad q \leftarrow v_{i+1}$
4  $side \leftarrow 1 \quad (*$ 1 is right side, $-1$ is left side $*)$
5  while $r < i$ or $l < n$
6      if $side = 1$
7          if $r < i$
8              $r \leftarrow r + 1$
9              if $v_{r-1}v_r$ enters LHP$(pq) \cap \square$
10                 if $v_{r-1}v_r$ intersects $v_Lq$
11                     return NULL
12                 $R \leftarrow r, \quad l \leftarrow L$
13     else $\quad (*$ $side = -1$ $*)$
14         if $l < n$
15             $l \leftarrow l + 1$
16             if $v_{l-1}v_l$ enters RHP$(pq) \cap \square$
17                 if $v_{l-1}v_l$ intersects $v_Rp$
18                     return NULL
19                 $L \leftarrow l, \quad r \leftarrow R$
20     Let $p$ be the intersection point between $\overrightarrow{v_Lv_R}$ and $v_0v_1$
21     Let $q$ be the intersection point between $\overrightarrow{v_Rv_L}$ and $v_iv_{i+1}$
22     if $p$ or $q$ does not exist
23         return NULL
24     $side \leftarrow -side$
25 if $pq \subset$ LHP$(v_0v_1) \cap$ LHP$(v_iv_{i+1})$
26     return $(R, L)$
27 else
28     return NULL
```

iteration j, since the lines 12 and 19 cannot both be executed. Therefore, $R_j = R_{j+1}$ or $L_j = L_{j+1}$. If R is redefined in iteration j, then $\overleftrightarrow{v_Rv_L}$ is rotating around v_L and the new value of R, namely R_{j+1}, satisfies $v_{R_{j+1}} \in$ LHP$(v_{R_j}v_{L_j})$. Therefore, p_{j+1} is on the segment v_0p_j and q_{j+1}

is on the segment q_jv_i. The same is true if L is updated. Hence, $p_1, \dots, p_{k+1}$ is monotinically moving along v_0v_1 from v_1 towards v_0 and $q_1, \dots, q_{k+1}$ is monotonically moving along v_iv_{i+1} from v_{i+1} towards v_i. Because of the monotonicity, the angles are additive, so that the CW angle from $\overleftrightarrow{v_{R_1}v_{L_1}}$ to $\overleftrightarrow{v_{R_{k+1}}v_{L_{k+1}}}$ is $\sum_{j=2}^{k+1} a_j$. Since v_0v_1 and v_iv_{i+1} are facing each other, at least one of them is totally facing the other. If v_0v_1 is totally facing v_iv_{i+1}, every q_j is contained in $\mathrm{LHP}(v_0v_1)$. Otherwise v_iv_{i+1} is totally facing v_0v_1 so that every p_j is contained in $\mathrm{LHP}(v_iv_{i+1})$. In either case, $\sum_{j=2}^{k+1} a_j$ is bounded by 180°. That bound cannot be reached, since it would require that v_0v_1 or v_iv_{i+1} was infinitely long in both directions. □

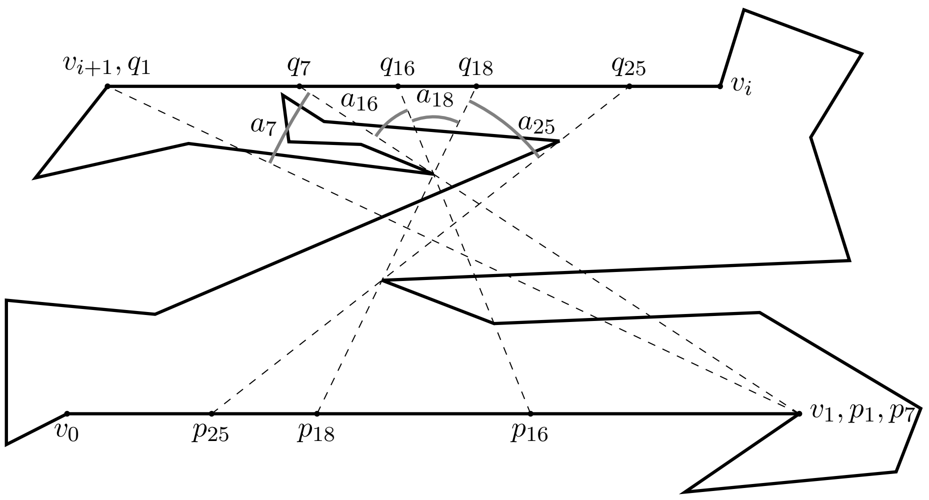


Figure 4. Illustration for Lemma 3. The points p_j, q_j are shown with the number j of the first iteration where they occur. The segments p_jq_j are drawn dashed. The angles $a_j > 0$ are indicated with grey arcs. The polygon is the same as the one in Figure 7.

Lemma 4. *Algorithm 2 correctly computes the rightmost beam from v_0v_1 to v_iv_{i+1} as specified. The algorithm is a constant-workspace algorithm.*

Proof. First, consider the cases where the algorithm returns `NULL`. In line 11, we have found an intersection point x between $\mathcal{P}(v_R, v_i)$ and v_Lq. That means that $\mathcal{P}(v_1, v_i)$ intersects pq properly at x, since no three vertices are collinear. Lemma 1 establishes that the only possible proper beams from v_0v_1 to v_iv_{i+1} are of the form $p'q'$, where $p' \in v_0p$ and $q' \in qv_i$. At the same time, if we use Lemma 1 with v_0v_1 and v_iv_{i+1} interchanged by each other and using x as 'v_L' and v_L as 'v_R', we get that $p'q'$ satisfies $p' \in pv_1$ and $q' \in v_{i+1}q$. Therefore, $p' = p$ and $q' = q$, but pq is not a beam. Hence, there are no proper beams from v_0v_1 to v_iv_{i+1}. The case in line 18 is analogous.

Due to Lemma 3, we know that p is moving monotonically from v_1 towards v_0 and q is moving monotonically from v_{i+1} towards v_i. The case of line 23 happens if p has moved outside v_0v_1, so that $v_0v_1 \cap \mathrm{LHP}(v_Rv_L) = \emptyset$, or q has moved outside v_iv_{i+1}, so that $v_iv_{i+1} \cap \mathrm{RHP}(v_Rv_L) = \emptyset$. In each of these

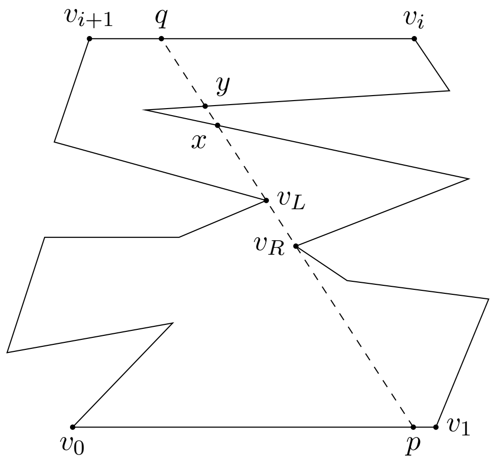


Figure 5. Case 2 in the proof of Theorem 4.

cases, it follows from Lemma 1 that there are no proper beams from v_0v_1 to v_iv_{i+1}.

The test at line 25 is to ensure that pq is a proper beam, which is not always the case if v_0v_1 is only partially facing v_iv_{i+1}.

Now, assume that the algorithm returns (R, L), but that pq is not a beam since some edge obstructs the visibility from p to q. Assume that $\mathcal{P}(v_1, v_i)$ intersects pq properly, and let x be the intersection point closest to p. $\mathcal{P}(v_1, v_i)$ enters $\text{LHP}(pq) \cap \square$ at x. Let y be the first point where $\mathcal{P}(x, v_i)$ crosses pq from left to right. Then $y \in xq$. We have two cases: $x \in \mathcal{P}(v_1, v_R)$ (case 1) and $x \in \mathcal{P}(v_R, v_i)$ (case 2). Assume that we are in case 2, see Figure 5. Assume that the final value of R is defined in a later iteration of the loop at line 5 than the final value of L. After R is defined in line 12, every edge $v_{r-1}v_r$ in $\mathcal{P}(v_R, v_i)$ is traversed and it is checked in line 10 if some edge intersects pq. In particular the edges in $\mathcal{P}(x, y)$ are traversed, in which case the algorithm either returns `NULL` or updates R, which is a contradiction. If R is defined in an earlier iteration than L, then r is reset to R in line 19 when L is defined, and it is checked if some edge in $\mathcal{P}(v_R, v_i)$ intersects pq, so that cannot happen either.

Assume that we are in case 1, i.e. $x \in \mathcal{P}(v_1, v_R)$. Consider the first iteration, say iteration j, at the beginning of which $\mathcal{P}(v_1, v_R)$ intersects pq properly, and let x' be the intersection point closest to p. Let y' be the first point where $\mathcal{P}(x', v_i)$ crosses pq from left to right. Then $y' \in x'q$ (x' and y' might not be the same as x and y, since R and L can change before the algorithm terminates). We must have $v_R \in \mathcal{P}(y', v_i)$. There are three possible cases to consider: $v_R \in px'$ (case 1.1), $v_R \in x'y'$ (case 1.2), and $v_R \in y'q$ (case 1.3).

Assume case 1.3. Let R_k, L_k, p_k, and q_k be defined as in Lemma 3 for each iteration k. Either R or L is redefined in iteration $j-1$ due to the minimality

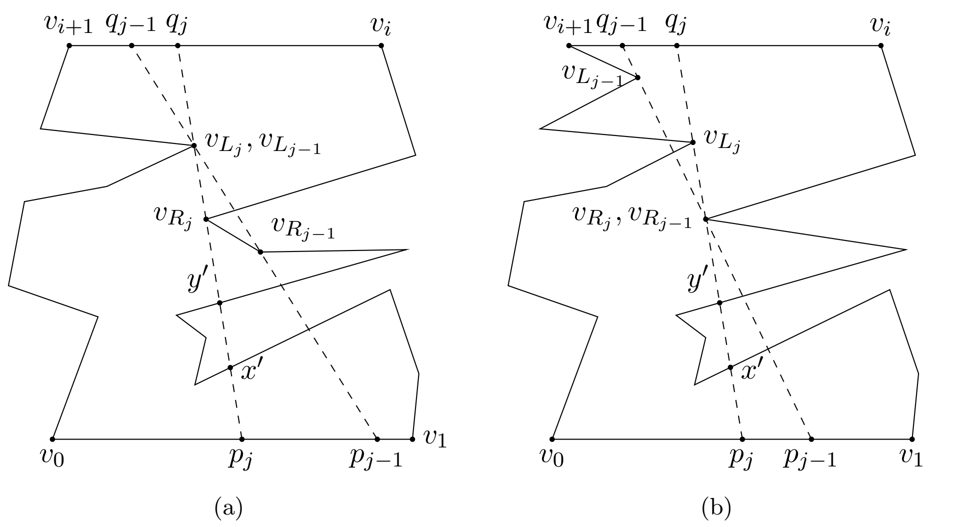


Figure 6. Cases in the proof of Theorem 4. (a) Case 1.3.1.3. (b) Case 1.3.2.

of j. Therefore, $R_{j-1} \neq R_j$ or $L_{j-1} \neq L_j$ (case 1.3.1 and 1.3.2, respectively). First, assume $R_{j-1} \neq R_j$ but $L_{j-1} = L_j$. Again, there are three cases to consider: $v_{R_{j-1}} \in \mathcal{P}(v_1, x')$ (case 1.3.1.1), $v_{R_{j-1}} \in \mathcal{P}(x', y')$ (case 1.3.1.2), and $v_{R_{j-1}} \in \mathcal{P}(y', v_{R_j})$ (case 1.3.1.3). Assume case 1.3.1.3, see Figure 6(a). According to Lemma 3, the CW angle between $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ and $\overleftrightarrow{v_{R_j}v_{L_j}}$ is less than 180°. Therefore, a subset of $\mathcal{P}(x', y')$ would also be contained in $\mathrm{LHP}(v_{R_{j-1}}v_{L_{j-1}}) \cap \Box$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1}q_{j-1}$, a contradiction because of the choice of j. $v_{R_{j-1}}$ cannot be in $\mathcal{P}(x', y')$ (case 1.3.1.2), because then v_{R_j} would be in $\mathrm{RHP}(v_{R_{j-1}}v_{L_{j-1}})$, so R would not have been redefined to R_j in iteration $j-1$. Finally, if $v_{R_{j-1}}$ was a vertex in $\mathcal{P}(v_1, x')$ (case 1.3.1.1), $\mathcal{P}(x', y')$ would be contained in $\mathrm{LHP}(v_{R_{j-1}}v_{L_{j-1}}) \cap \Box$, and therefore v_R would be redefined to a vertex in $\mathcal{P}(x'y')$ when the edges of that chain was traversed. Hence, v_R would not be redefined to v_{R_j} in iteration $j-1$, which is a contradiction.

Now, assume $L_{j-1} \neq L_j$ (case 1.3.2), see Figure 6(b). The CW angle between $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ and $\overleftrightarrow{v_{R_j}v_{L_j}}$ is less than 180°. Therefore, a part of $\mathcal{P}(x', y')$ is also in $\mathrm{LHP}(v_{R_{j-1}}v_{L_{j-1}})$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1}q_{j-1}$, which contradicts the choice of j.

The case where $v_R \in x'y'$ (case 1.2) can be eliminated in a similar way. Consider case 1.1, i.e. $v_R \in px'$. The chain $\mathcal{P}(p, x')$ and the segment $x'p$ forms a simple, closed curve, because x' is the intersection point between $\mathcal{P}(v_1, v_i)$ and pq closest to p. The curve can, for instance, be seen in Figure 6(a). Consider the region of $\mathcal{P}$ enclosed by the curve. In order to get to v_R, $\mathcal{P}(y', v_i)$ has to cross $x'p$ to get into the region. That contradicts that x' was the intersection point closest to p.

If we assume that $\mathcal{P}(v_{i+1}, v_n)$ intersects pq, we get a contradiction in an analogous way.

The conclusion is that if (R, L) is returned, v_R and v_L defines a proper beam pq with right support v_R and generalized left support v_L in that order. Therefore, pq must be the rightmost beam from v_0v_1 to v_iv_{i+1} according to Lemma 2.

Observe that the vertices of $\mathcal{P}$ are not altered. Hence the input is read only. In addition to that, we only use the variables R, L, r, l, p, q, and *side*. The computations of intersections and containment at lines 9, 10, 16, 17, 20, 21, and 25 are easily implemented using constant workspace. $\square$

Even though we reset l to L in line 12 or r to R in line 19, the running time is linear since the other variable is not reset, so half of the traversed edges are never traversed again, as the following lemma explains.

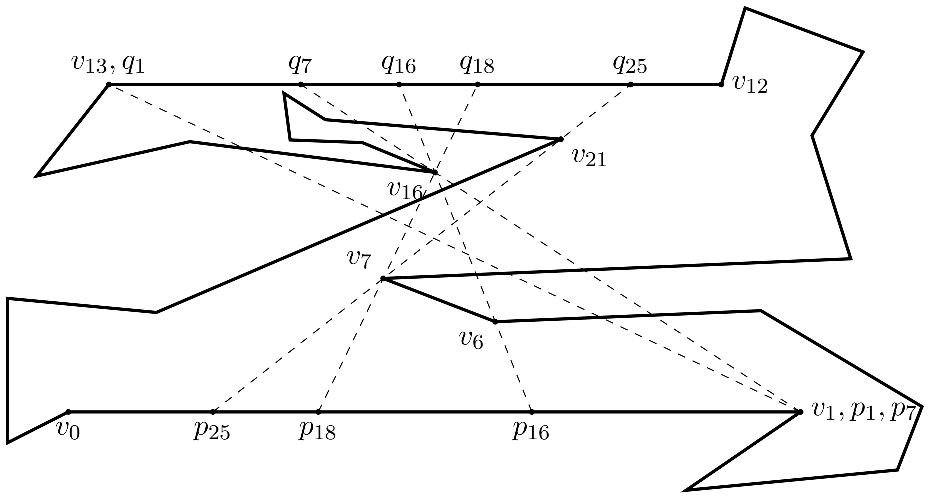
Lemma 5. *There are at most $2n - 6$ iterations of the loop at line 5 of Algorithm 2.*

Proof. Let $N(n)$ be the maximal number of edge visits for a polygon with n vertices. Consider the first time line 12 or 19 is executed. Assume it is line 12. There have been made $2(r-1)-1 < 2(r-1)$ iterations, because $\mathcal{P}(v_1, v_i)$ is traversed every second time, beginning with the first. The $r - 1$ edges in $\mathcal{P}(v_1, v_r)$ are never traversed again. Therefore, N satisifies the recurrence $N(n) \leq 2k + N(n - k)$, where $k = r - 1$. A similar bound holds for some $k \geq 1$ if line 19 is executed first. We know that $N(4) = 2$, so induction yields that $N(n) \leq 2n - 6$ is an upper bound. $\square$

It is clear that an algorithm to compute a leftmost beam from v_0v_1 to v_iv_{i+1} can be constructed symmetrically. As mentioned before, the rightmost and leftmost beams are only sufficient to find the visible part of v_iv_{i+1} if v_0v_1 and v_iv_{i+1} are totally facing each other. Otherwise, we can compute the visible part by taking the union of the interval on v_iv_{i+1} between the leftmost and rightmost beams and the result of point-to-point and point-to-edge visibility computed from the endpoints of the edges. That gives us the following theorem:

Theorem 1. *The visible part of an edge v_iv_{i+1} from v_0v_1 in a simple polygon can be computed in $O(n)$ time using constant workspace.*

Interestingly, as we shall see in the next section, we only need the method `FindRightmostBeam` and not its symmetrical counterpart `FindLeftmostBeam` in order to report the boundary of the weak visibility polygon $\mathcal{WV}_{\mathcal{P}}(v_0v_1)$, consisting of all the points that can be seen from v_0v_1.



(a)

Iteration	*side*	*r*	*R*	*l*	*L*
1	1	1	1	13	13
2	-1	2	1	13	13
3	1	2	1	14	13
4	-1	3	1	14	13
5	1	3	1	15	13
6	-1	4	1	15	13
7	1	1	1	16	16
8	-1	2	1	16	16
9	1	2	1	17	16
10	-1	3	1	17	16
11	1	3	1	18	16
12	-1	4	1	18	16
13	1	4	1	19	16
14	-1	5	1	19	16
15	1	5	1	20	16
16	-1	6	6	16	16
17	1	6	6	17	16
18	-1	7	7	16	16

Iteration	*side*	*r*	*R*	*l*	*L*
17	1	7	7	17	16
18	-1	8	7	17	16
19	1	8	7	18	16
20	-1	9	7	18	16
21	1	9	7	19	16
22	-1	10	7	19	16
23	1	10	7	20	16
24	-1	11	7	20	16
25	1	7	7	21	21
26	-1	8	7	21	21
27	1	8	7	22	21
28	-1	9	7	22	21
29	1	9	7	23	21
30	-1	10	7	23	21
31	1	10	7	24	21
32	-1	11	7	24	21
33	1	11	7	25	21
34	-1	12	7	25	21

(b)

Figure 7. Example showing how Algorithm 2 is working when finding the leftmost beam from v_0v_1 to $v_{12}v_{13}$ in the shown polygon. The values of p_j and q_j as defined in Lemma 3 are also shown. The value of each variable in the beginning of each iteration is listed in the table (b).

3. The Weak Visibility Polygon

Consider a segment $s = ab$ in a simple polygon $\mathcal{P}$. Visibility from s is a natural generalization of visibility from an edge of $\mathcal{P}$. We also define beams emanating from s and supports of the beams analogously. The *weak visibility polygon* $\mathcal{WV}_{\mathcal{P}}(s)$ is the set of all points in $\mathcal{P}$ visible from s.[1] See Figure 1. When $\mathcal{P}$ and s are clear from the context, we write $\mathcal{WV}$ for brevity. Intuitively, one can think of $\mathcal{WV}$ as the area in $\mathcal{P}$ that is visible to some guard patrolling back and forth on s, or the part of $\mathcal{P}$ that is enlightened if s emits light like a fluorescent tube lamp.

It is well known that $\mathcal{WV}$ is a polygon [6]. A beam pq is *maximal* if $\overrightarrow{pq}$ leaves $\mathcal{P}$ at q. The boundary of $\mathcal{WV}$ consists of chains of $\mathcal{P}$ and chords connecting the chains. The chords separate $\mathcal{WV}$ from regions of $\mathcal{P}$ not visible from s. Such regions are called *pockets* of $\mathcal{WV}$ and the chords are *doors*. Each door vq is the last part of a unique maximal beam pq. If a pocket is to the left of the beam containing its door, it is a *left pocket*. Otherwise, its a *right pocket*. At most two doors have an endpoint on a given edge of $\mathcal{P}$. If n and m denote the number of vertices of $\mathcal{P}$ and $\mathcal{WV}$, respectively, it follows that $m = O(n)$.

[1] The reason that we use the word 'weak' is that there are other natural definitions of visibility from a segment, namely 'strong' and 'complete' visibility. We refer to Avis and Toussaint [6] for the details.

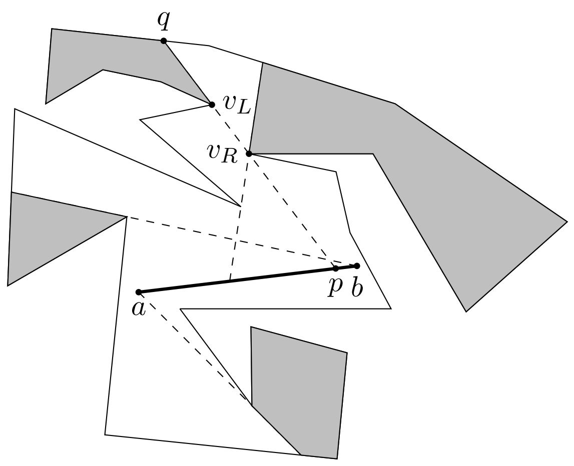


Figure 1. The weak visibility polygon $\mathcal{WV}$ of the segment ab. The beams containing doors are drawn with dashed lines. The grey areas are the pockets. The beam pq has right support v_R and left support v_L, and v_Lq is the door to a left pocket.

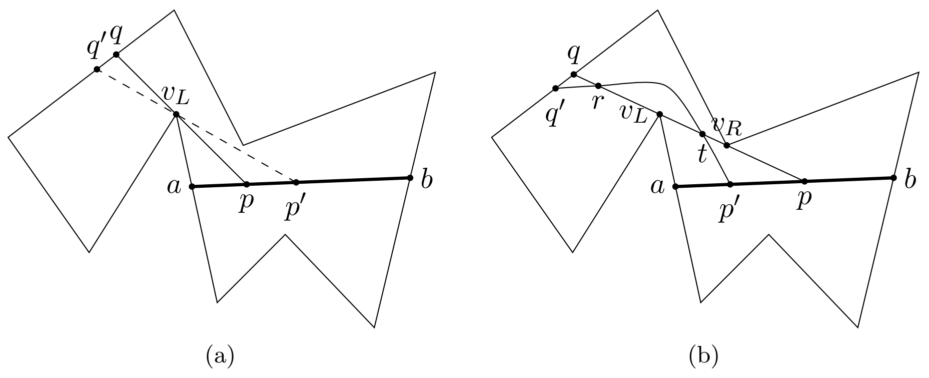


Figure 2. Illustrations for the proof of Lemma 6, where $s = ab$. (a) When there is no right support before v_L on beam pq, v_Lq is no door. (b) A beam hitting a point to the left of v_Lq does not exist.

The following lemma characterizes the beams containing a door.

Lemma 6. *Let pq be a maximal beam emanating from a segment s.*

(1) pq contains a door v_Lq to a left pocket if and only if pq has a right support v_R and a left support v_L in that order following pq from p.

(2) pq contains a door v_Rq to a right pocket if and only if pq has a left support v_L and a right support v_R in that order following pq from p.

Proof. We prove (1); the case (2) follows analogously. If pq has no supports, there exist beams containing all points in a neighborhood around pq, so pq does not contain a door. If pq has left support v_L but no right support before v_L, there exists beams containing all points in a neighborhood to the left of v_Lq, so there cannot be a door to a left pocket on pq, see Figure 2(a). The same is true if pq has a right support v_R but no left support after v_R.

Now assume that pq has supports v_R and v_L as defined in the lemma. See Figure 2(b). Assume that some point q' to the left of v_Lq is visible from s. Let $p'q'$ be a beam from s to q'. The segment v_Lq divides $\mathcal{P}$ into two regions, one containing p' and one containing q'. The beam $p'q'$ is a continuous curve from p' to q', so it intersects v_Lq at some point r. Also, the segment v_Rv_L divides $\mathcal{P}$ into a region containing p' and one containing q', so $p'q'$ must intersect v_Rv_L at some point t. Since the beams pq and $p'q'$ share two points t and r, they must be on the same line, so no point to the left of v_Lq is visible from s. Therefore, v_Lq is a door to a left pocket. □

The vertices of $\mathcal{WV}$ are either vertices of $\mathcal{P}$ visible from s or endpoints of beams containing a door. Hence, the following lemma clearly follows from Lemma 6:

Lemma 7. *A point q is a vertex of $\mathcal{WV}_{\mathcal{P}}(s)$ if and only if one of the following three properties holds:*

(1) q is a vertex of $\mathcal{P}$ visible from s.

(2) There is a maximal beam pq with a right support v_R and a left support v_L in that order following pq from p.

(3) *There is a maximal beam pq with a left support v_L and a right support v_R in that order following pq from p.*

Guibas et al. [16] presented an $O(n)$-time algorithm to compute the weak visibility polygon from an edge of $\mathcal{P}$ if a triangulation of $\mathcal{P}$ was provided, where n is the number of vertices of $\mathcal{P}$. Later, Chazelle [9] described an $O(n)$ time deterministic triangulation algorithm, implying that $\mathcal{WV}$ can be computed in $O(n)$ time using $O(n)$ space. We present an $O(mn)$ algorithm using constant workspace, where m is the number of edges of $\mathcal{WV}$. As is mentioned before, $m = O(n)$, so the algorithm runs in time $O(n^2)$.

3.1. Computing the weak visibility polygon from an edge

We show how to compute $\mathcal{WV}_{\mathcal{P}}(v_0v_1)$ given that the vertices v_0 and v_1 are both convex. Under that assumption, the weak visibility polygon is a subset of LHP(v_0v_1), so that we do not need to consider the visible part of $\mathcal{P}$ in RHP(v_0v_1) from the endpoints v_0 and v_1. We eventually show how to remove that assumption and how to generalize the method to computing $\mathcal{WV}_{\mathcal{P}}(ab)$ for an arbitrary segment ab in $\mathcal{P}$. The following lemmas lead to the desired algorithm.

Lemma 8. *Assume that v_0 and v_1 are convex vertices and that q_1q_2 is the visible part of an edge v_iv_{i+1}, $q_1 \neq q_2$ and $q_1 \in v_iq_2$. Then the rightmost beam from v_0v_1 to v_iv_{i+1} has the form pq_2, $p \in v_0v_1$.*

Proof. Since v_0 and v_1 are convex, a point q on an edge v_iv_{i+1} where $q \in$ RHP(v_0v_1) is not visible, because $v_{n-1}v_0$ obstructs the visibilty from v_0 and v_1v_2 obstructs the visibilty from v_1 to all points in RHP(v_0v_1). Hence all beams from v_0v_1 to v_iv_{i+1} are contained in LHP(v_0v_1). Since $q_1 \neq q_2$, the visible part of v_iv_{i+1} is not just an endpoint. Therefore, there must be beams contained in LHP(v_iv_{i+1}) to all the points on the segment q_1q_2. In particular, there is a proper beam ending at q_2, so the rightmost beam is on the form pq_2. □

Lemma 8 argues for the existence of the rightmost beam pq_2 in the following lemma.

Lemma 9. *Assume that v_0 and v_1 are convex vertices and that $\mathcal{WV} = \mathcal{WV}_{\mathcal{P}}(v_0v_1)$ has an edge $q_1q_2 \subseteq v_iv_{i+1}$ where $q_1 \in v_iq_2$. Let pq_2 be the rightmost beam from v_0v_1 to v_iv_{i+1} and let v_R and v_L be its right support and generalized left support, respectively. Let s be the point on v_0v_1 closest to v_0 that can see q_2.*

(1) *If $L = i+1$ and $s \in$ LHP$(v_{i+1}v_{i+2})$, the edge following $q_1q_2 = q_1v_{i+1}$ CCW on $\partial\mathcal{WV}$ has the form $v_{i+1}u \subseteq v_{i+1}v_{i+2}$.*

(2) *If $L = i+1$ and $s \notin$ LHP$(v_{i+1}v_{i+2})$, let x be the point where $\overrightarrow{sv_{i+1}}$ exits $\mathcal{P}$ and let v_jv_{j+1} be the edge containing x. The edge following $q_1q_2 = q_1v_{i+1}$ CCW on $\partial\mathcal{WV}$ is $v_{i+1}x$ and then an edge on the form $xu \subset v_jv_{j+1}$ follows.*

(3) *If $L \neq i+1$, the edge following q_1q_2 CCW on $\partial\mathcal{WV}$ is q_2v_L and then an edge on the form $v_Lu \subseteq v_Lv_{L+1}$ follows.*

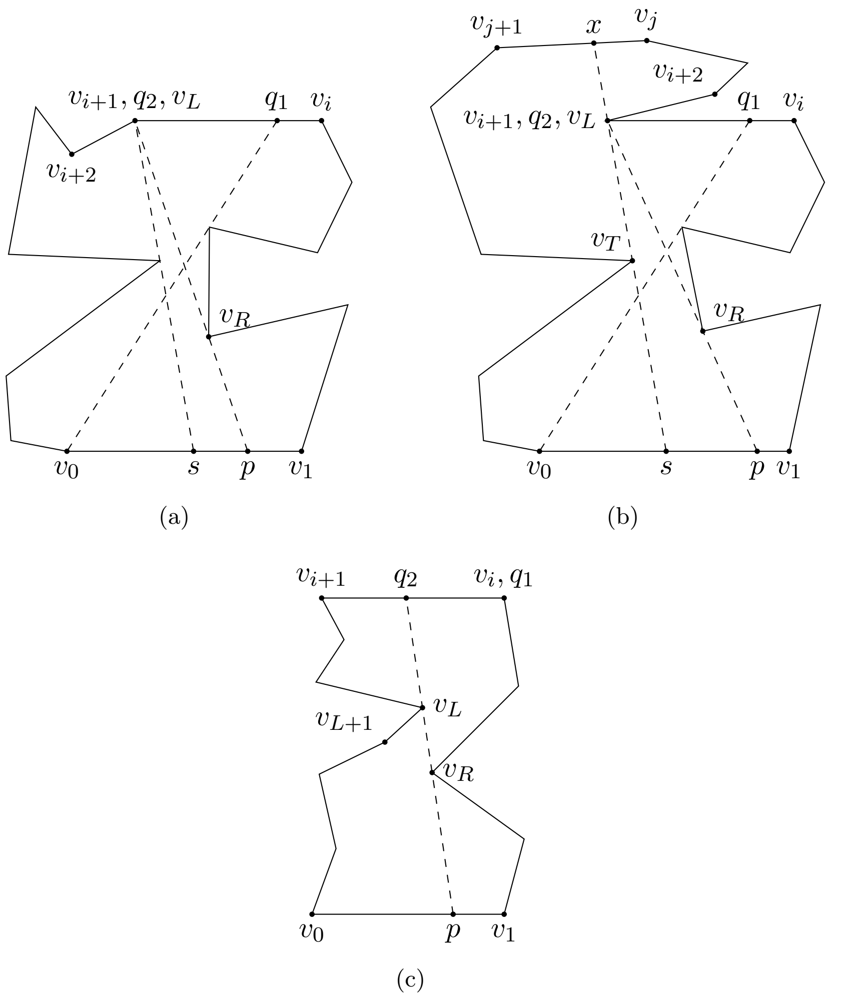


Figure 3. The three cases of Lemma 9. (a) Case (1). (b) Case (2). (c) Case (3).

Proof. Figure 3 illustrates each of the cases. Consider case (1). Since $L = i+1$, vertex v_{i+1} is visible from s. It is also given that $s \in \mathrm{LHP}(v_{i+1}v_{i+2})$, so a portion $v_{i+1}u$ of $v_{i+1}v_{i+2}$ must be visible from sv_1. The edge $v_{i+1}u$ is therefore the edge following q_1v_{i+1} on $\partial\mathcal{WV}$.

Consider case (2). Since s is the point closest to v_0 that can see v_{i+1}, the beam sv_{i+1} has a left support v_T. Therefore, the maximal beam sx has a left support v_T and a right support v_{i+1}. Hence $v_{i+1}x$ is a door to a right pocket according to Lemma 6. It follows that $v_{i+1}x$ is the edge after q_1v_{i+1} on $\partial\mathcal{WV}$, and that the edge following that has the form $xu \subset v_jv_{j+1}$.

Consider case (3). According to Lemma 6, the segment q_2v_L is a door to a left pocket. Therefore, the edge following q_1q_2 on $\partial\mathcal{WV}$ is q_2v_L, and then follows an edge of the form $v_Lu \subseteq v_Lv_{L+1}$. $\square$

Algorithm 3 reports $\mathcal{WV}_{\mathcal{P}}(v_0v_1)$ using the characterisation of Lemma 9. The correctness follows immediately.

Algorithm 3: Report $\mathcal{WV}_{\mathcal{P}}(v_0v_1)$

```
Input: A polygon $\mathcal{P}$ defined by its vertices $v_0, v_1, \dots, v_{n-1}$ in CCW
       order. The vertices $v_0$ and $v_1$ are convex.
Output: The vertices of $\mathcal{WV}_{\mathcal{P}}(v_0v_1)$ starting with $v_0$ are reported.
1  report $v_0$
2  report $v_1$
3  $i \leftarrow 1$
4  while $i < n-1$
5      $(R, L) \leftarrow$ FindRightmostBeam($i$)
6      if $L = i + 1$
7          report $v_{i+1}$
8          Let $s$ be the point on $v_0v_1$ closest to $v_0$ that can see $v_{i+1}$
9          if $s \in$ LHP$(v_{i+1}v_{i+2})$
10             $i \leftarrow i + 1$
11         else
12             Let $x$ be the point where $\overrightarrow{sv_{i+1}}$ exits $\mathcal{P}$ and let $v_iv_{i+1}$ be the
               edge containing $x$
13             report $x$
14     else
15         Let $q$ be the intersection point between $v_iv_{i+1}$ and $\overrightarrow{v_Rv_L}$
16         report $q$
17         report $v_L$
18         $i \leftarrow L$
```

Theorem 2. *Algorithm 3 can be implemented using constant workspace so that it runs in $O(mn)$ time, where m is the number of vertices of $\mathcal{WV}$.*

Proof. Each call of line 5 takes $O(n)$ time according to Theorem 5. Line 8 can be implemented as point-to-edge visibility by computing the visible part of v_0v_1 from v_{i+1}. Line 12 also takes $O(n)$ time, since we need to find the first edge intersecting $\overrightarrow{sv_{i+1}}$ properly. In total, we use $O(n)$ time on each iteration of the loop at line 4. In each iteration, we report one or two vertices. Therefore, the algorithm runs in $O(mn)$ time. □

3.2. Generalizing the method to arbitrary segments

Consider the task of computing $\mathcal{WV}_{\mathcal{P}}(ab)$ for an arbitrary segment ab in $\mathcal{P}$. We assume that no vertex of $\mathcal{P}$ is on the line $\overleftrightarrow{ab}$. We leave it to the reader to consider the case where there is a vertex on $\overleftrightarrow{ab}$, in particular the case where a or b is a vertex. Let $a'b'$ be the maximal chord containing ab such that $\overrightarrow{ab}$ leaves $\mathcal{P}$ at b' and $\overrightarrow{ba}$ leaves $\mathcal{P}$ at a'. Assume that a' is on edge

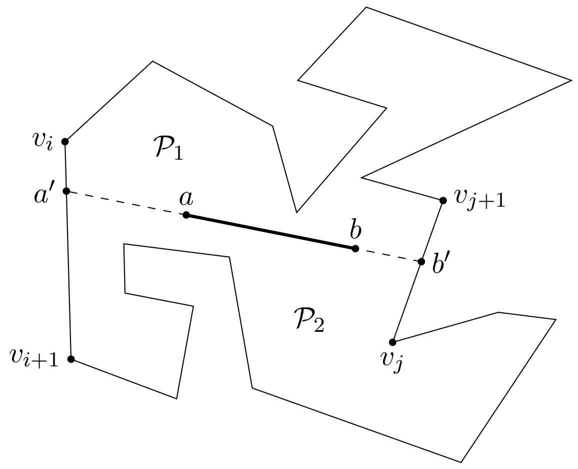


Figure 4. Computing $\mathcal{WV}_{\mathcal{P}}(ab)$ by splitting $\mathcal{P}$ into two pieces.

v_iv_{i+1} and b' is on v_jv_{j+1}, $0 \leq i < j \leq n-1$, see Figure 4. We make a modification $\mathcal{P}'$ of $\mathcal{P}$ in the following way: We cut the edge v_iv_{i+1} at a' and insert the edges $a'a$ and aa'. Likewise, we cut v_jv_{j+1} at b' and insert $b'b$ and bb'. The segment ab is a chord in $\mathcal{P}'$, dividing it into two polygons $\mathcal{P}_1$ and $\mathcal{P}_2$. In both $\mathcal{P}_1$ and $\mathcal{P}_2$, the four vertices a', a, b, and b' are collinear, but after inspection, it is seen that Algorithm 3 can report the boundaries of $\mathcal{WV}_{\mathcal{P}_1}(ab)$ and $\mathcal{WV}_{\mathcal{P}_2}(ab)$ correctly. Combining the two results and being careful not to report unwanted vertices, we get a method for reporting the boundary of $\mathcal{WV}_{\mathcal{P}}(ab)$.

4. Minimum Link Paths Between Two Points

Let two points s and t inside a simple polygon $\mathcal{P}$ be given. A *link path* from s to t is a polygonal chain $x_0 \cdots x_k$ such that $x_0 = s$, $x_k = t$, and $x_i x_{i+1}$ is contained in $\mathcal{P}$ for each $i = 0, \ldots, k-1$. The link path is a *minimum link path* if k is as small as possible, in which case k is the *link distance* between s and t. Contrary to the shortest path between s and t inside $\mathcal{P}$, a minimum link path is not unique in general. One can think of the problem of computing a minimum link path as the problem of placing as few mirrors as possible that enables an observer at s to see t, namely one mirror at each vertex x_i, $i = 1, \ldots, k-1$. Suri [26] showed how to compute a minimum link path using $O(n)$ time if a triangulation of $\mathcal{P}$ is provided. Using the very complicated triangulation algorithm of Chazelle [9], a minimum link path can be computed from scratch in $O(n)$ time and space. In this section, we show how to compute a minimum link path using constant workspace in $O(n^2)$ time.

Let $d_0 = ss$ be the degenerated segment consisting only of the point s and let $\mathcal{P}_0 = \mathcal{P}$. Consider the weak visibility polygon $\mathcal{WV}_0 = \mathcal{WV}_{\mathcal{P}_0}(d_0)$, i.e., the set of points q such that $sq \subset \mathcal{P}$. If t is contained in $\mathcal{WV}_0$, the segment st is contained in $\mathcal{P}$, and hence st is the unique minimal link path. Otherwise, t is contained in some pocket $\mathcal{P}_1 \subset \mathcal{P}_0$ of $\mathcal{WV}_0$ with door d_1. In that case, the link distance must be larger than 1. In general, if t is in $\mathcal{P}_i$ but not in $\mathcal{WV}_i$, t must be in a pocket $\mathcal{P}_{i+1}$ of $\mathcal{WV}_i$ with door d_{i+1}, and we let $\mathcal{WV}_{i+1} = \mathcal{WV}_{\mathcal{P}_{i+1}}(d_{i+1})$, see Figure 1. Thus, we have a sequence of nested polygons

$$t \in \mathcal{P}_{k-1} \subset \mathcal{P}_{k-2} \subset \ldots \subset \mathcal{P}_0 = \mathcal{P},$$

and a sequence of weak visibility polygons

$$\mathcal{WV}_i \subset \mathcal{P}_i, \quad i = 0, \ldots, k-1,$$

where $\mathcal{WV}_i$ has door d_{i+1} to the pocket $\mathcal{P}_{i+1}$. Therefore, d_{i+1} is a common edge on the boundaries of $\mathcal{WV}_i$ and $\mathcal{WV}_{i+1}$. There is a unique beam $p_i q_i$ emanating from d_i that contains d_{i+1} because d_{i+1} is a door to a pocket of $\mathcal{WV}_i$, see Lemma 6. Choose p_{k-1} such that $p_{k-1}t$ is a beam from d_{k-1} and let $p_k = t$. The following result is due to Suri [26].

Lemma 10. *The path $p_0 p_1 \cdots p_k$ is a minimum link path.*

Proof. We first notice that each of the segments $p_i p_{i+1}$ is contained in $\mathcal{P}$, since $p_{i+1} \in d_{i+1}$ and p_i is the starting point of the unique beam $p_i q_i$ emanating from d_i that contains d_{i+1}. It remains to prove that k is minimal.

Consider a minimal link path $X = x_0 x_1 \cdots x_l$ from s to t. Since $p_0 p_1 \cdots p_k$ is a link path from s to t, we have $l \leq k$. The path X must cross each of the

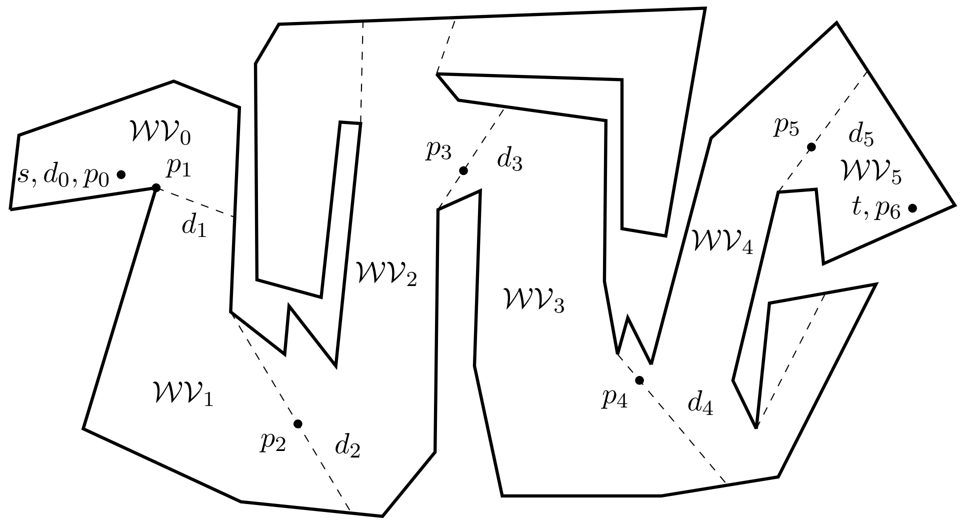


Figure 1. The construction of a minimum link path $p_0p_1 \ldots p_6$ from s to t. The doors on the boundaries of weak visibility polygons are drawn with dashed lines.

doors $d_1, \ldots, d_{k-1}$, since each of them divides $\mathcal{P}$ into one part containing s and one containing t. A segment x_ix_{i+1} crossing door d_j ends in $\mathcal{WV}_j$ because $\mathcal{WV}_j$ contains all points visible from d_j. The segment x_0x_1 is contained in $\mathcal{WV}_0$. Therefore, there are at least as many links on X as weak visibility polygons $\mathcal{WV}_j$, $j = 0, \ldots, k-1$. Hence, $l \geq k$ and we conclude that $p_0p_1 \cdots p_k$ is indeed a minimum link path. $\square$

Algorithm 4 reports a minimum link path using the construction. The variable $\mathcal{P}'$ stores the pocket in which t is currently known to be. The variable d is the door of $\mathcal{P}'$.

Theorem 3. *Algorithm 4 can be implemented so that it uses constant workspace and $O(n^2)$ time.*

Proof. The pocket $\mathcal{P}'$ can be stored using two values, namely the start- and endpoint of the common boundary of $\mathcal{P}$ and $\mathcal{P}'$ in CCW order. The test in the loop at line 3 can be decided by point-to-edge visibility in $O(n)$ time using constant workspace. In order to perform the test in line 5 in constant time, we do the following precomputation inspired by Asano et al. [4]: We let t' be the point where the upward-going vertical half-line starting at t exits $\mathcal{P}$. t' can be found in linear time using constant workspace. We store the indices of the vertices that are the endpoints of the segment containing t' as well. That enables us to determine if t' is on the boundary of the pocket $\mathcal{Q}$ with door ab in constant time, since we know the indices of a and b or the indices of the edges containing them. We can also in constant time determine if ab intersects tt'. t is in $\mathcal{Q}$ if and only if $t' \in \partial\mathcal{Q}$ and ab and tt' do not intersect or $t' \notin \partial\mathcal{Q}$ and ab and tt' do intersect, see Figure 2.

Let m_i be the number of vertices of $\mathcal{WV}_i$, $i = 0, \ldots, k-1$. According to Theorem 2, Algorithm 4 uses time $\sum_{i=0}^{k-1} O(m_in)$. Since the boundaries

Algorithm 4: Report a minimum link path from s to t

```
Input: A polygon $\mathcal{P}$ defined by its vertices $v_0, v_1, \ldots, v_{n-1}$ in CCW
       order. Two points $s$ and $t$ inside $\mathcal{P}$.
Output: A minimum link path $p_0 p_1 \ldots p_k$ from $s$ to $t$.
1  $d \leftarrow ss$   (* a segment consisting of one point *)
2  $\mathcal{P}' \leftarrow \mathcal{P}$
3  while $t$ is not visible from $d$
4      for each door $ab$ reported while computing $\mathcal{WV}_{\mathcal{P}'}(d)$
5          if the pocket $\mathcal{Q}$ with door $ab$ contains $t$
6              report the point on $d$ from where the beam containing $ab$
               starts
7              $d \leftarrow ab$
8              $\mathcal{P}' \leftarrow \mathcal{Q}$
9              continue from the loop at line 3
10 report a point on $d$ that can see $t$
11 report $t$
```

of the weak visibility polygons are disjoint except for one common edge between $\mathcal{WV}_i$ and $\mathcal{WV}_{i+1}$, we have that $\sum_{i=0}^{k-1} m_i = O(n)$, so the running time is $O(n^2)$ in total. □

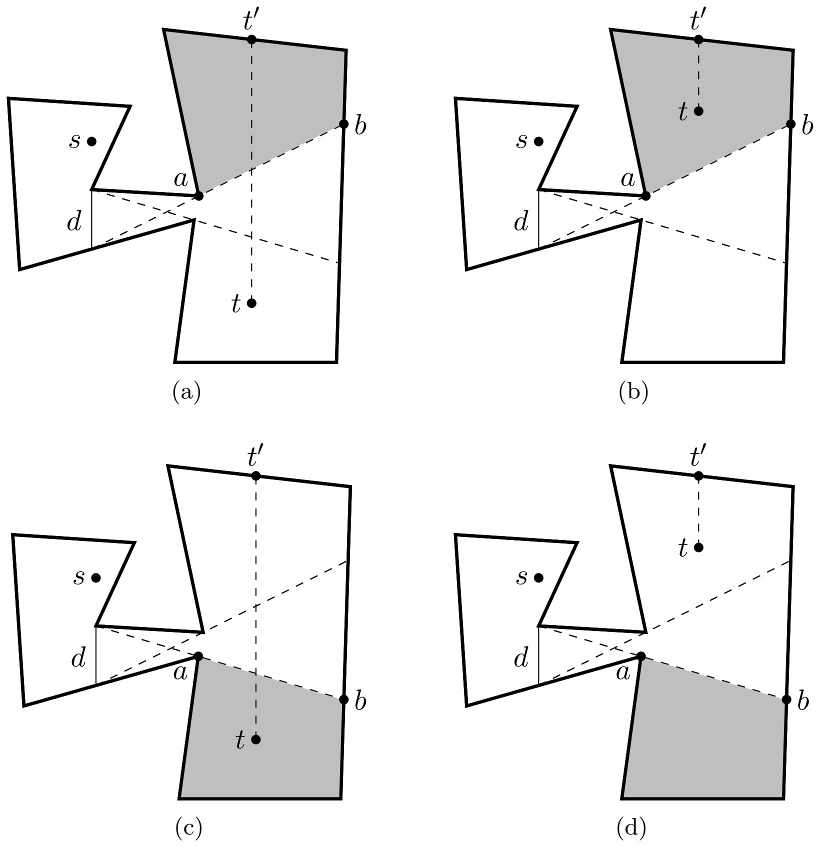


Figure 2. Figures showing how to decide in which pocket t is in Algorithm 4, assuming that t is not visible from d. The door d divides the polygon into two, $\mathcal{P}'$ is on the right-hand side of d. The visibility polygon $\mathcal{WV}_{\mathcal{P}'}(d)$ has two pockets. In each of these figures, the grey area is the pocket $\mathcal{Q}$ considered in line 5 of the algorithm. t is in $\mathcal{Q}$ if and only if $t' \in \partial\mathcal{Q}$ and ab and tt' do not intersect or $t' \notin \partial\mathcal{Q}$ and ab and tt' do intersect. This is the case in figures (b) and (c), respectively.

5. The Circular Visibility Region

5.1. The visibility polygon from a point in a simple polygon

One of the most fundamental visibility problems in the plane is to compute the visibility polygon $\mathcal{LV}_{\mathcal{P}}(p)$ given a point p inside a simple polygon $\mathcal{P}$. A point q in $\mathcal{P}$ is *linearly visible* from p if the line segment pq is contained in $\mathcal{P}$. The *visibility polygon* $\mathcal{LV}_{\mathcal{P}}(p)$ is the set of all points in $\mathcal{P}$ linearly visible from p, see Figure 1(a). When $\mathcal{P}$ and p are clear from the context, we might omit them and just write $\mathcal{LV}$.

It is well known that $\mathcal{LV}$ is a polygon, and that a vertex of $\mathcal{LV}$ is either a vertex of $\mathcal{P}$ visible from p or an intersection point between an edge of $\mathcal{P}$ and a line going through p [7]. The boundary of $\mathcal{LV}$ consists of chains of $\partial\mathcal{P}$ and chords connecting the chains. Each chord d on $\partial\mathcal{LV}$ separates a region $\mathcal{Q} \subseteq \mathcal{P}$ from $\mathcal{LV}$, so that no point in $\mathcal{Q}$ is visible from p. A *pocket* is such a region $\mathcal{Q}$, and the chord d is called the *door* to the pocket. A door has the form rr', where r is a reflex vertex of $\mathcal{P}$ and r' is the point where $\overrightarrow{pr}$ leaves $\mathcal{P}$. A pocket is a *left pocket* if it is on the left-hand side of its door. Otherwise, it is a *right pocket*.

The problem of computing $\mathcal{LV}$ was first described as the "hidden-line problem" by Freeman and Loutrel in 1967 [14]. They gave an $O(n^2)$ algorithm for the slightly more general problem where p is also allowed to be outside $\mathcal{P}$ and n is the number of vertices of $\mathcal{P}$. Joe and Simpson [18] described an $O(n)$-time and -space algorithm. Barba et al. [7] recently devised an output sensitive algorithm that computes the visibility polygon using constant workspace in $O(nd + n)$ time, where d is the number of doors of $\mathcal{LV}$.[2] It is clear that $d = O(n)$ since one of the endpoints of a door is a reflex vertex of $\mathcal{P}$. Actually, though the algorithm of Freeman and Loutrel is not presented as a constant-workspace algorithm, the general strategy of their algorithm closely resembles that of the algorithm of Barba et al. An algorithm using $O(\log r)$ variables that runs in $O(n \log r)$ randomized expected time or $O(n \log^2 r)$ deterministic time, where r is the number of reflex vertices of $\mathcal{P}$, was also presented by Barba et al.

[2] Actually, the authors claim the running time to be $O(nd)$, but it is clear that the algorithm uses $\Omega(n)$ time if $d = 0$, so the claimed bound is slightly incorrect.

5.2. Computing the circular visibility region from a point in a simple polygon

Circular visibility is defined analogously to linear visibility: q is *circularly visible* from p if there exists a circular arc $\widehat{pq}$ from p to q contained in $\mathcal{P}$, the arc can be CW (clockwise) or CCW (counterclockwise). The arc $\widehat{pq}$ is called a *visibility arc*. Notice that a circular arc $\widehat{pq}$ is not uniquely defined from its notation, since there are infinitely many circular arcs going through p and q. We merely use $\widehat{pq}$ as the name of an arc defined by the context. We sometimes write $\widehat{abc}$ to denote the unique arc starting at a, going through b, and ending at c.

A *maximal visibility arc* is a visibility arc $\widehat{pq}$ such that no visibility arc $\widehat{pq'}$ exists where $q \neq q'$ and $\widehat{pq}$ is contained in $\widehat{pq'}$. Hence, if $\widehat{pq}$ is a maximal visibility arc, we know that $\widehat{pq}$ is a full circle or that q is on the boundary of $\mathcal{P}$. Since a line segment may be considered an arc of a circle with infinite radius, q is circularly visible from p if it is linearly visible, but the opposite is not true in general, since one can “look around corners” using circular visibility.

We often have to find the maximal visibility arc $\widehat{pq}$ given a the start point p and the center of the circle containing $\widehat{pq}$, i.e. we want to find the point q where the circle exits $\mathcal{P}$ when followed CW or CCW from p. The point q is the first proper intersection point between $\partial\mathcal{P}$ and the circle. In order to find intersection points between line segments and a circle, we need access to the square-root function in $O(1)$ time, since we need to solve second-order equations.

The *circular visibility region* $\mathcal{CV}_{\mathcal{P}}(p)$ is the set of all points in $\mathcal{P}$ circularly visible from p, see Figure 1(b). We write $\mathcal{CV}$ if $\mathcal{P}$ and p are clear from the context. We see that the visibility polygon is a subset of the circular visibility region, i.e. $\mathcal{LV} \subseteq \mathcal{CV}$.

We use the word ‘region’ and not ‘polygon’ since, as we shall see, the boundary of the circular visibility region consists of line segments and circular arcs. Chou et al. [10] showed how to compute the circular visibility region in $O(n)$ time using an auxiliary data structure called a *circular visibility diagram* of $\mathcal{P}$. Chou and Woo [11] showed how to compute the circular visibility diagram in $O(n)$ time. Hence, the two papers together yield an algorithm to compute the circular visibility region in $O(n)$ time. That method seems to be of little use in the constant-workspace model, since the circular visibility diagram takes up $\Omega(n)$ space. Here we present an $O(n^2)$-time algorithm for computing the circular visibility region using constant workspace. As far as we know, there is no literature about computing circular visibility regions in any memory-constrained model. In addition to the requirement of using constant workspace, we think our algorithm has the advantage of being much simpler than the before mentioned, since we avoid the rather cumbersome computation of the circular visibility diagram.

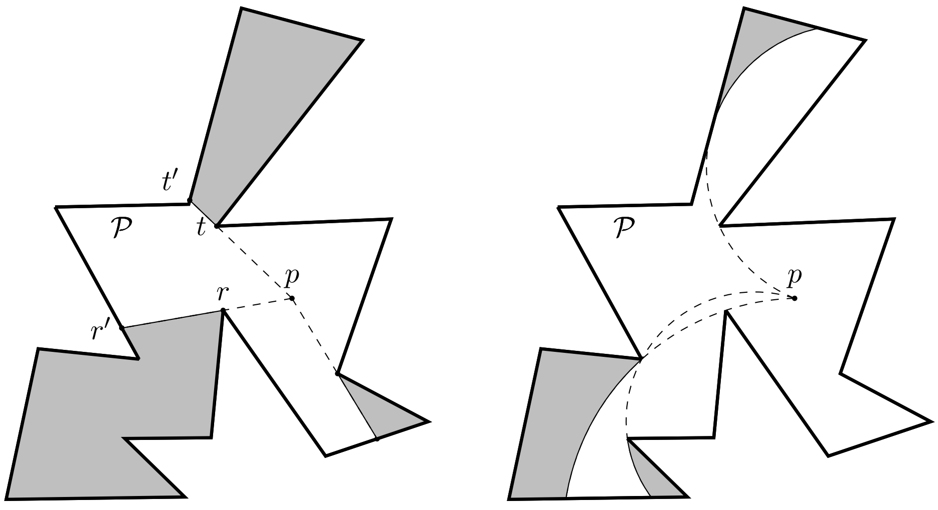


Figure 1. Comparison of $\mathcal{LV}$ and $\mathcal{CV}$ for the same polygon $\mathcal{P}$ and point $p \in \mathcal{P}$. (a) The visibility polygon $\mathcal{LV}$ of p in $\mathcal{P}$. The pockets are shown in grey. The segment rr' is the door of a left pocket and tt' is the door of a right pocket. (b) The circular visibility region $\mathcal{CV}$ of p in $\mathcal{P}$. The regions of $\mathcal{P}$ not visible from p are grey. Maximal visibility arcs containing caps are drawn.

The boundary of the circular visibility region is to be reported in CCW order. The main idea is to use the algorithm of Barba et al. [7] to compute the visibility polygon of p. Their algorithm reports the boundary of $\mathcal{LV}$ in CCW order. Each time a door of a pocket of $\mathcal{LV}$ is reported, we compute the circularly visible part of the pocket. We only explain how to deal with left pockets, since the method for right pockets is symmetric. It is clear that only CCW visibility arcs can enter a left pocket and only CW arcs can enter a right pocket, so in the following we only consider CCW visibility arcs.

A visibility arc $\widehat{pq}$ has two sides: the *convex side* is the side towards the interior of the circle containing $\widehat{pq}$ and the *concave side* is the side towards the exterior of the circle. A point u on the boundary $\partial\mathcal{P}$ is a *support* of $\widehat{pq}$ if u is on $\widehat{pq}$ but $\partial\mathcal{P}$ is not crossing $\widehat{pq}$ at u, such that $\partial\mathcal{P}$ is on the same side of $\widehat{pq}$ before and after u. If $\partial\mathcal{P}$ is on the convex side, u is a *convex support* of $\widehat{pq}$, and if $\partial\mathcal{P}$ is on the concave side, u is a *concave support* of $\widehat{pq}$. It is seen that a convex support must be a reflex vertex of $\mathcal{P}$, whereas a concave support is either a reflex vertex or a point on an edge of $\mathcal{P}$ tangential to $\widehat{pq}$. See Figure 2.

The boundary of a circular visibility region consists of chains of $\mathcal{P}$ and parts of visibility arcs. The parts of visibility arcs are called *caps.* A cap separates the circular visibility region from a region not circularly visible from p. Such a region is called a *deficiency.* In Figure 1(b), the deficiencies are the grey areas. The deficiencies are the connected components of $\mathcal{P} \setminus \mathcal{CV}$.

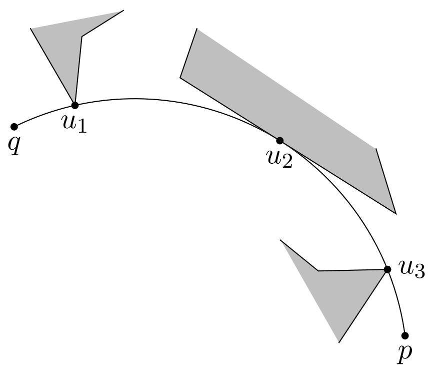


Figure 2. A CCW arc $\widehat{pq}$ with concave supports u_1 and u_2 and convex support u_3. The grey areas indicate where the interior of the polygon is relative to the supports.

The boundary of a deficiency consists of some chain of $\partial\mathcal{P}$ and one arc, which is a cap. Let $\widehat{pq}$ be a maximal visibility arc. Assume that $\widehat{uq}$ is a maximal subset of $\widehat{pq}$ such that no point on the convex side of $\widehat{uq}$ is circularly visible from p. Then $\widehat{uq}$ is a *convex cap* and the region to the convex side of $\widehat{uq}$ is a *convex deficiency.* A *concave cap* and *concave deficiency* are defined analogously. We compute the circular visibility region by computing all the visibility arcs that contain a cap in CCW order around $\partial\mathcal{P}$.

The following lemma characterizes the visibility arcs containing caps by means of their supports. The lemma is due to Chou et al. [10]. Notice the similarity with Lemma 6.

Lemma 11. *Let $\widehat{pq}$ be a maximal visibility arc.*

(1) $\widehat{pq}$ contains the convex cap $\widehat{s_2q}$ if and only if $\widehat{pq}$ has concave support s_1 and convex support s_2 in that order when following $\widehat{pq}$ from p.

(2) $\widehat{pq}$ contains the concave cap $\widehat{s_4q}$ if and only if $\widehat{pq}$ has convex support s_3 and concave support s_4 in that order when following $\widehat{pq}$ from p.

Proof. We prove (1); the case (2) follows analogously. If $\widehat{pq}$ has no supports, there exist visibility arcs containing all points in a neighborhood around $\widehat{pq}$, so $\widehat{pq}$ does not contain a cap. If $\widehat{pq}$ has a concave support s_1 but no convex support after s_1, there exist visibility arcs containing all points in a neighborhood on the convex side of $\widehat{s_1q}\subseteq\widehat{pq}$, so there cannot be a convex cap. The same is true if $\widehat{pq}$ has convex support s_2 but no concave support before s_2, see Figure 3(a).

Now assume that $\widehat{pq}$ has supports s_1 and s_2 as defined in the lemma. See Figure 3(b). Assume that some point q' on the convex side of $\widehat{s_2q}$ is circularly visible from p, that is, there is a visibility arc $\widehat{pq'}$. The arc $\widehat{s_2q}$ divides $\mathcal{P}$ into two parts, one containing p and one containing q'. Since $\widehat{pq'}$ is a continuous curve from p to q', it must intersect $\widehat{s_2q}$ at some point r. Likewise, $\widehat{s_1s_2}\subset\widehat{pq}$ divides $\mathcal{P}$ into two parts, one containing p and one

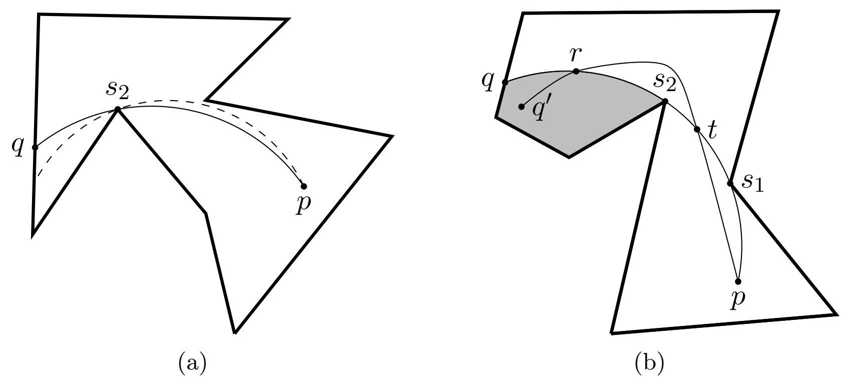


Figure 3. Illustrations for the proof of Lemma 11. (a) When there is no concave support before s_2 on arc $\widehat{pq}$, arc $\widehat{s_2q}$ is no cap. (b) A visibility arc hitting a point on the convex side of $\widehat{s_2q}$ does not exist.

containing q', so $\widehat{pq'}$ must intersect $\widehat{s_1s_2}$ at some point t. Since the arcs $\widehat{pq}$ and $\widehat{pq'}$ share the three points p, t, and r, they must be on the same circle, so no point on the convex side of $\widehat{s_2q}$ is circularly visible from p. Therefore, $\widehat{s_2q}$ is a convex cap. □

From this lemma, we immediately get a naive $O(n^3)$-time algorithm using constant workspace written in pseudocode as Algorithm 5 and explained in words in the following.

Let $\mathcal{P}$ be given as the vertices $v_0, \ldots, v_{n-1}$ in CCW order. We traverse $\partial\mathcal{P}$ CCW from v_0v_1 while reporting the boundary of $\mathcal{CV}$. For each edge v_iv_{i+1}, we check if there is a maximal visibility arc $\widehat{pq}$ containing a cap $\widehat{sq}$ such that s is a point on v_iv_{i+1} and s is a support of $\widehat{pq}$. To check this, we traverse each edge v_jv_{j+1} of $\mathcal{P}$ to find each pair (s', s) such that there is an arc $\widehat{pq'}$ having concave (resp. convex) support $s' \in v_jv_{j+1}$ and convex (resp. concave) support $s \in v_iv_{i+1}$, in that order. So far, we know that $\widehat{pq'}$ is an arc with the desired supports, but it needs to be a visibility arc. Therefore, we find the maximal visibility arc $\widehat{pq}$ contained in the same circle as $\widehat{pq'}$ by traversing all of $\partial\mathcal{P}$ to find the point where the circle exits $\mathcal{P}$. We check if $\widehat{pq}$ contains s, in which case $\widehat{sq}$ is a cap. That is done in lines 4, 7, 10, and 15. We report the part of $\partial\mathcal{P}$ since the last found cap until the new cap $\widehat{sq}$, and the new cap $\widehat{sq}$. Lines 4–9 finds the caps ending on v_iv_{i+1}, whereas lines 10–19 finds the caps starting on v_iv_{i+1}.

For each edge v_iv_{i+1}, we use $O(n^2)$ time: we traverse each edge v_jv_{j+1} of $\mathcal{P}$ to find each pair (s', s) where $s' \in v_jv_{j+1}$ and $s \in v_iv_{i+1}$ such that there is an arc $\widehat{ps}$ with supports of opposite types s' and s. For each of these pairs, we traverse all of $\mathcal{P}$ to find the maximal visibility arc. Therefore, the algorithm runs in $O(n^3)$ time.

Algorithm 5: Report the circular visibility region $\mathcal{CV}_{\mathcal{P}}(p)$ in a naive way

```
Input: A polygon $\mathcal{P}$ defined by its vertices $v_0v_1 \ldots v_{n-1}$ and a point $p \in \mathcal{P}$.
Output: The boundary of the circular visibility region $\mathcal{CV}_{\mathcal{P}}(p)$ is reported.
1  $i \leftarrow 0$
2  while $i < n$ do
3      for each $j = 0, \ldots, n-1$
4          if there is a CCW arc $\widehat{pq'}$ with a concave support on $v_jv_{j+1}$ and
           convex support $v_i$ and the maximal visibility arc $\widehat{pq}$ contained in the
           same circle as $\widehat{pq'}$ contains $v_i$
5              report the boundary of $\mathcal{P}$ since the last found cap to $q$
6              report the convex cap $\widehat{qv_i}$
7          else if there is a CW arc $\widehat{pq'}$ with convex support $v_j$ and a concave
           support $s_4$ on $v_iv_{i+1}$ and the maximal visibility arc $\widehat{pq}$ contained in
           the same circle as $\widehat{pq'}$ contains $s_4$
8              report the boundary of $\mathcal{P}$ since the last found cap to $q$
9              report the concave cap $\widehat{qs_4}$
10         if there is a CW arc $\widehat{pq'}$ with a concave support on $v_jv_{j+1}$ and convex
           support $v_{i+1}$ and the maximal visibility arc $\widehat{pq}$ contained in the same
           circle as $\widehat{pq'}$ contains $v_{i+1}$
11             report the boundary of $\mathcal{P}$ since the last found cap to $v_{i+1}$
12             report the convex cap $\widehat{v_{i+1}q}$
13             Let $i$ be increased so that $q \in v_iv_{i+1}$
14             continue from the loop at line 2
15         else if there is a CCW arc $\widehat{pq'}$ with convex support $v_j$ and a concave
           support $s_4$ on $v_iv_{i+1}$ and the maximal visibility arc $\widehat{pq}$ contained in
           the same circle as $\widehat{pq'}$ contains $s_4$
16             report the boundary of $\mathcal{P}$ since the last found cap to $s_4$
17             report the concave cap $\widehat{s_4q}$
18             Let $i$ be increased so that $q \in v_iv_{i+1}$
19             continue from the loop at line 2
20     $i \leftarrow i + 1$
21 report the boundary between the last and first found caps
```

We can make a more efficient algorithm by finding stronger necessary conditions that a vertex is a support of a visibility arc containing a cap. The following lemma characterizes the caps in a more detailed way by describing which points on $\partial\mathcal{P}$ can possibly be the first support of a CCW maximal visibility arc containing a cap.

Lemma 12. *Let $\widehat{pq}$ be a CCW maximal visibility arc where q is in a left pocket with door rr'.*

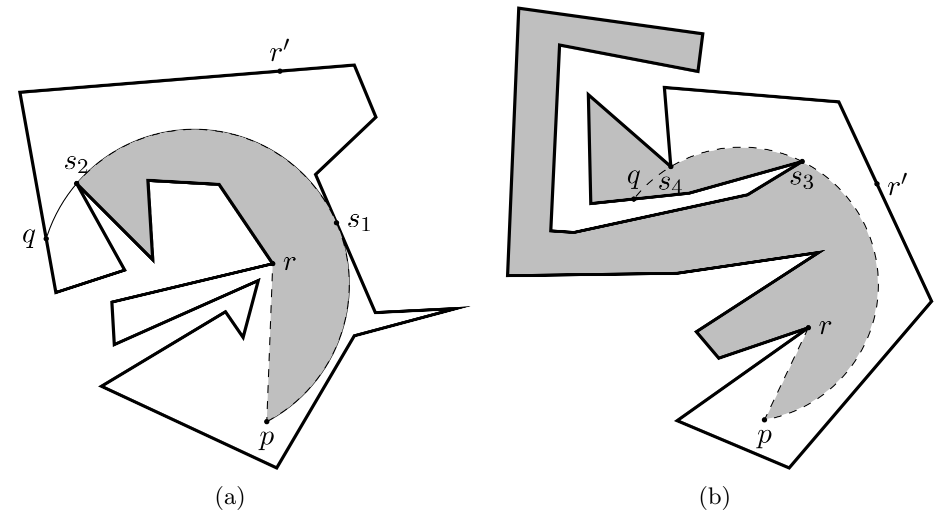


Figure 4. Two polygons with maximal CCW visibility arcs $\widehat{pq}$ and doors rr' to a left pocket. The regions separated from the rest of the polygon by segment pr and $\widehat{ps_2}$ or $\widehat{ps_4}$ are grey. (a) Arc $\widehat{pq}$ contains a convex cap $\widehat{s_2q}$. (b) Arc $\widehat{pq}$ contains a concave cap $\widehat{s_4q}$.

(1) If $\widehat{pq}$ has concave support s_1 and convex support s_2, then $s_1 \in \mathcal{P}(r, s_2)$.

(2) If $\widehat{pq}$ has convex support s_3 and concave support s_4, then $s_3 \in \mathcal{P}(s_4, r)$.

Proof. Consider the case (1). The segment rp and the arc $\widehat{ps_2} \subset \widehat{pq}$ separates a region from the rest of $\mathcal{P}$, see Figure 4(a). The boundary of the region is a closed, simple curve consisting of $\mathcal{P}(s_2, r)$, rp, and $\widehat{ps_2}$. The region is on the convex side of $\widehat{ps_2}$, so no vertex in $\mathcal{P}(s_2, r)$ can be a concave support. Therefore, the concave support s_1 must be in $\mathcal{P}(r, s_2)$.

Figure 4(b) shows case (2), where the regions separated from the rest of $\mathcal{P}$ by segment pr and arc $\widehat{ps_4}$ are grey. The convex support s_3 has to be on $\mathcal{P}(s_4, r)$, since all other points are on the concave side of $\widehat{pq}$. □

Assume we are in case (1) of Lemma 12. Let us follow $\partial\mathcal{P}$ CCW from r. We partition $\mathcal{P}(r, s_2)$ into three different types:

1a. When $\partial\mathcal{P}$ crosses the segment ps_2 from left to right it becomes type 1a. Also, $\mathcal{P}(r, s_2)$ starts as type 1a.

2a. When $\partial\mathcal{P}$ crosses the line $\overleftrightarrow{ps_2}$ from right to left it becomes type 2a.

3a. When $\partial\mathcal{P}$ crosses the line $\overleftrightarrow{ps_2}$, but not the segment ps_2, from left to right, it becomes type 3a.

See Figure 5 where each type has a different color. The polygon in Figure 5 is identical (up to scaling) to that in 4(a). The following lemma says that we only have to consider the part of $\mathcal{P}(r, s_2)$ of type 3a.

Lemma 13. *Let $\widehat{pq}$ be a CCW maximal visibility arc where q is in a left pocket with door rr'. If $\widehat{pq}$ has concave support s_1 and convex support s_2, then s_1 is a point of type 3a.*

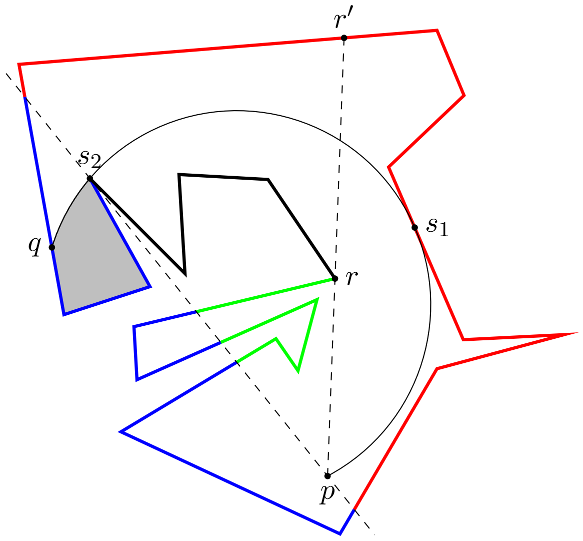


Figure 5. The partition of $\mathcal{P}(r, s_2)$ into types 1a (green), 2a (blue), and 3a (red).

Proof. Let us follow $\partial\mathcal{P}$ CCW from r. When $\partial\mathcal{P}$ is of type 1a, it is moving around inside the wedge of the segments ps_2 and pr. It cannot exit the wedge through the segment pr, because then r would not be linearly visible from p. A point of type 1a cannot be the concave support of $\widehat{pq}$, because $\widehat{pq}$ needs to have r on its convex side (as noted in the proof of Lemma 12), and r is on the concave side of any arc going through a point of type 1a. A point on $\partial\mathcal{P}$ of type 2a is to the left of the line $\overleftrightarrow{ps_2}$, so it cannot be the concave support of a CCW arc $\widehat{ps_2}$. Therefore, s_1 must be of type 3a. $\square$

Given three points a, b, and c which are not collinear, let $\mathcal{C}(a, b, c)$ be the centre of their circumcircle. If one of the arguments is `NULL`, so is $\mathcal{C}(a, b, c)$. It is well-known that the centre of the circumcircle is the common intersection point of the perpendicular bisectors of triangle abc, which can be determined in constant time using elementary vector computations. All the arcs passing through two points a and c have centres on the perpendicular bisector of the segment ac. That means that $\mathcal{C}(a, b, c)$ has the form $\frac{a+c}{2} + t \cdot \widehat{c-a}$ for some $t \in \mathbb{R}$. Here, $\widehat{v}$ of some vector $v \in \mathbb{R}^2$ is the CCW rotation of v by 90°. We say that $\mathcal{C}(a, b_1, c)$ is further to the left (resp. to the right) than $\mathcal{C}(a, b_2, c)$ if the t-value corresponding to b_1 is larger (resp. smaller) than the t-value corresponding to b_2.

The locus of centres of circles that are tangential to a segment e and pass through a point a is a fragment of a parabola. Let $\mathcal{D}(a, b, e)$ be the contact point on e of the circle tangential to e and passing through a and b. If the circle does not exist, let $\mathcal{D}(a, b, e) =$ `NULL`. See Figure 6, where we explain how to compute $\mathcal{D}(a, b, e)$ in $O(1)$ time if we have access to the square-root function in $O(1)$ time.

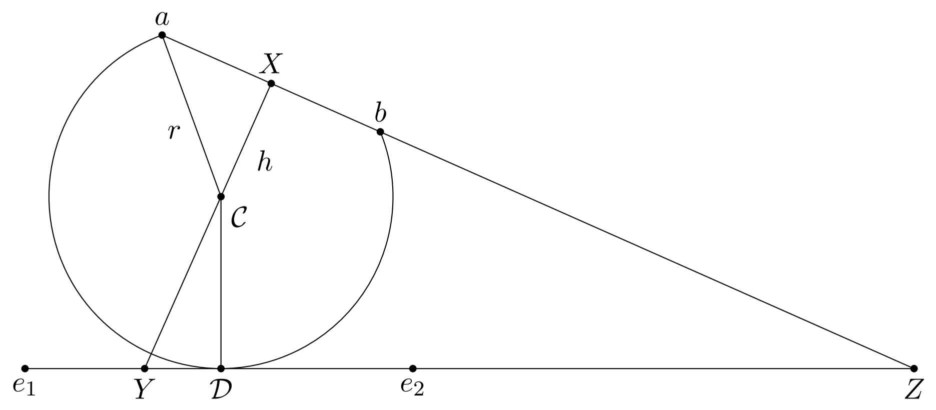


Figure 6. How to find the contact point $\mathcal{D} = \mathcal{D}(a, b, e)$ on the edge $e = e_1e_2$ of the arc going through a and b tangential to e: Assume that ab and e are not parallel, the other case is simpler. Let X be the middle point of segment ab, Y be the intersection point between the perpendicular bisector of ab and the line through e, and Z be the intersection point between the lines through ab and e. Let $r = |a\mathcal{C}| = |b\mathcal{C}| = |\mathcal{D}\mathcal{C}|$ be the radius of the arc and let $h = |X\mathcal{C}|$, here $|\cdot|$ is the length function. We know that $r^2 = h^2 + |aX|^2$ $(*)$. We see that the triangles XYZ and $\mathcal{D}Y\mathcal{C}$ are similar. Therefore $r = |\mathcal{C}Y| \cdot |XZ|/|YZ|$ $(**)$. We also know that $|\mathcal{C}Y| = |XY| - h$. That gives two equations $(*)$ and $(**)$ with the unknowns r and h. Isolating one of the unknowns in equation $(**)$ and inserting in equation $(*)$ gives a second-order equation in the other unknown. When the relevant solution is found, $\mathcal{C}$ and $\mathcal{D}$ can be constructed, and we can check if $\mathcal{D} \in e$.

The following lemma uniquely determines the concave support of a CCW maximal visibility arc with a convex cap given its convex support. That makes it possible to find the arc in $O(n)$ time if it exists.

Lemma 14. *Let $\widehat{pq}$ be a CCW maximal visibility arc where q is in a left pocket with door rr'. If $\widehat{pq}$ has concave support s_1 and convex support s_2, then s_1 is a point in $\mathcal{P}(r, s_2)$ of type 3a such that $\mathcal{C}(p, s_1, s_2)$ is as far to the left as possible.*

Proof. Assume that there is a point s on $\mathcal{P}(r, s_2)$ such that $\mathcal{C}(p, s, s_2)$ is to the left of $\mathcal{C}(p, s_1, s_2)$, see Figure 7. In that case s is contained in the circumcircle through p, s_1, and s_2. Therefore $\partial\mathcal{P}$ crosses the arc $\widehat{ps_2} \subset \widehat{pq}$ to get to s, so $\widehat{pq}$ is no visibility arc. □

Based on these observations, we get an $O(n)$-time algorithm to check if a convex cap $\widehat{s_2q}$ exists given a vertex s_2 in a left pocket with door rr'. The function `FindConvexCap`(s_2, rr') in Algorithm 6 returns q if there is a maximal visibility arc $\widehat{pq}$ containing a convex cap $\widehat{s_2q}$, otherwise `NULL` is returned. The algorithm runs through $\mathcal{P}(r, s_2)$. We keep a variable s_1, which is the point such that $\mathcal{C}(p, s_1, s_2)$ is furthest to the left of all the points on $\mathcal{P}(r, s_2)$ visited so far. Each time we visit a segment u_1u_2 of type 3a, we check if there is a point s on u_1u_2 such that $\mathcal{C}(p, s, s_2)$ is further to the left than $\mathcal{C}(p, s_1, s_2)$, in which case we update $s_1 \leftarrow s$. We know that we only need to check points s such that $\mathcal{D}(p, s_1, u_1u_2)$ is well defined (line 6) and

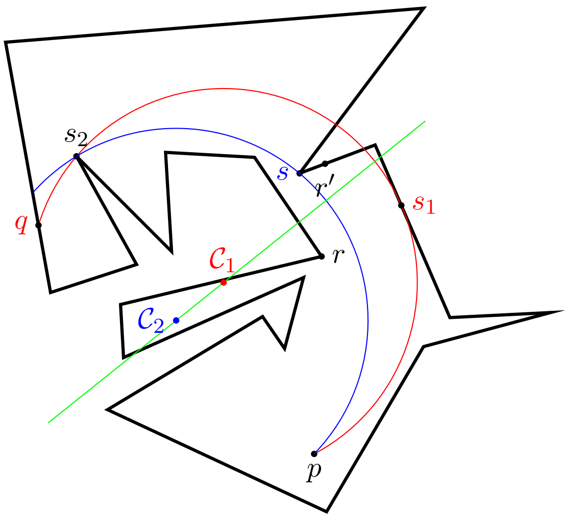


Figure 7. Illustration for the proof of Lemma 14. The center $\mathcal{C}_2$ of the circle through p, s, and s_2 (blue) is further to the left than the center $\mathcal{C}_1$ of the circle through p, s_1, and s_2 (red). The perpendicular bisector of ps_2 is green.

vertex u_2 (line 3). Notice that some conditions in lines 6 and 3 assume that $s_1 \neq \texttt{NULL}$. If $s_1 = \texttt{NULL}$, those tests are always true. We only check one endpoint since u_1 was checked in the previous iteration. To execute line 10, we need to run through all of $\partial\mathcal{P}$ to find the exit point q. In line 11, we check that $\widehat{pq}$ contains $\widehat{ps_2}$ and that s_1 and s_2 are indeed concave and convex supports, respectively.

Now assume we are in case (2) of Lemma 12. Consider a CCW maximal visibility arc $\widehat{pq}$ containing a concave cap $\widehat{s_4q}$ in a pocket with door rr'. Remember that the convex support s_3 of the arc satisfies $s_3 \in \mathcal{P}(s_4, r)$. We make a similar characterization of the possible supports s_3, but the situation is not completely analogous. The concave support s_4 is either a vertex u_2 or a point on some edge u_1u_2. We partition $\mathcal{P}(u_1, r)$ into two different types while following the chain CCW from u_1.

1b. When $\partial\mathcal{P}$ crosses the segment pu_2 from right to left, it becomes type 1b. Also, $\mathcal{P}(u_1, r)$ starts as type 1b.

2b. When $\partial\mathcal{P}$ crosses the segment pu_2 from left to right, it becomes type 2b.

See Figure 8, where the two types are drawn in blue and red, respectively. The polygon in Figure 8 is identical (up to scaling) to that in Figure 4(b). According to the following lemma, we only need to consider vertices on the chain $\mathcal{P}(s_4, r)$ of type 2b.

Lemma 15. *Let $\widehat{pq}$ be a CCW maximal visibility arc. If $\widehat{pq}$ has convex support s_3 and concave support s_4, then s_3 is a point of type 2b.*

Proof. Let u_1u_2 be the segment containing s_4, $s_4 \neq u_1$. A convex support is in the region bounded by pu_2, $\widehat{ps_4} \subset \widehat{pq}$ and the possibly degenerated segment

Algorithm 6: `FindConvexCap`(s_2, rr')

```
Input: A polygon $\mathcal{P}$ defined by its boundary $\partial\mathcal{P}$, a point $p \in \mathcal{P}$, the
       door $rr'$ of a left pocket in $\mathcal{P}$, and a vertex $s_2$ in $\mathcal{P}(r', r)$.
Output: $q$ if there is a maximal visibility arc $\widehat{pq}$ containing a convex
        cap $\widehat{s_2q}$, otherwise NULL.
1  if $s_2$ is not a reflex vertex
2      return NULL
3  $s_1 \leftarrow$ NULL
4  for each segment $u_1u_2$ in $\mathcal{P}(r, s_2)$ of type 3a
5      $s \leftarrow \mathcal{D}(p, s_2, u_1u_2)$
6      if $s \neq$ NULL and $\mathcal{C}(p, s, s_2)$ is further to the left than $\mathcal{C}(p, s_1, s_2)$
7          $s_1 \leftarrow s$
8      if $\mathcal{C}(p, u_2, s_2)$ is further to the left than $\mathcal{C}(p, s_1, s_2)$
9          $s_1 \leftarrow u_2$
10 Let $q$ be the point where the circle containing $\widehat{ps_2}$ exits $\mathcal{P}$ followed
   CCW from $p$
11 if $\widehat{pq}$ has concave support $s_1$ and convex support $s_2$
12     return $q$
13 else
14     return NULL
```

s_4u_2. Therefore $\partial\mathcal{P}$ must cross pu_2 from left to right in order to get to the convex support s_3. Hence, s_3 is of type 2b. □

The following lemma says that we the only possible convex support of an arc containing a concave cap is the one of type 2b that "presses" the arc as far to the right as possible. There are in a sense two different cases depending on whether the concave support is a vertex or an interior point on a segment.

Lemma 16. *Let $\widehat{pq}$ be a CCW maximal visibility arc where q is in a pocket with door rr'. Assume that $\widehat{pq}$ has convex support s_3 and concave support s_4, where s_4 is on the segment u_1u_2, $s_4 \neq u_1$. If there is no vertex v_j on $\mathcal{P}(u_2, r)$ of type 2b such that $\mathcal{D}(p, v_j, u_1u_2)$ is defined, then $s_4 = u_2$ and s_3 is the vertex on $\mathcal{P}(u_2, r)$ such that $\mathcal{C}(p, s_3, s_4)$ is as far to the right as possible. Otherwise, s_3 is the vertex on $\mathcal{P}(u_2, r)$ such that $s_4 = \mathcal{D}(p, s_3, u_1u_2)$ is as close to u_1 as possible.*

Proof. Assume first that there is no vertex v_j on $\mathcal{P}(u_2, r)$ such that $\mathcal{D}(p, v_j, u_1u_2)$ is defined. We must have $s_4 = u_2$ since $s_4 \neq u_1$. If there is a vertex v_j such that $\mathcal{C}(p, v_j, s_4)$ is further to the right than $\mathcal{C}(p, s_3, s_4)$, then $\partial\mathcal{P}$ must cross $\widehat{ps_4} \subset \widehat{pq}$ to get to v_j, so $\widehat{pq}$ is no visibility arc. If there is a vertex v_j such that $\mathcal{D}(p, v_j, u_1u_2)$ is defined, s_4 cannot be u_2, since the arc $\widehat{pv_ju_2}$ is outside $\mathcal{P}$ just before it reaches u_2, see Figure 9(a). Therefore

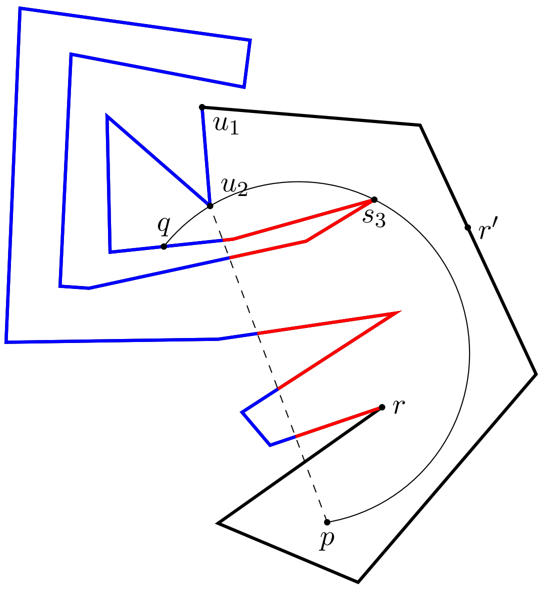


Figure 8. The partition of $\mathcal{P}(u_1, r)$ into types 1b (blue), 2b (red).

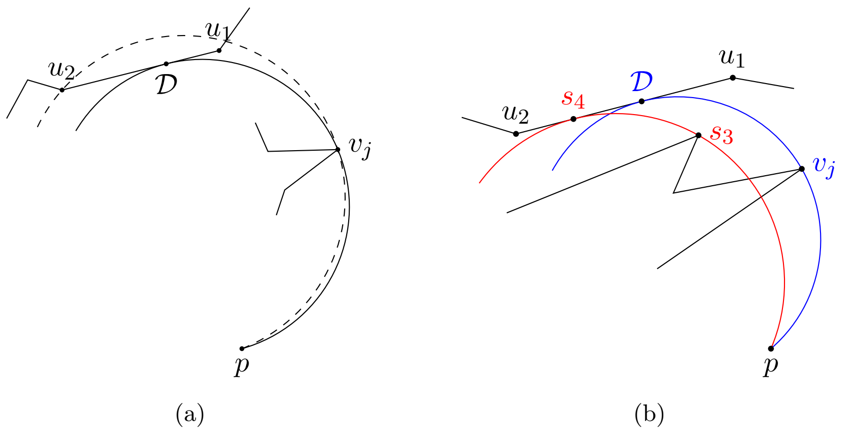


Figure 9. Situations from the proof of Lemma 16. (a) When $\mathcal{D} = \mathcal{D}(p, v_j, u_1u_2)$ is defined for some vertex v_j, u_2 cannot be a concave support. (b) If $\mathcal{D} = \mathcal{D}(p, v_j, u_1u_2)$ is closer to u_1 than $s_4 = \mathcal{D}(p, s_3, u_1u_2)$, $\widehat{ps_4}$ is no visibility arc.

s_3 must be the vertex that pushes the contact point $s_4 = \mathcal{D}(p, s_3, u_1u_2)$ as close to u_1 as possible, since otherwise $\partial\mathcal{P}$ crosses $\widehat{ps_4}$, see Figure 9(b). $\square$

The function $\texttt{FindConcaveCap}(u_1u_2, rr')$ in Algorithm 7 checks if there is a concave cap $\widehat{s_4q}$ where s_4 is a point on a given segment u_1u_2 in $\mathcal{P}(r', r)$. The algorithm runs through the reflex vertices v_j on $\mathcal{P}(u_2, r)$ and finds the one that makes $\mathcal{C}(p, v_j, u_2)$ become furthest to the right or $\mathcal{D}(p, v_j, u_1u_2)$

become closest to u_1. Notice that some conditions in 3 and 6 assume that $s_3 \neq$ `NULL`. If $s_3 =$ `NULL`, those tests are always true.

Algorithm 7: `FindConcaveCap`(u_1u_2, rr')

```
Input: A polygon $\mathcal{P}$ defined by its boundary $\partial\mathcal{P}$, a point $p \in \mathcal{P}$, the
       door $rr'$ of a left pocket in $\mathcal{P}$, and a segment $u_1u_2$ in $\mathcal{P}(r', r)$.
Output: $(q, s_4)$ if there is a maximal visibility arc $\widehat{pq}$ containing a
        concave cap $\widehat{s_4q}$ where $s_4$ is on $u_1u_2$, otherwise NULL.
1  $s_3 \leftarrow$ NULL,   $s_4 \leftarrow u_2$
2  for each reflex vertex $v_j$ in $\mathcal{P}(u_2, r)$ of type 2a
3      if $s_4 = u_2$ and $\mathcal{C}(p, v_j, u_2)$ is further to the right than $\mathcal{C}(p, s_3, s_4)$
4          $s_3 \leftarrow v_j$
5      $s \leftarrow \mathcal{D}(p, v_j, u_1u_2)$
6      if $s \neq$ NULL and $s$ is contained in $s_4u_1$
7          $s_3 \leftarrow v_j$,   $s_4 \leftarrow s$
8  Let $q$ be the point where the circle containing $\widehat{ps_4}$ exits $\mathcal{P}$ followed
   CCW from $p$
9  if $\widehat{pq}$ has convex support $s_3$ and concave support $s_4$
10     return $(q, s_4)$
11 else
12     return NULL
```

Theorem 4. *The circular visibility region $\mathcal{CV}_{\mathcal{P}}(p)$ can be reported in $O(n^2)$ time using constant workspace, where n is the number of vertices of $\mathcal{P}$.*

Proof. Algorithm 8 reports the circular visibility region using `FindConvexCap` and `FindConcaveCap`. For each pocket of $\mathcal{LV}_{\mathcal{P}}(p)$ found by the algorithm by Barba et al. [7], we run through the boundary of the pocket. For each vertex u_2, we check if there is a convex cap with convex support u_2. Similarly, for each segment u_1u_2, we check if there is a concave cap with the concave support on u_1u_2. Whenever a cap is found, we report the chain of $\mathcal{P}$ visited since the last found cap as well as the new cap. We need $O(n)$ time for each call to `FindConvexCap` and `FindConcaveCap`. We make $O(n)$ such calls, giving $O(n^2)$ time. The algorithm to compute $\mathcal{LV}$ takes $O(nd + n)$ time, so the total time is also $O(n^2)$. □

5.3. The parabolic visibility region from a point in a simple polygon

In this section we briefly demonstrate that the techniques of the preceding section can be applied to compute the region of points in a polygon $\mathcal{P}$ parabolically visible from a point $p \in \mathcal{P}$. We shall not prove the correctness of the method rigorously, but merely sketch the idea. Without loss

Algorithm 8: Report $\mathcal{CV}_{\mathcal{P}}(p)$ using constant workspace

```
Input: A polygon $\mathcal{P}$ defined by its boundary $\partial\mathcal{P}$ and a point $p \in \mathcal{P}$.
Output: The boundary of $\mathcal{CV}_{\mathcal{P}}(p)$ in CCW order is reported.
1  for each pocket of $\mathcal{LV}_{\mathcal{P}}(p)$
2      if the pocket is a left pocket with door $rr'$
3          $u_1 \leftarrow r'$
4          Let $u_2$ be the vertex following $u_1$ CCW on $\partial\mathcal{P}$
5          repeat
6              $q \leftarrow$ FindConvexCap$(u_2, rr')$
7              if $q \neq$ NULL
8                  report the chain of $\mathcal{P}$ from the last found cap to $q$
9                  report the cap $\overset{\frown}{qu_2}$
10                 Let $u_1u_2$ be the CCW next segment on $\partial\mathcal{P}$
11                 if $u_1 \neq r$
12                     continue from the loop at line 5
13                 else
14                     continue from the loop at line 1
15             $(q, s_4) \leftarrow$ FindConcaveCap$(u_1u_2, rr')$
16             if $(q, s_4) \neq$ NULL
17                 report the chain of $\mathcal{P}$ from the last found cap to $s_4$
18                 report the cap $\overset{\frown}{s_4q}$
19                 $u_1 \leftarrow q$
20                 Let $u_2$ be the vertex following $u_1$ CCW on $\partial\mathcal{P}$
21                 continue from the loop at line 5
22             Let $u_1u_2$ be the CCW next segment on $\partial\mathcal{P}$
23         until $u_1 = r$
24     else
25         Symmetric to the case in line 2
26 report the chain of $\mathcal{P}$ between the last and first found cap
```

of generality, we assume that p has coordinates $p = (0, 0)$. We say that a point $q = (q_x, q_y)$ is *parabolically visible* from p if there exists a function $f(x) = ax^2 + bx$ such that $f(q_x) = q_y$ and all the points on the graph of f between p and q are in $\mathcal{P}$, i.e.

$$\{(x, f(x)) \mid 0 \leq x \leq q_x \text{ or } q_x \leq x \leq 0\} \subset \mathcal{P}.$$

If $a = 0$, the parabolically visible point q is also linearly visible. We make the convention that the linearly visible points on the vertical line through p are also parabolically visible. The *parabolical visibility region* is all the points parabolically visible from p. To our knowledge, the notion of parabolical visibility has not been described anywhere in the literature.

The notions of concave and convex supports, caps, and deficiencies have natural analogies in the context of parabolic visibility. A parabolic arc starting at p has two degrees of freedom, namely a and b, just as a circular arc starting at p, namely its radius and start angle. That means that two parabolic arcs emanating from p with three common points must be on the same parabola. Therefore Lemma 11 is readily transferred to parabolic visibility by reading "visibility arc" as "parabolic visibility arc".[3] Likewise, Lemma 12–16 also hold for parabolic visibility. The analogy of finding the support giving the leftmost (resp. rightmost) centre of a circle is to find the support giving the parabola with a vertex having a minimal (resp. maximal) x-coordinate. The *vertex* of the parabola $f(x) = ax^2 + bx$ is the point $(-b/2a, f(-b/2a)) = (-b/2a, -b^2/4a)$. The vertex is the point on the parabola which has the minimal y-coordinate when $a > 0$ and maximal y-coordinate when $a < 0$. Therefore, one can easily alter Algorithm 8 to compute the parabolical visibility region in $O(n^2)$ time using constant workspace.

We say that a point q is *shootable* if there exists a parabola $f(x) = ax^2+bx$ with q on its graph such that $a \leq 0$ and the graph between q and $(0, 0)$ is contained in $\mathcal{P}$. The *shootable region* is all the shootable points, see Figure 10. The portion of the graph of f from p to q is written $\widehat{pq}$ and we say that $\widehat{pq}$ is a *shot*. Analogously to maximal visibility arcs, we define a *maximal shot* to be a shot where the graph of f leaves $\mathcal{P}$ at q. If we assume that there is a uniform gravitational force in $\mathcal{P}$ working in the direction $(0, -1)$ and that there is a vacuum in $\mathcal{P}$, a point q is shootable from p if and only if q is on the trajectory of some punctiform projectile shot from p [29]. Therefore, the shootable region consists of all the points that one can hit when firing a canon placed at p. In this ballistic context, it is more natural to think of the two degrees of freedom of a shot as the angle of the canon and the initial speed of the projectile. The analogy of having $a = 0$ is to fire the projectile with infinite speed.

If rr' is the door to a left pocket of the visibility polygon $\mathcal{LV}_{\mathcal{P}}(p)$ and r has non-negative x-coordinate, then no point in the pocket is shootable from p. Likewise, if rr' is a door of a right pocket of $\mathcal{LV}_{\mathcal{P}}(p)$ and r has a non-positive x-coordinate, no point in the pocket is shootable. Therefore, we have the following lemma.

Lemma 17. *A maximal region of points of $\mathcal{P}$ which are not shootable from p is either*

- *a left pocket of $\mathcal{LV}_{\mathcal{P}}(p)$ with door rr', where r has a non-negative x-coordinate.*
- *a right pocket of $\mathcal{LV}_{\mathcal{P}}(p)$ with door rr', where r has a non-positive x-coordinate.*
- *a convex or concave deficiency due to a maximal shot containing a convex or concave cap.*

[3] Whereas, for instance, the result cannot immediately be used to describe what is elliptically visible from p, since an elliptic arc has four degrees of freedom, namely two radii, its start angle, and the angle between its major semi-axis and the x-axis.

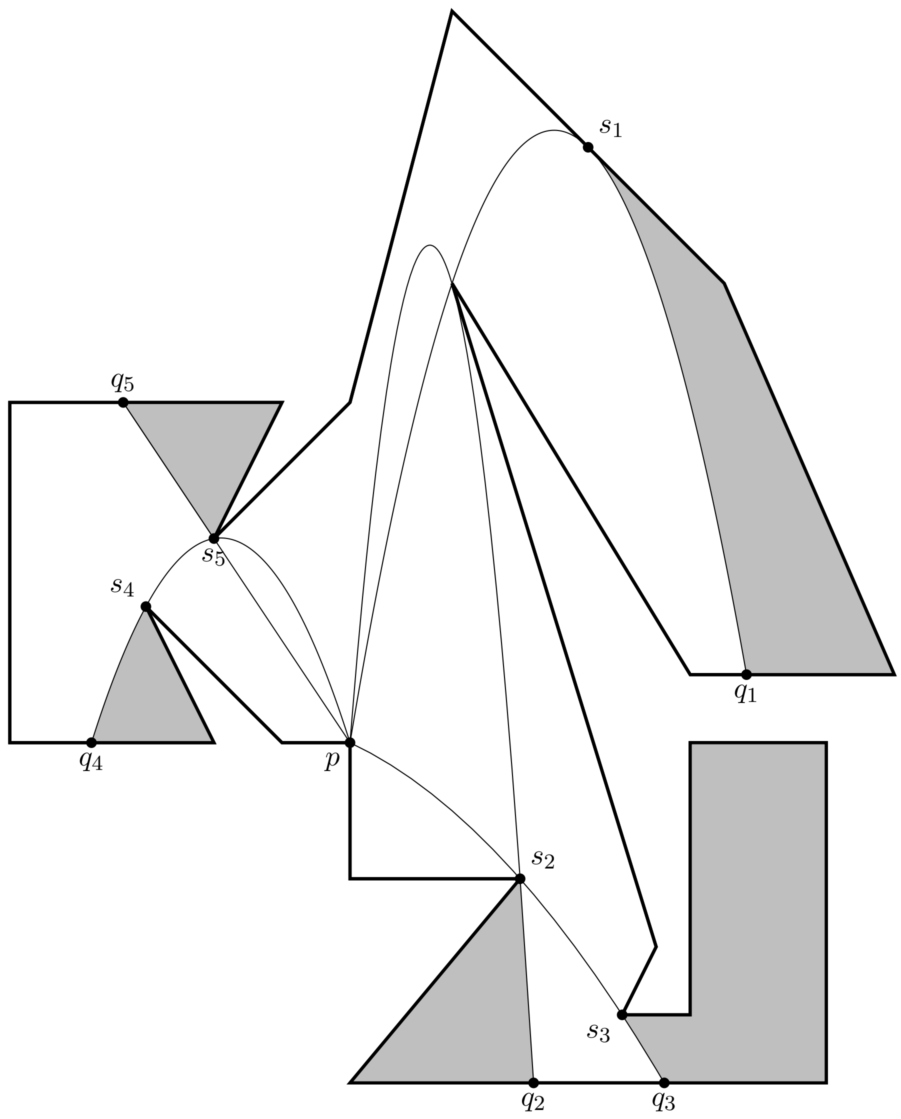


Figure 10. The shootable region from p. The regions that are not shootable are grey. Each of the drawn parabolic arcs contains the boundary between the shootable region and a region of points that are not shootable from p. The arcs $\widehat{s_1 q_1}$ and $\widehat{s_3 q_3}$ are concave caps. The arcs $\widehat{s_2 q_2}$ and $\widehat{s_4 q_4}$ are convex caps. $s_5 q_5$ is the door of a right pocket.

Using this characterisation, one can make a constant-workspace algorithm closely resembling Algorithm 8 computing the shootable region in $O(n^2)$ time. Such an algorithm could for instance be used in the Worms games developed by the British company Team17 to compute the region of points that some worm could shoot from its current position.

6. Concluding Remarks

We have developed the following algorithms:

- An $O(n)$-time algorithm for computing the visible part of one edge from another edge in a polygon.
- An $O(mn)$-time algorithm for computing the weak visibility polygon from a segment in a polygon, where m is the size of the output.
- An $O(n^2)$-time algorithm for computing a minimum link path between two points in a polygon.
- An $O(n^2)$-time algorithm for computing the circular visibility region of a polygon from a point.

The algorithm for computing the visibility between two edges is clearly optimal with respect to both time and space, improving the previously best known algorithm [5].

There are many planar visibility problems for which no constant-workspace algorithm has been described. As an example closely related to our work, we mention the problem of reporting all edges from which every point in the polygon is visible. That problem has been solved in $O(n)$ time using $\Omega(n)$ variables by Shin and Woo [24, 25]. Using the method described in Chapter 3, we can test if $\mathcal{WV}_{\mathcal{P}}(v_i v_{i+1}) = \mathcal{P}$ for each edge $v_i v_{i+1}$ of $\mathcal{P}$, leading to a constant-workspace algorithm running in $O(n^3)$ time. It could be interesting to try to find a faster algorithm.

The *visibility graph* of a polygon $\mathcal{P}$ is a graph with a vertex set equal to the set of vertices of $\mathcal{P}$. Two vertices of the visibility graph are adjacent if the two vertices of $\mathcal{P}$ can see each other. The visibility graph is a very important structure and has applications in many other visibility algorithms [15, Chapter 5]. The graph has $O(n^2)$ edges, and hence an algorithm to find all the edges runs in $\Omega(n^2)$ time. Asano et al. [1] and Welzl [28] described $O(n^2)$-time and -space algorithms, each solving a generalized version of the problem. A naive constant-workspace algorithm using $O(n^3)$ time considers each pair of vertices (v_i, v_j) and traverses all edges of the polygon to see if some edge obstructs the visibility from v_i to v_j. It would be interesting to see if a faster algorithm can be made.

Barba et al. [7] described a randomized algorithm for computing the visibility polygon $\mathcal{LV}_{\mathcal{P}}(p)$ in $O(n \log r)$ expected time using $O(\log r)$ variables. They also gave an $O(n \log^2 r)$-time deterministic variant of the algorithm. Here, $r < n$ is the number of reflex vertices of $\mathcal{P}$. The algorithm is a divide-and-conquer algorithm that repeatedly divides the boundary of $\mathcal{P}$ into smaller and smaller chains until a chain contains at most two reflex vertices. The endpoints of each chain are visible from p. That makes it easy to compute the visible part of the chain. We have thought a lot about how to adapt that method to the computation of the circular visibility region or the weak visibility polygon to get algorithms faster than the ones in this thesis by only using $O(\log n)$ variables, but we have not found a way to do it. The big difference seems to be that the pockets of the visibility polygon are simpler characterized. A beam emanating from p and containing a door of the visibility polygon have one support whereas the visibility arcs containing caps and the beams containing doors of the weak visibility polygon have two supports. That makes it harder to make a divide-and-conquer algorithm. It would be interesting to go deeper into these problems.

The $O(n)$-time algorithm by Joe and Simpson [18] for computing the visibility polygon has the space-time product $O(n^2 \log n)$. The algorithms of Barba et al. [7] improves the bound to $O(n \log n \log r)$ and $O(n \log n \log^2 r)$, respectively, when using $O(\log r)$ variables. It would be very interesting to know more about the space-time product for various sizes of workspace, also for the other problems of this thesis.

References

[1] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1–4):49–63, 1986.

[2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry: Theory and Applications*, to appear.

[3] T. Asano, A. Elmasry, and J. Katajainen. Priority queues and sorting for read-only data. In *Proceedings of the 10th annual conference on Theory and Applications of Models of Computation*, pages 32–41, 2013.

[4] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.

[5] D. Avis, T. Gum, and G. Toussaint. Visibility between two edges of a simple polygon. *The Visual Computer*, 2(6):342–357, 1986.

[6] D. Avis and G.T. Toussaint. An optimal algorithm for determining the visibility of a polygon from an edge. *IEEE Transactions on Computers*, C-30(12):910–914, 1981.

[7] L. Barba, M. Korman, S. Langerman, and R.I. Silveira. Computing the visibility polygon using few variables. In *Proceedings of the 22nd International Symposium on Algorithms and Computation*, volume 7014 of *Lecture Notes in Computer Science*, pages 70–79. Springer, 2011.

[8] P. Beame. A general sequential time-space tradeoff for finding unique elements. *SIAM Journal on Computing*, 20(2):270–277, 1991.

[9] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

[10] S.-Y. Chou, L.-L. Chen, and T.C. Woo. Circular visibility of a simple polygon. Technical report, Industrial and Operations Engineering, University of Michigan, 1992.

[11] S.-Y. Chou and T.C. Woo. A Linear-time algorithm for constructing a circular visibility diagram. *Algorithmica*, 14(3):203–228, 1995.

[12] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition.* MIT Press, 2009.

[13] M. De, A. Maheshwari, and S.C. Nandy. Space-efficient algorithms for visibility problems in simple polygon. E-print, arXiv:1204.2634, 2012.

[14] H. Freeman and P.P. Loutrel. An algorithm for the solution of the two-dimensional "hidden-line" problem. *IEEE Transactions on Electronic Computers*, EC-16(6):784–790, 1967.

[15] S. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, New York, 2007.

[16] L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.

[17] R.A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.

[18] B. Joe and R.B. Simpson. Corrections to lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27(4):458–473, 1987.

[19] D.E. Knuth. *The Art of Computer Programming, Vol. 3, Sorting and Searching, Second Edition.* Addison-Wesley, 1998.

[20] A. Maheshwari. Private communication.

[21] J. O'Rourke. *Art Gallery Theorems and Algorithms.* Oxford University Press, 1987.

[22] J. Pagter and T. Rauhe. Optimal time-space trade-offs for sorting. In *Proceedings of the 39th Annual Symposium on Foundations of Computer Science*, pages 264–268. IEEE, 1998.

[23] J.E. Savage. *Models of Computation: Exploring the Power of Computing.* 2008. Available at `http://cs.brown.edu/~jes/book/`.

[24] S.Y. Shin and T.C. Woo. An optimal algorithm for determining edge visibility in a point-visible polygon. Technical report, Department of Industrial and Operations

Engineering, University of Michigan, 1987.

[25] S.Y. Shin and T.C. Woo. An optimal algorithm for finding all visible edges in a simple polygon. *IEEE Transactions on Robotics and Automation*, 5(2):202–207, 1989.

[26] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

[27] G.T. Toussaint. Shortest path solves edge-to-edge visibility in a polygon. *Pattern Recognition Letters*, 4(3):165–170, 1986.

[28] E. Welzl. Constructing the visibility graph for n line segments in $O(n^2)$ time. *Information Processing Letters*, 20(4):167–171, 1985.

[29] Wikipedia. Article about trajectories. Available at `http://en.wikipedia.org/wiki/Trajectory`, accessed: 11th of March, 2013.

Appendix A: Paper Accepted for CCCG 2013

We submitted the following paper for The 25th Canadian Conference on Computational Geometry, which is to take place in Waterloo, Ontario, Canada in August 8th-10th, 2013. The focus of the paper is on Algorithm 2 and Theorem 1 of this thesis. The paper was accepted and the version provided here is the final version. Notice that a maximum of six pages was allowed.

An Optimal Algorithm Computing Edge-to-Edge Visibility in a Simple Polygon

Mikkel Abrahamsen[*][†]

Abstract

Let $\mathcal{P}$ be a simple polygon with n vertices. We present a new $O(n)$-time algorithm to compute the visible part of one edge from another edge of P. The algorithm does not alter the input and only uses $O(1)$ variables and is therefore a constant-workspace algorithm. The algorithm can be used to make a constant-workspace algorithm for computing the weak visibility polygon from an edge in $O(mn)$ time, where m is the number of vertices of the resulting polygon, and a constant-workspace algorithm for computing a minimum link path between two points inside a simple polygon in $O(n^2)$ time.

1 Introduction

Much research has been done on visibility problems in the plane. See the book by Ghosh [8] for an overview of the most important problems and results.

Let $\mathcal{P}$ be a simple polygon with vertices $v_0v_1 \dots v_{n-1}$ in counterclockwise (CCW) order, and let $v_n = v_0$. A point $q \in \mathcal{P}$ is said to be *visible* from v_jv_{j+1} if there exists a point $p \in v_jv_{j+1}$ such that the segment pq is contained in $\mathcal{P}$. In this paper we show how to compute the visible part of an edge v_iv_{i+1} from the edge v_jv_{j+1}. Without loss of generality, we assume that $j = 0$ for the rest of this paper. The algorithm uses $O(n)$ time is therefore optimal. The input is given in read-only memory and only $O(1)$ variables are needed in the workspace, each consisting of $O(\log n)$ bits. Therefore, the algorithm is a *constant-workspace algorithm.*

The problem of computing visibility between two edges was first addressed by Toussaint [13], who gave a linear-time query algorithm deciding if two edges are visible to each other if a triangulation of $\mathcal{P}$ is provided. Later, Avis et al. [4] described an $O(n)$-time algorithm to compute the visible part of one edge from another which does not require a triangulation or other involved data structures, but uses $\Omega(n)$ variables in the workspace. De et al. [7] claimed to present an $O(n)$-time algorithm using constant workspace. However, their algorithm has a fault, as we shall see.

One of the best-known constant-workspace algorithms for a geometric problem is *Jarvis' march* [10] for the computation of the convex hull of n points in the plane in $O(hn)$ time, where h is the number of points on the hull. Recently, Asano et al. [2], Asano et al. [3], and Barba et al. [5] gave constant-workspace algorithms solving many elementary tasks in planar computational geometry. The research presented in this paper is part of a master's thesis [1], which contains more details and space-efficient solutions to some other planar visibility problems.

1.1 Notation and definitions

Given two points a and b in the plane, the line segment with endpoints a and b is written ab. Both endpoints are included in segment ab. If s is a line segment, the line containing s which is inifinite in both directions is written $\overleftrightarrow{s}$. The *half-line* $\overrightarrow{ab}$ is a line infinite in one direction, starting at a and passing through b. The *right half-plane* RHP(ab) is the closed half plane with boundary $\overleftrightarrow{ab}$ lying to the right of ab. The *left half-plane* LHP(ab) is just RHP(ba).

If $\mathcal{P}$ is a simple polygon, the boundary of $\mathcal{P}$ is written $\partial\mathcal{P}$. Let $\mathcal{P}(p, q)$ for two points $p, q \in \partial\mathcal{P}$ be the set of points on $\partial\mathcal{P}$ we meet when traversing $\partial\mathcal{P}$ CCW from p to q, both included. A *chain* of $\mathcal{P}$ is such a set $\mathcal{P}(p, q)$ for some points $p, q \in \partial\mathcal{P}$. We use the general position assumption that no three vertices of $\mathcal{P}$ are collinear.

Consider the edge v_0v_1 of a simple polygon $\mathcal{P}$. A *beam* emanating from v_0v_1 is a segment pq where $p \in v_0v_1$ and pq is contained in $\mathcal{P}$. Thus, a point q is visible from v_0v_1 if and only if there exists a beam pq emanating from v_0v_1. A *right support* of the beam pq is a reflex vertex v of $\mathcal{P}$ such that $v \in pq$ and the edges meeting at v are both contained in RHP(pq). A *left support* is defined analogously. Since no beam emanates from a point to the left of v_0, we use the convention that v_0 is a left support of any beam v_0q. Likewise, v_1 is a right support of any beam v_1q. A *support* is a right support or a left support.

The edge v_iv_{i+1} is *totally facing* the edge v_jv_{j+1} if both of the points v_j and v_{j+1} are in LHP(v_iv_{i+1}). Notice that v_iv_{i+1} can be totally facing v_jv_{j+1} even though no point on v_jv_{j+1} is visible from v_iv_{i+1}. Edge v_iv_{i+1} is *partially facing* v_jv_{j+1} if excactly one of the points v_j

[*]Department of Computer Science, University of Copenhagen, mikkel.abrahamsen@gmail.com

[†]Autodesk ApS, Havnegade 39, DK-1058 Copenhagen K, Denmark

and v_{j+1} is in LHP(v_iv_{i+1}) and *not facing* v_jv_{j+1} if none of the points are in LHP(v_iv_{i+1}). We say that v_iv_{i+1} is *facing* v_jv_{j+1} if v_iv_{i+1} is partially or totally facing v_jv_{j+1}. It follows from the definitions that v_iv_{i+1} is either totally facing, partially facing or not facing v_jv_{j+1}. That gives 9 different combinations of how v_iv_{i+1} is facing v_jv_{j+1} and how v_jv_{j+1} is facing v_iv_{i+1}. However, only 8 of the cases are possible when v_iv_{i+1} and v_jv_{j+1} are edges of a simple polygon, since they cannot both partially face each other. That would imply that they intersect each other properly. All of the remaining 8 cases are possible. See for instance the paper of Avis et al. [4].

2 Visibility Between Two Edges of a Polygon

2.1 Point-to-point and point-to-edge visibility

If the edge v_iv_{i+1} is not facing edge v_0v_1, the only point on v_iv_{i+1} that can be visible from v_0v_1 is one of the endpoints v_i or v_{i+1}. Likewise, if v_0v_1 is not facing v_iv_{i+1}, the only point on v_0v_1 that can possibly see v_iv_{i+1} is one of the endpoints v_0 or v_1 by means of beams contained in RHP(v_0v_1). In such cases, the problem of computing the visible part of v_iv_{i+1} is reduced to point-to-point and point-to-edge visibility.

Point-to-point visibility is the problem of determining if ab is contained in $\mathcal{P}$ for two given points a and b. That can easily be tested in $O(n)$ time using constant workspace by traversing all edges of $\partial\mathcal{P}$, seeing if $\partial\mathcal{P}$ crosses ab somewhere.

Point-to-edge visibility is the slightly more complicated task of computing the visible part of an edge from a point p. This can also be done using constant workspace and $O(n)$ time by traversing all edges of $\partial\mathcal{P}$ once while keeping track of the vertices shadowing the largest part of the edge in each of the ends [1].

We now turn our attention to the more interesting case of computing the visible part of v_iv_{i+1} from v_0v_1 if the edges are facing each other. We motivate the development of a new algorithm by giving a counterexample to the constant-workspace algorithm of De et al. [7]. The authors are aware of the error [11]. The reader who has not consulted their paper can skip this section.

2.2 Counterexample to the algorithm proposed by De et al. [7]

The textual description and the pseudocode in [7] do not agree. Figure 1 is an example of a polygon where the algorithm computes a wrong result in both cases. After *PASS1*(), the line segment L is still $p_{i+1}p_{j+1}$. After *PASS2*(), L is θp_{j+1}. The text says that *PASS3*() is to check if a vertex on $\mathcal{P}(p_{j+1}, p_i)$ is to the right of L. All the vertices are to the left, so the algorithm returns that the rightmost visible point on p_jp_{j+1} from p_ip_{i+1}

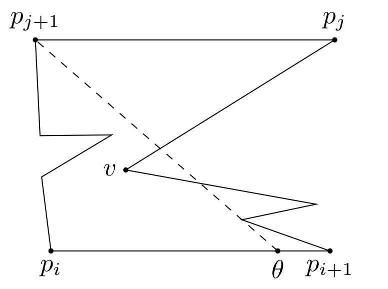


Figure 1: The algorithm from [7] reports the wrong visible part of p_jp_{j+1} from p_ip_{i+1} in this polygon.

is p_{j+1}, which is wrong. The pseudocode gives another definition of *PASS3*(), according to which we also check if a vertex on $\mathcal{P}(p_{i+1}, p_j)$ is to the left of L. Vertex v is, so the algorithm reports that nothing of p_jp_{j+1} is visible. That is clearly also wrong.

2.3 Computing visibility between edges facing each other

Assume for the rest of this section that the edges v_0v_1 and v_iv_{i+1} are facing each other. We want to compute the part of v_iv_{i+1} containing v_{i+1} that is not visible from v_0v_1. The main idea is to consider the edges in the right side chain $\mathcal{P}(v_1, v_i)$ and the left side chain $\mathcal{P}(v_{i+1}, v_0)$ alternately, changing side after each edge. When an edge in one side is found that causes more of v_iv_{i+1} to be invisible from v_0v_1, we retract the search in the other chain to the last interfering edge in that chain. This will be made more precise in the following.

Let $\square = \square v_0v_1v_iv_{i+1}$ be the quadrilateral with vertices $v_0v_1v_iv_{i+1}$ in that order. The possible beams from v_0v_1 to v_iv_{i+1} are all contained in $\square$, so when computing the visible part of v_iv_{i+1}, we are only concerned about the edges of $\mathcal{P}$ that are (partially) in $\square$. A beam pq is a *proper beam* if $pq \subset$ LHP(v_0v_1) and $pq \subset$ LHP(v_iv_{i+1}). An *improper beam* is a beam that is not proper. Each beam pq where p is an interior point on v_0v_1 and q is an interior point on v_iv_{i+1} is necessarily proper. Therefore, if pq is improper, $p = v_0$, $p = v_1$, $q = v_i$, or $q = v_{i+1}$. The visibility due to improper beams can be computed using point-to-edge visibility, so in this section, we focus on the visibility due to proper beams only. We leave out the proof of the following lemma due to limited space [1].

Lemma 1 *Let $v_R \in \mathcal{P}(v_1, v_i) \cap \square$ and $v_L \in \mathcal{P}(v_{i+1}, v_0) \cap \square$. Every proper beam pq from v_0v_1 to v_iv_{i+1} satisfies $p \in LHP(v_Rv_L)$ and $q \in RHP(v_Rv_L)$. In particular, if $v_0v_1 \cap LHP(v_Rv_L) = \emptyset$ or $v_iv_{i+1} \cap RHP(v_Rv_L) = \emptyset$, then no proper beam from v_0v_1 to v_iv_{i+1} exists.*

Assume that there are some proper beams from v_0v_1

to v_iv_{i+1}. We say that the beam pq is the *rightmost beam* from v_0v_1 to v_iv_{i+1} if p is as close to v_1 as possible and q is as close to v_{i+1} as possible among all proper beams. Similarly, pq is the *leftmost beam* from v_0v_1 to v_iv_{i+1} if p is as close to v_0 as possible and q is as close to v_i as possible. If v_0v_1 and v_iv_{i+1} are totally facing each other, all beams from v_0v_1 to v_iv_{i+1} are proper, so the visible part of v_iv_{i+1} is the points between the endpoints of the leftmost and rightmost beams. If one of the edges is only partially facing the other, the visible part of v_iv_{i+1} can be computed using the leftmost and rightmost beams in combination with point-to-edge visibility.

If pq is a beam from v_0v_1 to v_iv_{i+1}, a *generalized left support* of pq is v_{i+1} if $q = v_{i+1}$ or a left support of pq otherwise. The following lemma characterizes rightmost beams by means of their supports. The proof is given in [1].

Lemma 2 *Let pq be a proper beam from v_0v_1 to v_iv_{i+1}. The beam pq is a rightmost beam if and only if pq has a right support v_R and a generalized left support v_L and $v_L \in v_Rq$.*

If the edges v_0v_1 and v_iv_{i+1} are totally facing each other and no edge obstructs the visibility between the edges, then the rightmost beam is $pq = v_1v_{i+1}$ and it has supports $v_R = v_1$ and $v_L = v_{i+1}$.

Algorithm 1 returns the indices (R, L) of the supports of the rightmost beam if it exists. The algorithm iteratively computes the correct value of R and L, taking the edges into consideration one by one. Initially, R is set to 1 and L is set to $i + 1$, as if no edges obstructs the visibility between the edges. The points p and q on v_0v_1 and v_iv_{i+1}, respectively, are always defined such that the segment pq contains v_R and v_L. The algorithm alternately traverses $\mathcal{P}(v_1, v_i)$ and $\mathcal{P}(v_{i+1}, v_n)$ one edge at a time using the index variables r and l. The variable *side* is 1 when an edge in $\mathcal{P}(v_1, v_i)$ is traversed and -1 when an edge in $\mathcal{P}(v_{i+1}, v_n)$ is traversed. Each time an edge $v_{r-1}v_r$ or $v_{l-1}v_l$ is found that crosses pq, the value of R or L is updated to r or l, respectively. If the value of R is updated, we reset l to L, since it is possible that there are some edges on $\mathcal{P}(v_L, v_n)$ that did not intersect the old segment pq, but intersect the updated one. Likewise, when L is updated, we reset r to R. Although segment pq is changed when R or L is updated, $\mathcal{P}(v_1, v_R)$ or $\mathcal{P}(v_{i+1}, v_L)$ does not cross pq after the update. That is because pq is rotated clockwise (CW) away from the chains.

All our figures illustrate the case where v_0v_1 and v_iv_{i+1} are totally facing each other, but that assumption is not used in any of the proofs. If v_iv_{i+1} is partially facing v_0v_1 such that $v_0 \in \mathrm{LHP}(v_iv_{i+1})$, then v_i might be the right support of the rightmost beam from v_0v_1 to v_iv_{i+1}. Likewise, if v_0v_1 is partially facing v_iv_{i+1} such that $v_i \in \mathrm{LHP}(v_0v_1)$, v_0 can be the left support of the rightmost beam.

Algorithm 1: `FindRightmostBeam`(i)

Input: A polygon $\mathcal{P}$ defined by its vertices $v_0, v_1, \ldots, v_{n-1}$ in CCW order and an index i such that v_0v_1 and v_iv_{i+1} are facing each other.

Output: If no proper beam from v_0v_1 to v_iv_{i+1} exists, `NULL` is returned. Otherwise, a pair of indices (R, L) is returned such that the rightmost beam from v_0v_1 to v_iv_{i+1} has right support v_R and generalized left support v_L.

```
1  $R \leftarrow 1$,   $L \leftarrow i + 1$
2  $r \leftarrow R$,   $l \leftarrow L$
3  $p \leftarrow v_1$,   $q \leftarrow v_{i+1}$
4  $side \leftarrow 1$   (* 1 is right side, $-1$ is left side *)
5  while $r < i$ or $l < n$
6      if $side = 1$
7          if $r < i$
8              $r \leftarrow r + 1$
9              if $v_{r-1}v_r$ enters $\mathrm{LHP}(pq) \cap \square$
10                 if $v_{r-1}v_r$ intersects $v_Lq$
11                     return NULL
12                 $R \leftarrow r$,   $l \leftarrow L$
13     else   (* $side = -1$ *)
14         if $l < n$
15             $l \leftarrow l + 1$
16             if $v_{l-1}v_l$ enters $\mathrm{RHP}(pq) \cap \square$
17                 if $v_{l-1}v_l$ intersects $v_Rp$
18                     return NULL
19                 $L \leftarrow l$,   $r \leftarrow R$
20     Let $p$ be the intersection point between $\overrightarrow{v_Lv_R}$ and $v_0v_1$
21     Let $q$ be the intersection point between $\overrightarrow{v_Rv_L}$ and $v_iv_{i+1}$
22     if $p$ or $q$ does not exist
23         return NULL
24     $side \leftarrow -side$
25 if $pq \subset \mathrm{LHP}(v_0v_1) \cap \mathrm{LHP}(v_iv_{i+1})$
26     return $(R, L)$
27 else
28     return NULL
```

Lemma 3 *Assume that Algorithm 1 terminates after k iterations of the loop at line 5. Let R_j, L_j, p_j, and q_j be the values of R, L, p, and q, respectively, in the beginning of iteration j, $j = 1, 2, \ldots, k + 1$, where the values when the algorithm terminates have index $k + 1$. Then $R_j = R_{j+1}$ or $L_j = L_{j+1}$ for*

$j = 1, \ldots, k$. $p_1, p_2, \ldots, p_{k+1}$ is a sequence of points moving monotonically along v_0v_1 from v_1 towards v_0. Likewise, $q_1, q_2, \ldots, q_{k+1}$ is a sequence of points moving monotonically along v_iv_{i+1} from v_{i+1} towards v_i. Let a_j be the CW angle from $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ to $\overleftrightarrow{v_{R_j}v_{L_j}}$. Then $\sum_{j=2}^{k+1} a_j < 180°$. In particular, $a_j < 180°$ for each $2 = 1, \ldots, k+1$.

Proof. See Figure 2. It is clear that at most one of R and L changes in iteration j, since the lines 12 and 19 cannot both be executed. Therefore, $R_j = R_{j+1}$ or $L_j = L_{j+1}$. If R is redefined in iteration j, then $\overleftrightarrow{v_Rv_L}$ is rotating around v_L and the new value of R, namely R_{j+1}, satisfies $v_{R_{j+1}} \in \text{LHP}(v_{R_j}v_{L_j})$. Therefore, p_{j+1} is on the segment v_0p_j and q_{j+1} is on the segment q_jv_i. The same is true if L is updated. Hence, $p_1, \ldots, p_{k+1}$ is monotinically moving along v_0v_1 from v_1 towards v_0 and $q_1, \ldots, q_{k+1}$ is monotonically moving along v_iv_{i+1} from v_{i+1} towards v_i. Because of the monotonicity, the angles are additive, so that the CW angle from $\overleftrightarrow{v_{R_1}v_{L_1}}$ to $\overleftrightarrow{v_{R_{k+1}}v_{L_{k+1}}}$ is $\sum_{j=2}^{k+1} a_j$. If v_0v_1 is totally facing v_iv_{i+1}, every q_j is contained in $\text{LHP}(v_0v_1)$. Otherwise v_iv_{i+1} is totally facing v_0v_1 so that every p_j is contained in $\text{LHP}(v_iv_{i+1})$. In either case, $\sum_{j=2}^{k+1} a_j$ is bounded by $180°$. That bound cannot be reached, since it would require that v_0v_1 or v_iv_{i+1} was infinitely long in both directions. $\square$

Lemma 4 *Algorithm 1 correctly computes the rightmost beam from v_0v_1 to v_iv_{i+1} as specified. The algorithm is a constant-workspace algorithm.*

Proof. First, consider the cases where the algorithm returns `NULL`. In line 11, we have found an intersection point x between $\mathcal{P}(v_R, v_i)$ and v_Lq. That means that $\mathcal{P}(v_1, v_i)$ intersects pq properly at x, since no three vertices are collinear. Lemma 1 establishes that the only possible proper beams from v_0v_1 to v_iv_{i+1} are of the form $p'q'$, where $p' \in v_0p$ and $q' \in qv_i$. At the same time, if we use Lemma 1 with v_0v_1 and v_iv_{i+1} interchanged by each other and using x as 'v_L' and v_L as 'v_R', we get that $p'q'$ satisfies $p' \in pv_1$ and $q' \in v_{i+1}q$. Therefore, $p' = p$ and $q' = q$, but pq is not a beam. Hence, there are no proper beams from v_0v_1 to v_iv_{i+1}. The case in line 18 is analogous.

Due to Lemma 3, we know that p is moving monotonically from v_1 towards v_0 and q is moving monotonically from v_{i+1} towards v_i. The case of line 23 happens if p has moved outside v_0v_1, so that $v_0v_1 \cap \text{LHP}(v_Rv_L) = \emptyset$, or q has moved outside v_iv_{i+1}, so that $v_iv_{i+1} \cap \text{RHP}(v_Rv_L) = \emptyset$. In each of these cases, it follows from Lemma 1 that there are no proper beams from v_0v_1 to v_iv_{i+1}.

The test at line 25 is to ensure that pq is a proper beam, which is not always the case if v_0v_1 is only partially facing v_iv_{i+1}.

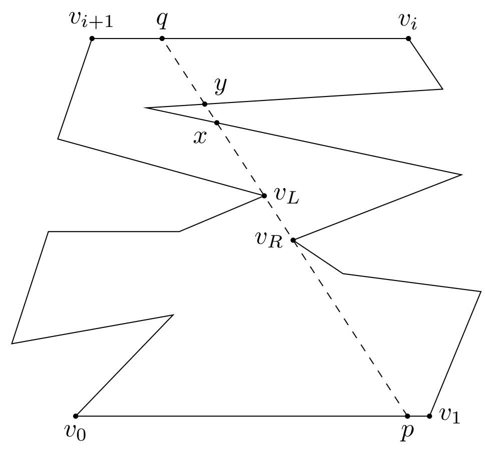


Figure 3: Case 2 in the proof of Theorem 4.

Now, assume that the algorithm returns (R, L), but that pq is not a beam since some edge obstructs the visibility from p to q. Assume that $\mathcal{P}(v_1, v_i)$ intersects pq properly, and let x be the intersection point closest to p. $\mathcal{P}(v_1, v_i)$ enters $\text{LHP}(pq) \cap \square$ at x. Let y be the first point where $\mathcal{P}(x, v_i)$ crosses pq from left to right. Then $y \in xq$. We have two cases: $x \in \mathcal{P}(v_1, v_R)$ (case 1) and $x \in \mathcal{P}(v_R, v_i)$ (case 2). Assume that we are in case 2, see Figure 3. Assume that the final value of R is defined in a later iteration of the loop at line 5 than the final value of L. After R is defined in line 12, every edge $v_{r-1}v_r$ in $\mathcal{P}(v_R, v_i)$ is traversed and it is checked in line 10 if some edge intersects pq. In particular the edges in $\mathcal{P}(x, y)$ are traversed, in which case the algorithm either returns `NULL` or updates R, which is a contradiction. If R is defined in an earlier iteration than L, then r is reset to R in line 19 when L is defined, and it is checked if some edge in $\mathcal{P}(v_R, v_i)$ intersects pq, so that cannot happen either.

Assume that we are in case 1, i.e. $x \in \mathcal{P}(v_1, v_R)$. Consider the first iteration, say iteration j, at the beginning of which $\mathcal{P}(v_1, v_R)$ intersects pq properly, and let x' be the intersection point closest to p. Let y' be the first point where $\mathcal{P}(x', v_i)$ crosses pq from left to right. Then $y' \in x'q$ (x' and y' might not be the same as x and y, since R and L can change before the algorithm terminates). We must have $v_R \in \mathcal{P}(y', v_i)$. There are three possible cases to consider: $v_R \in px'$ (case 1.1), $v_R \in x'y'$ (case 1.2), and $v_R \in y'q$ (case 1.3).

Assume case 1.3. Let R_k, L_k, p_k, and q_k be defined as in Lemma 3 for each iteration k. Either R or L is redefined in iteration $j-1$ due to the minimality of j. Therefore, $R_{j-1} \neq R_j$ or $L_{j-1} \neq L_j$ (case 1.3.1 and 1.3.2, respectively). First, assume $R_{j-1} \neq R_j$ but $L_{j-1} = L_j$. Again, there are three cases to consider: $v_{R_{j-1}} \in \mathcal{P}(v_1, x')$ (case 1.3.1.1), $v_{R_{j-1}} \in \mathcal{P}(x', y')$ (case 1.3.1.2), and $v_{R_{j-1}} \in \mathcal{P}(y', v_{R_j})$ (case 1.3.1.3). As-

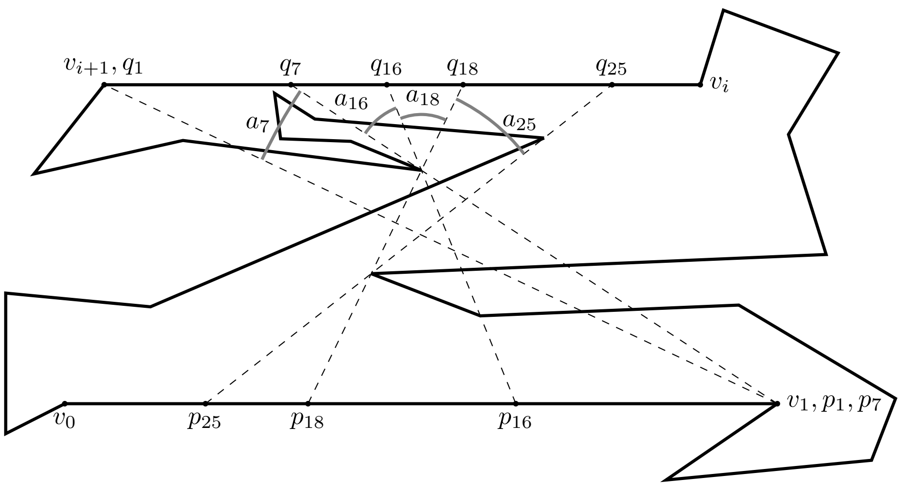


Figure 2: Illustration for Lemma 3. The points p_j, q_j are shown with the number j of the first iteration where they occur. The segments p_jq_j are drawn dashed. The angles $a_j > 0$ are indicated with grey arcs.

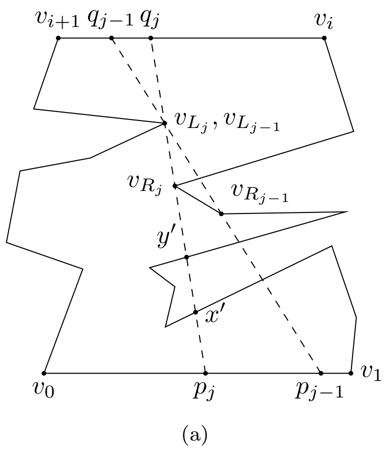
sume case 1.3.1.3, see Figure 4(a). According to Lemma 3, the CW angle between $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ and $\overleftrightarrow{v_{R_j}v_{L_j}}$ is less than 180°. Therefore, a subset of $\mathcal{P}(x', y')$ would also be contained in LHP$(v_{R_{j-1}}v_{L_{j-1}}) \cap \square$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1}q_{j-1}$, a contradiction because of the choice of j. $v_{R_{j-1}}$ cannot be in $\mathcal{P}(x', y')$ (case 1.3.1.2), because then v_{R_j} would be in RHP$(v_{R_{j-1}}v_{L_{j-1}})$, so R would not have been redefined to R_j in iteration $j-1$. Finally, if $v_{R_{j-1}}$ was a vertex in $\mathcal{P}(v_1, x')$ (case 1.3.1.1), $\mathcal{P}(x', y')$ would be contained in LHP$(v_{R_{j-1}}v_{L_{j-1}}) \cap \square$, and therefore v_R would be redefined to a vertex in $\mathcal{P}(x'y')$ when the edges of that chain was traversed. Hence, v_R would not be redefined to v_{R_j} in iteration $j-1$, which is a contradiction.

Now, assume $L_{j-1} \neq L_j$ (case 1.3.2), see Figure 4(b). The CW angle between $\overleftrightarrow{v_{R_{j-1}}v_{L_{j-1}}}$ and $\overleftrightarrow{v_{R_j}v_{L_j}}$ is less than 180°. Therefore, a part of $\mathcal{P}(x', y')$ is also in LHP$(v_{R_{j-1}}v_{L_{j-1}})$. That implies that $\mathcal{P}(v_1, v_{R_{j-1}})$ intersects $p_{j-1}q_{j-1}$, which contradicts the choice of j.

The case where $v_R \in x'y'$ (case 1.2) can be eliminated in a similar way. Consider case 1.1, i.e. $v_R \in px'$. The chain $\mathcal{P}(p, x')$ and the segment $x'p$ forms a simple, closed curve, because x' is the intersection point between $\mathcal{P}(v_1, v_i)$ and pq closest to p. The curve can, for instance, be seen in Figure 4(a). Consider the region of $\mathcal{P}$ enclosed by the curve. In order to get to v_R, $\mathcal{P}(y', v_i)$ has to cross $x'p$ to get into the region. That contradicts that x' was the intersection point closest to p.

If we assume that $\mathcal{P}(v_{i+1}, v_n)$ intersects pq, we get a contradiction in an analogous way.

The conclusion is that if (R, L) is returned, v_R and v_L defines a proper beam pq with right support v_R and generalized left support v_L in that order. Therefore, pq must be the rightmost beam from v_0v_1 to v_iv_{i+1} according to Lemma 2.



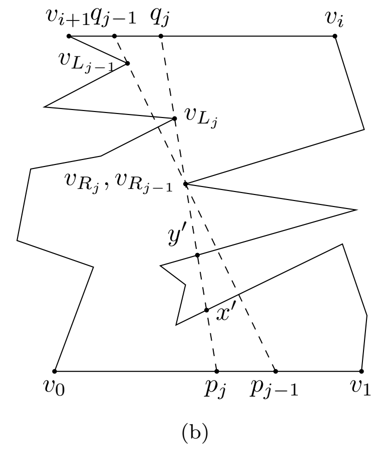


Figure 4: Cases in the proof of Theorem 4. (a) Case 1.3.1.3. (b) Case 1.3.2.

Observe that the vertices of $\mathcal{P}$ are not altered. Hence the input is read only. In addition to that, we only use the variables R, L, r, l, p, q, and *side*. The computations of intersections and containment at lines 9, 10, 16, 17, 20, 21, and 25 are easily implemented using constant workspace. □

Even though we reset l to L in line 12 or r to R in line 19, the running time is linear since the other variable is not reset, so half of the traversed edges are never traversed again, as the following lemma explains.

Lemma 5 *There are at most $2n-6$ iterations of the loop at line 5 of Algorithm 1.*

Proof. Let $N(n)$ be the maximal number of edge visits for a polygon with n vertices. Consider the first time line 12 or 19 is executed. Assume it is line 12. There have been made $2(r-1)-1 < 2(r-1)$ iterations, because $\mathcal{P}(v_1, v_i)$ is traversed every second time, beginning with the first. The $r-1$ edges in $\mathcal{P}(v_1, v_r)$ are never traversed again. Therefore, N satisifies the recurrence $N(n) \leq 2k + N(n-k)$, where $k = r-1$. A similar bound holds for some $k \geq 1$ if line 19 is executed first. We know that $N(4) = 2$, so induction yields that $N(n) \leq 2n-6$ is an upper bound. □

It is clear that an algorithm to compute a leftmost beam from v_0v_1 to v_iv_{i+1} can be constructed symmetrically. That gives us the following theorem:

Theorem 6 *The visible part of an edge v_iv_{i+1} from v_0v_1 in a simple polygon can be computed in $O(n)$ time using constant workspace.*

3 Weak Visibility Polygons and Minimum Link Paths

The weak visibility polygon of the polygon $\mathcal{P}$ from the edge v_0v_1 consists of all the points in $\mathcal{P}$ visible from v_0v_1. Guibas et al. [9] presented an $O(n)$-time algorithm to compute the weak visibility polygon if a triangulation of $\mathcal{P}$ is provided, where n is the number of vertices of $\mathcal{P}$. Later, Chazelle [6] described an $O(n)$-time deterministic triangulation algorithm, implying that the weak visibility polygon can be computed in $O(n)$ time using $O(n)$ space. Using Algorithm 1, one can make a $O(mn)$-time algorithm using constant workspace, where m is the number of edges of the weak visibility polygon [1]. It is well-known that $m = O(n)$.

A *minimum link path* between two points s and t in a simple polygon is a polygonal path from s to t which is contained in $\mathcal{P}$ and which consists of as few segments as possible. Suri [12] showed how to compute a minimum link path using $O(n)$ time if a triangulation of $\mathcal{P}$ is provided. Using the algorithm to compute the weak visibility polygon, it is possible to devise an $O(n^2)$-time algorithm to compute a minimum link path using constant workspace. The algorithm does not use a triangulation of $\mathcal{P}$. The details are given in [1].

Acknowledgements

We would like to thank Jyrki Katajainen, Ashwini Joshi, Kristian Mortensen, and the anonymous reviewers for suggesting many improvements to the present paper.

References

[1] M. Abrahamsen. Constant-workspace algorithms for visibility problems in the plane. Master's thesis. University of Copenhagen, Department of Computer Science, 2013. Available at `http://www.diku.dk/forskning/performance-engineering/Mikkel/thesis.pdf`.

[2] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Computational Geometry: Theory and Applications*, to appear.

[3] T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithms for geometric problems. *Journal of Computational Geometry*, 2(1):46–68, 2011.

[4] D. Avis, T. Gum, and G. Toussaint. Visibility between two edges of a simple polygon. *The Visual Computer*, 2(6):342–357, 1986.

[5] L. Barba, M. Korman, S. Langerman, and R. Silveira. Computing the visibility polygon using few variables. In *Proceedings of the 22nd International Symposium on Algorithms and Computation*, volume 7014 of *Lecture Notes in Computer Science*, pages 70–79. Springer, 2011.

[6] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(1):485–524, 1991.

[7] M. De, A. Maheshwari, and S. Nandy. Space-efficient algorithms for visibility problems in simple polygon. E-print arXiv:1204.2634, 2012.

[8] S. Ghosh. *Visibility Algorithms in the Plane.* Cambridge University Press, 2007.

[9] 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(1):209–233, 1987.

[10] R. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2(1):18–21, 1973.

[11] A. Maheshwari. Private communication.

[12] S. Suri. A linear time algorithm for minimum link paths inside a simple polygon. *Computer Vision, Graphics, and Image Processing*, 35(1):99–110, 1986.

[13] G. Toussaint. Shortest path solves edge-to-edge visibility in a polygon. *Pattern Recognition Letters*, 4(3):165–170, 1986.