# Computing a visibility polygon using few variables*

Luis Barba † Matias Korman ‡ Stefan Langerman†§ Rodrigo I. Silveira‡

**Abstract**

We present several algorithms for computing the visibility polygon of a simple polygon $\mathcal{P}$ of $n$ vertices (out of which $r$ are reflex) from a viewpoint inside $\mathcal{P}$, when $\mathcal{P}$ resides in read-only memory and only few working variables can be used. The first algorithm uses a constant number of variables, and outputs the vertices of the visibility polygon in $O(n\bar{r})$ time, where $\bar{r}$ denotes the number of reflex vertices of $\mathcal{P}$ that are part of the output. Whenever we are allowed to use $O(s)$ variables, the running time decreases to $O(\frac{nr}{2^s} + n\log^2 r)$ (or $O(\frac{nr}{2^s} + n\log r)$ randomized expected time), where $s \in O(\log r)$. This is the first algorithm in which an exponential space-time trade-off for a geometric problem is obtained.

## 1 Introduction

The *visibility polygon* of a simple polygon $\mathcal{P}$ from a viewpoint $q$ is the set of all points of $\mathcal{P}$ that can be seen from $q$, where two points $p$ and $q$ can see each other whenever the segment $pq$ is contained in $\mathcal{P}$. The visibility polygon is a fundamental concept in computational geometry and one of the first problems studied in planar visibility. The first correct and optimal algorithm for computing the visibility polygon from a point was found by Joe and Simpson [18]. It computes the visibility polygon from a point in linear time and space. We refer the reader to the survey of O'Rourke [21] and the book of Gosh [15] for an extensive discussion of such problems.

In this paper we look for an algorithm that computes the visibility polygon of a given point and uses few variables. This kind of algorithm not only provides an interesting trade-off between running time and memory needed, but is also useful in portable devices where important hardware constraints are present (such as the ones found in digital cameras or mobile phones). In addition, this model has direct applications in concurrent environments where several devices with limited memory resources perform some computation on a large centralized input. Since several devices may access the input at the same time, allowing writing to the input memory can result in compromising its integrity.

A significant amount of research has focused on the design of algorithms that use few variables, some of them even dating from the 80s [19]. Although many models exist, most of the research considers that the input is in some kind of read-only data structure. In addition to the input values, we are allowed to use few additional variables (typically a variable holds a logarithmic number of bits).

One of the most studied problems in this setting is that of selection. For any constant $\epsilon \in (0,1)$, Munro and Raman [20] gave an algorithm that runs in $O(n^{1+\epsilon})$ time and uses $O(1/\epsilon)$ variables. Frederickson [14] extended this result to the case in which $s$ working variables are available (and $s \in \Omega(\log n) \cap O(2^{\log n/\log^* n})$). Raman and Ramnath [22] gave several exact and approximation

*A preliminary version of this paper appeared in the proceedings of the 22nd International Symposium on Algorithms and Computation (ISAAC 2011) [8].

†Université Libre de Bruxelles (ULB), Brussels, Belgium. {lbarbafl,stefan.langerman}@ulb.ac.be

‡Universitat Politècnica de Catalunya (UPC), Barcelona, Spain. {matias.korman, rodrigo.silveira}@upc.edu. With the support of the Secretary for Universities and Research of the Ministry of Economy and Knowledge of the Government of Catalonia, the European Union, the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235, and ESF EUROCORES programme EuroGIGA - ComPoSe IP04 - MICINN Project EUI-EURC-2011-4306.

§Directeur de Recherches du FRS-FNRS.

algorithms for the case in which fewer variables are available. Among other results, they provide a 2/3-approximation of the median that runs in $O(sn^{1+1/s})$ time using $O(s)$ variables (for $s \in o(\log n)$), or $O(n \log n)$ time, using $O(\log n)$ variables. More recently Chan [12] provided several lower bounds for performing selection with few variables.

In recent years there has been a growing interest in finding algorithms for geometric problems that use a constant number of variables. An early example is the well-known gift-wrapping algorithm (also known as Jarvis march [17]), which can be used to report the points on the convex hull of a set of $n$ points in $O(n\bar{h})$ time using a constant number of variables, where $\bar{h}$ is the number of vertices on the convex hull. Recently, Asano and Rote [6] and afterwards Asano *et al.* [3, 5] gave efficient methods for computing well-known geometric structures, such as the Delaunay triangulation, the Voronoi diagram, a polygon triangulation, and a minimum spanning tree (MST) using a constant number of variables. These algorithms run in $O(n^2)$ time (except computing the MST, which needs $O(n^3)$ time). Observe that, since these structures have linear size, they are not stored but reported. Prior to this work, there was no algorithm for computing the visibility polygon in memory-constrained models. Indeed, this problem was explicitly posed as an open problem by Asano *et al.* [4] for the case in which only a constant number of variables are allowed.

### Results

In this paper we present a novel approach for computing the visibility polygon of a given point inside a simple polygon. It is easy to see that reflex vertices have a much larger influence on the visibility polygon than convex vertices. Therefore, whenever possible we express the running time of our algorithms not only in terms of $n$, the complexity of $\mathcal{P}$, but also in terms of $r$ and $\bar{r}$ (the number of reflex vertices of $\mathcal{P}$ that are present in the input and in the output, respectively). This approach continues a line of research relating the combinatorial and computational properties of polygons to the number of their reflex vertices. We refer the reader to [1, 9, 10] and references found therein for a deep review of existing similar results.

In Section 2 we begin the paper with some preliminaries, followed by some observations and basic algorithms in Section 3. In Section 4 we give an output-sensitive algorithm that reports the vertices of the visibility polygon in $O(n\bar{r})$ time using $O(1)$ variables. Using this algorithm as a stepping stone, in Section 5 we present a divide-and-conquer algorithm. This algorithm runs in $O(\frac{nr}{2^s} + n \log^2 r)$ time (or $O(\frac{nr}{2^s} + n \log r)$ randomized expected time) using $O(s)$ variables (for any $s \in O(\log r)$), giving an exponential trade-off between running time and space.

**Remark:** prior to this research there was no known method for computing visibility polygons using few variables. Following the conference version of this paper [8], De *et al.* [13] provided a linear-time algorithm that uses $O(\sqrt{n})$-variables. Parallel to this research, Barba *et al.* [7] gave a general method for transforming stack-based algorithms into memory constrained workspaces. Since Joe and Simpson's algorithm for computing the visibility polygon [18] is stack-based, their approach can be used for this problem as well.

## 2 Preliminaries

### Model definition and considerations on input/output precision

We use a slight variation of the constant workspace model, introduced by Asano and Rote [6]. In this model the input of the problem resides in a read-only data structure and we are allowed to perform random access to any of the values of the input in constant time. An algorithm can use a constant number of variables and we assume that each variable or pointer contains a data word of $O(\log n)$ bits. Implicit storage consumption required by recursive calls is also considered part of the workspace. This model is also referred as *log space* [2] in the complexity literature.

Many other similar models have been studied. We note that in some of them (like the *streaming* [16] or the *multi-pass* model [11]) the values of the input can only be read once or a fixed number of times. As in the constant workspace model of Asano and Rote [6], our model allows scanning the input as many times as necessary. However, our model differs from theirs in two

aspects: we are allowed to use a workspace of $O(s)$ variables (instead of $O(1)$), and we do not require random access to the vertices of the input.

The input to our problem is a simple polygon $\mathcal{P}$ in a read-only data structure and a point $q$ in the plane, from where the visibility polygon needs to be computed. We do not make any assumptions on whether the input coordinates are rational or real numbers (in some implicit form). The only operations that we perform on the input are determining whether a given point is above, below or on a given line and determining the intersection point between two lines. In both cases, the line is defined as passing through two points of the input, hence both operations can be expressed as finding the root of linear equations whose coefficients are values of the input. We assume that these two operations can be done in constant time. Moreover, if the coordinates of the input are algebraic values, we can express the coordinates of the output as "the intersection point of the line passing through points $p_i$ and $p_j$ and the line passing through $p_k$ and $p_l$" (where $p_i, p_j, p_k$ and $p_l$ are vertices of the input).

**Definitions and basic properties**

We say that a point $p$ is *visible* from $q$ (with respect to $\mathcal{P}$) if and only if the segment $pq$ is contained in $\mathcal{P}$ (note that we regard $\mathcal{P}$ as a closed subset of the plane). The set of points visible from $q$ is called the *visibility polygon* of $\mathcal{P}$, and is denoted $\text{Vis}_{\mathcal{P}}$. Note that if $q$ is outside of $\mathcal{P}$ then $\text{Vis}_{\mathcal{P}}$ is by definition empty. Thus, when considering visibility with respect to polygons, we always assume that $q$ is inside the polygon. From now on, we assume that $q$ is fixed, hence we omit the "with respect to $q$" term.

We assume we are given $\mathcal{P}$ as a list of its vertices in counterclockwise order along its boundary (denoted by $\partial\mathcal{P}$). Let $p_0$ be a point on $\partial\mathcal{P}$ closest to $q$ on the horizontal line passing through $q$. It is easy to see that $p_0$ is visible and can be computed in linear time. In the following, we will treat $p_0$ as a vertex of $\mathcal{P}$ (even though it does not need to be one). By implicitly reordering the vertices of the input, we can assume that we are given the vertices of $\mathcal{P}$ in counterclockwise direction starting from $p_0$ (i.e., $\mathcal{P} = (p_0, \ldots, p_n)$).

For simplicity of exposition, we assume that the vertices of $\mathcal{P}$ are in a weak general position; that is, we assume that there do not exist two vertices $p, p' \in \mathcal{P}$ such that $p, p'$, and $q$ are aligned (but we note that the algorithms can be extended easily for the general case).

Along this paper, we will often work with polygonal chains (instead of polygons). However, we will restrict our scope to polygonal chains contained in $\partial\mathcal{P}$. For any two points $a, b$ on $\partial\mathcal{P}$, there is a unique path from $a$ to $b$ that travels counterclockwise along $\partial\mathcal{P}$; let $\mathcal{C} = \text{Chain}(a, b)$ be the set of points traversed in this path, including both $a$ and $b$ (this set is called the *chain* between $a$ and $b$). We say that $a$ and $b$ are the *endpoints* of $\mathcal{C}$, and we refer to the rest of the points on $\mathcal{C}$ as its *interior points*.

We now extend the concept of visibility to chains. Due to technical reasons, we define this concept for chains contained in $\partial\mathcal{P}$ whose endpoints are visible from $q$. We say that a chain $\mathcal{C}$ is *independent* if and only if its endpoints are visible points of $\mathcal{P}$. Given a chain $\mathcal{C} = \text{Chain}(a, b)$ with endpoints $a$ and $b$, let $\mathcal{R}_{\mathcal{C}}$ denote the polygon enclosed by the union of $\mathcal{C}$ and the segments $qa$ and $qb$ (equivalently, we use the notation $\mathcal{R}(a, b)$).

Given a point $q \in \mathbb{R}^2$ and an independent chain $\mathcal{C}$ with endpoints $a$ and $b$, we say that a point $p \in \mathbb{R}^2$ is *visible* from $q$ with respect to $\mathcal{C}$ if and only if $p$ is visible from $q$ with respect to $\mathcal{R}_{\mathcal{C}}$. The set of points that are visible with respect to $\mathcal{C}$ is called the *visibility polygon* of $\mathcal{C}$, and is denoted $\text{Vis}_{\mathcal{C}}$. We start by observing that both concepts of visibility are equivalent (hence we need not distinguish between them).

**Observation 1.** *If $\mathcal{C} = \text{Chain}(a, b)$ is an independent subchain, then $\mathcal{R}_{\mathcal{C}}$ is a simple polygon. Moreover, a point $x \in \mathcal{C}$ is visible with respect to $\mathcal{P}$ if and only if it is visible with respect to $\mathcal{C}$.*

The above observation certifies that visibility within independent chains is well-defined. Since we will only consider independent chains, from now on we omit the "with respect to $\mathcal{P}$" (or to $\mathcal{C}$), and simply say that a point $p$ is visible.

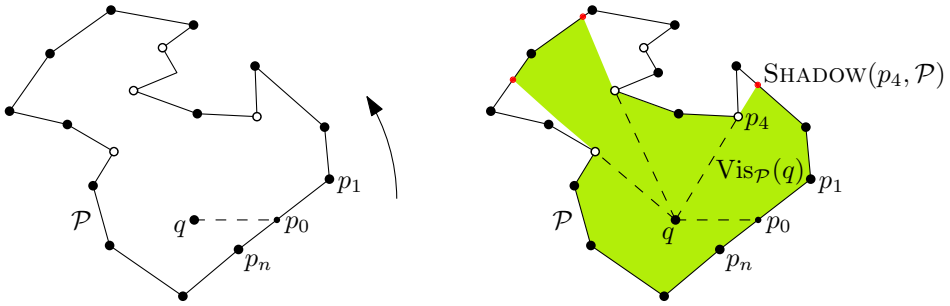


Figure 1: Left: general setting, vertices that are reflex with respect to $q$ are shown with a white point (black otherwise). Right: the visibility polygon $\text{Vis}_{\mathcal{P}}$.

True to its name, the visibility polygon of an independent chain $\mathcal{C} \subseteq \partial\mathcal{P}$ can be computed without having knowledge of the remainder of $\mathcal{P}$.

**Observation 2.** *Let $\mathcal{C} = \text{Chain}(a, b)$ be an independent chain such that $\mathcal{C} = (a, p_i, \ldots, p_j, b)$ for some $0 < i < j < n$. Let $x$ be a visible interior point of $\mathcal{C}$ lying on the edge $p_k p_{k+1}$ for some $i \leq k < j$. If we let $\mathcal{C}_1 = \{a, p_i, \ldots, p_k, x\}$ and $\mathcal{C}_2 = \{x, p_{k+1}, \ldots, p_j, b\}$, then*

$$\text{Vis}_{\mathcal{C}} = \text{Vis}_{\mathcal{C}_1}(q) \cup \text{Vis}_{\mathcal{C}_2}(q),$$

$$\text{Vis}_{\mathcal{C}_1}(q) \cap \text{Vis}_{\mathcal{C}_2}(q) = qx.$$

Given a point $p$ on the plane, let $\rho_q(p)$ be the ray emanating from $q$ and passing through $p$. We define $\theta_q(p)$ as the angle that $\rho_q(p)$ makes with the positive $x$-axis, $0 \leq \theta_q(p) < 2\pi$. We call $\theta_q(p)$ the *CCW-angle* of $p$.

We also need to define what a *reflex* vertex is in our context. Given any vertex $p_k$, the line $\ell_k$ passing through $p_k$ and $q$ splits $\mathcal{P} \setminus \ell_k$ into disjoint components. A vertex $p_k$ is *reflex with respect to $q$* if the angle at the vertex interior to $\mathcal{P}$ is more than $\pi$ and the vertices $p_{k-1}$ and $p_{k+1}$ lie on the same connected component of $\mathbb{R}^2 \setminus \ell_k$ (see Fig. 1). Observe that any vertex that is reflex with respect to $q$ is a reflex vertex (in the usual sense), but the converse is not true. Since the point $q$ is fixed, from now on we omit the "with respect to $q$" term and simply refer to these points as reflex. Note that being reflex is a local property that can be verified in $O(1)$ time. Intuitively speaking, reflex vertices with respect to $q$ are the vertices where important changes occur in the visibility polygon. That is, where the polygon boundary can change between visible or not-visible. Let $r$ be the number of reflex of vertices of $\mathcal{P}$. We also define $\bar{r}$ as the number of reflex vertices of $\mathcal{P}$ that are present in $\text{Vis}_{\mathcal{P}}$. Naturally, we always have $\bar{r} \leq r < n$.

Given two points $p$ and $p'$ on a chain $\mathcal{C} = \text{Chain}(a, b)$, we say that $p$ lies *before* $p'$ (resp. $p'$ lies *after* $p$) if, when we walk from $a$ towards $b$ along $\mathcal{C}$, we first pass through $p$ and then through $p'$. We say that a chain is *visible* if all the points of the chain are visible. A visible chain $\mathcal{C} = \text{Chain}(a, b)$ is *CCW-maximal* if no other visible chain starting at $a$ strictly contains $\mathcal{C}$. In this case, we say that $\mathcal{C}$ is the maximal chain *starting* at $a$ and *ending* at $b$.

Given a visible reflex vertex $v$ on a chain $\mathcal{C}$, we say that a point $w \neq v$ on $\partial\mathcal{P}$ is the *shadow* of $v$ if $w$ is collinear with $q$ and $v$, and $w$ is visible from $q$ (this point is denoted by $\textsc{Shadow}(v, \mathcal{C})$). Due to the general position assumption, $w$ is uniquely defined and must be an interior point of an edge. That is, each visible reflex vertex is uniquely associated to a shadow point (and vice versa). We say that a visible reflex vertex $v$ is of *type R* (resp. *type L*) if its shadow lies after (resp. before) $v$; see Fig. 2 (left). Equivalently, a vertex $v$ is of type R if $q, v, x$ and $q, v, y$ make a *right* turn (where $x$ and $y$ are the predecessor and the successor of $v$ on $\mathcal{C}$, respectively). Analogously, vertices of type L make a *left* turns instead.

A *ray shooting query* is a basic operation that, given an independent chain $\mathcal{C}$ and a point $x \in \mathcal{C}$, considers the ray $\rho_q(x)$ and reports the last visible point in $\rho_q(x)$ with respect to $\mathcal{C}$ (i.e., the one furthest away from $q$). Observe that when $x$ is a visible reflex vertex we obtain its shadow. We

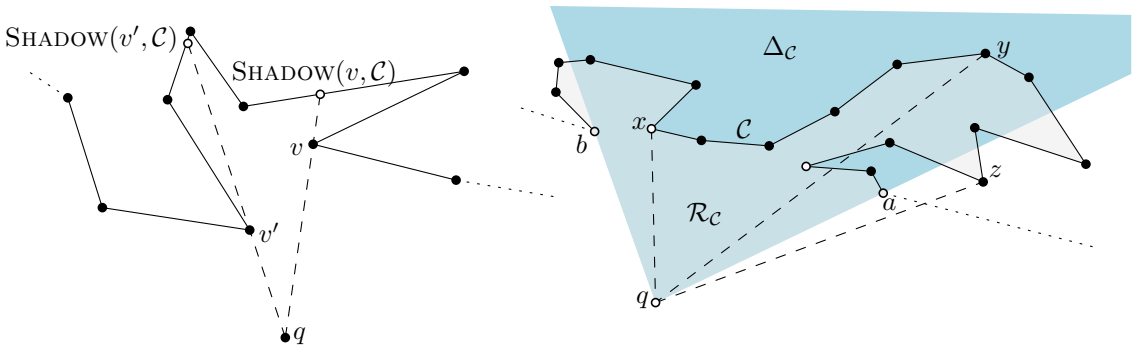


Figure 2: Left: $v$ is a reflex vertex of type R, while $v'$ is of type L. The chain between $v$ and SHADOW$(v, \mathcal{C})$, together with the segment joining them, bound a simple polygon containing no visible points other than $v$ and SHADOW$(v, \mathcal{C})$. Observe that the chain $\mathcal{C}($SHADOW$(v, \mathcal{C}),$ SHADOW$(v', \mathcal{C}))$ is CCW-maximal. Right: A polygonal chain $\mathcal{C} = \text{Chain}(a, b)$ and its associated polygon $\mathcal{R}(a, b) = \mathcal{R}_{\mathcal{C}}$. Point $x$ is visible while points $y$ and $z$ are not. Note that every visible point of $\mathcal{C}$ lies inside $\Delta_{\mathcal{C}}$.

denote the output of this operation by RAYSHOOTING$(x, \mathcal{C})$. It is easy to see that a ray shooting query can be performed in linear time, using only $O(1)$ extra variables, by scanning the edges of $\mathcal{C}$ one by one and computing their intersections with $\rho_q(x)$.

Finally, for $\mathcal{C} = \text{Chain}(a, b)$ we define $\Delta_{\mathcal{C}}$ as the cone with apex $q$ that contains every point in the plane with CCW-angle between $\theta_q(a)$ and $\theta_q(b)$; see Fig. 2 (right).

# 3 Understanding the visibility polygon

The basic scheme of our algorithms is to partition the input polygon into independent subchains so that the visibility within one subchain is unaffected by the others. We will use $\partial\mathcal{P}$ as the starting chain, thus the first chain will be closed, but the following chains will be open. Notice that since $\mathcal{P}$ is simple, any chain of it will be simple too.

In this section we present some observations about the independence between chains.

**Observation 3.** *Let $v$ be a visible reflex vertex of $\mathcal{P}$ of type R (resp. L) whose shadow is $w$. The* $\text{Chain}(v, w)$ *(resp.* $\text{Chain}(w, v)$*) contains no visible point other than its endpoints. In particular, a visible chain cannot contain an interior reflex vertex.*

The following important lemma characterizes the endpoints of CCW-maximal chains.

**Lemma 1.** *Let $\mathcal{C} =$* $\text{Chain}(a, b)$ *be an independent chain, let $p \in \mathcal{C}$ be a visible point and let $v$ the first visible reflex vertex encountered when walking from $p$ towards $b$ (or $b$ if none exists). Let* $\text{Chain}(p, p')$ *be the CCW-maximal chain starting at $p$. The point $p'$ is either equal to $v$ (if $v = b$ or $v$ is of type R), or equal to the shadow of $v$ (if $v$ is of type L).*

*Proof.* Clearly, all the points lying after $p$ are visible if and only if $p' = b$. If $p' \neq b$, then, when walking on $\mathcal{C}$, $p'$ is the last visible point of the chain before a change in visibility occurs (i.e. points at distance $\varepsilon > 0$ lying after $p'$ are not visible for sufficiently small values of $\varepsilon$). This can happen for only two reasons: either $p'$ is a reflex vertex of type R, or there is some reflex vertex $v'$ of type L such that $p'$ is the shadow of $v'$. In the former case, $p'$ is equal to $v$ since no reflex vertex lies in the interior of $\text{Chain}(p, p')$ by Observation 3. In the latter case, $v'$ must be the first visible reflex vertex lying after $p$ since no point between $p'$ and $v'$ can be visible by Observation 3. □

The following result is a direct consequence of Lemma 1 that allows us to report $\text{Vis}_{\mathcal{C}}$ of an independent chain $\mathcal{C}$ with no interior reflex vertices.

**Corollary 1.** *Let $p \in \partial\mathcal{P}$ be a visible point such that $p$ is not a reflex vertex of type R. Let $v$ be the first visible reflex vertex lying after $p$ and let $w$ be its shadow. The following statements hold:*

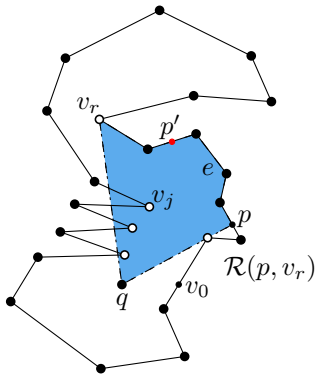


Figure 3: When the first reflex vertex $v$ lying after $p$ is not visible, there must exist a visible reflex vertex in Chain$(p, b)$ angularly located between $p$ and $v$. The one with smallest CCW-angle inside $\mathcal{R}(p, v)$ ($v'$ in the figure) will determine the change between visible and non-visible regions between $p$ and $v$.

- *If $v$ is of type R, then Chain$(p, v)$ is CCW-maximal.*
- *If $v$ is of type L, then Chain$(p, w)$ is CCW-maximal.*

# 4 Output Sensitive Algorithm

In this section we present an algorithm that will be used as stepping stone for our divide and conquer algorithm presented in Section 5. This base algorithm computes the visibility polygon of an independent chain using $O(1)$ extra variables.

For this purpose, we introduce an operation that we call NextVisReflex. This operation receives an independent chain $\mathcal{C} = \text{Chain}(a, b)$ and a visible point $p$ on $\mathcal{C}$. Its objective is to compute the next *visible* reflex vertex lying after $p$ on $\mathcal{C}$, along with its shadow. That is, let $v$ be the first reflex vertex lying after $p$ on $\mathcal{C}$. If $v$ is visible, then NextVisReflex$(p, \mathcal{C})$ should return $v$ and its shadow. Otherwise, we know that at some point when walking along the path from $p$ to $v$ we change from a visible to a non-visible region. By Lemma 1, this change occurs at the shadow of some visible reflex vertex. In this case, NextVisReflex$(p, \mathcal{C})$ should return this reflex vertex (as well as its shadow). For well-definedness purposes, we say that NextVisReflex$(p, \mathcal{C})$ should return $b$ if $\mathcal{C}$ contains no reflex vertex.

The following observation (illustrated in Fig. 3) allows us to compute NextVisReflex$(p, \mathcal{C})$ efficiently.

**Observation 4.** *For any independent chain $\mathcal{C} = \text{Chain}(a, b)$, and visible point $p \in \mathcal{C}$ that is not an R-type reflex vertex, let $v$ be the first reflex vertex encountered when walking from $p$ towards $b$ on $\mathcal{C}$ (or $b$ if none exists). If $v$ is not visible, then the CCW-maximal chain starting at $p$ ends at the shadow of the L-type reflex vertex in $\mathcal{R}(p, v)$ with smallest CCW-angle.*

**Lemma 2.** *For any independent chain $\mathcal{C}$ of $n$ vertices and a visible point $p \in \mathcal{C}$,* NextVisReflex$(p, \mathcal{C})$ *can be computed in $O(n)$ time using $O(1)$ additional variables.*

*Proof.* Let $\mathcal{C} = \text{Chain}(a, b)$ and let $v$ be the first reflex vertex lying after $p$ on $\mathcal{C}$. In $O(n)$ time we can perform a ray shooting query: if $v$ is visible then we output it (and its shadow) as the result of NextVisReflex$(p, \mathcal{C})$. Otherwise, we use Observation 4 and compute the reflex vertex on Chain$(p, b)$ with smallest CCW-angle (among those that are in $\mathcal{R}(p, v)$). This vertex is found by walking along Chain$(p, b)$ and keeping track of every time we enter or leave $\mathcal{R}(p, v)$. Note that since $p$ is visible and $\mathcal{C}$ is simple, we can only enter or leave $\mathcal{R}(p, v)$ when we cross line segment $qv$, hence this can be checked in constant time per edge of $\mathcal{C}$. Since a constant number of operations is needed per vertex, at most $O(n)$ time will be needed for computing NextVisReflex$(p, \mathcal{C})$.

Finally note that Observation 4 holds whenever $p$ is not an R-type reflex vertex. Thus, if $p$ is a reflex vertex of type R, it suffices to first compute its shadow, and return the same as NextVisReflex(Shadow($p, \mathcal{C}$), $\mathcal{C}$) would. □

Given an independent chain $\mathcal{C} = (c_1, \ldots, c_n)$, our base algorithm works as follows: start from $c_1$, use NextVisReflex to obtain the next visible reflex vertex, and report the CCW-maximal chain starting from $c_1$. We then repeat the procedure starting from the last reported vertex until we reach $c_n$ (see the details in Algorithm 1).

**Theorem 1.** *Algorithm 1 reports the visibility polygon of an independent chain of $n$ vertices and $\overline{r}$ visible reflex vertices in counterclockwise order in $O(n\overline{r})$ time, using $O(1)$ additional variables.*

*Proof.* Correctness of the algorithm is given by Lemma 2 and Corollary 1. It is easy to verify that Algorithm 1 uses a constant number of variables, hence it remains to show a bound on the running time. Notice that at each iteration of the algorithm we report a visible reflex vertex. Hence, we can charge the cost of NextVisReflex operation to the reported vertex. Since no vertex is reported twice and operation NextVisReflex needs linear time, the total running time is bounded by $O(n\overline{r})$. □

**Algorithm 1** Computing the visibility polygon of an independent chain $\mathcal{C} = (c_1, \ldots c_n)$

```
1: c_start ← c_1 (or c_start ← Shadow(c_1, C) if c_1 is an R-type reflex vertex)
2: repeat
3:    v ← NextVisReflex(c_start, C)
4:    if v = c_n then
5:       (* The remainder of the chain is visible *)
6:       c_stop ← c_n
7:       c_next ← c_n
8:    else
9:       (* We found next visible reflex v. The reported chain will depend on the type of v *)
10:      if v is of type R then
11:         c_stop ← v
12:         c_next ← Shadow(v, C)
13:      else
14:         c_stop ← Shadow(v, C)
15:         c_next ← v
16:      end if
17:   end if
18:   Report every vertex between c_start and c_stop
19:   c_start ← c_next
20: until c_start = c_n
```

# 5 A divide-and-conquer approach

We now consider the case in which we are allowed a slightly larger amount of variables. We parametrize the running time of our algorithms by the number of working variables allowed, which we denote by $s$. Our aim is to obtain an algorithm whose running time decreases as $s$ grows.

Using the result of the previous section as base algorithm, we now present a divide-and-conquer approach to solve the problem. The general scheme of our algorithm is the natural one: choose a reflex vertex $z$ inside the cone $\Delta_{\mathcal{C}}$, perform a ray shooting query to find the visible point in the direction of $z$, and split the polygonal chain into two smaller independent subchains $\mathcal{C}_1, \mathcal{C}_2$ (see Fig. 4, left). We repeat the process recursively, until either (1) a chain $\mathcal{C}$ has a constant number

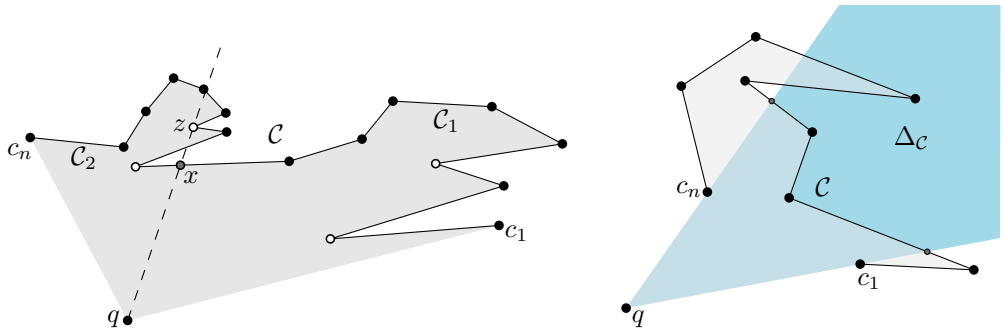


Figure 4: Left: Split of $\mathcal{C}$ into two subchains $\mathcal{C}_1, \mathcal{C}_2$ using a visible point $x$ in the direction of a reflex vertex $z$. Right: A polygonal chain $\mathcal{C}$ with no reflex vertices inside the cone $\Delta_{\mathcal{C}}$, only one subchain of $\mathcal{C}$ is visible.

of reflex vertices (see Fig. 4, right) or (2) the depth of the recursion is such that we would exceed the number of allowed working variables. Whenever either of these two conditions is met, we compute the visibility polygon of the chain using Algorithm 1. See a scheme of this approach in Algorithm 2.

**Algorithm 2** Given a polygonal chain $\mathcal{C} = (c_1, \ldots, c_n)$ such that $c_1, c_n$ are both visible points of $\mathcal{P}$ and a positive integer depth $d$ (initially 1), compute $\text{Vis}_{\mathcal{C}}$

```
k ← number of reflex vertices of C inside the cone Δ_C
if k ≤ 2 or d ≥ h(s) then
    Run Algorithm 1 on C
else
    v ← findPartitionVertex(C)
    x ← RayShooting(v, C)
    Algorithm 2({c_1, ..., x}, d + 1)
    Algorithm 2({x, ..., c_n}, d + 1)
end if
```

In order to control the depth of the recursion we use a depth counter, hence the algorithm stops dividing once $d = h(s)$ (for some value $h(s) \in O(s)$ that will be determined later). Note that the split direction is decided by subroutine FindPartitionVertex($\mathcal{C}$). This procedure should give a direction so that the resulting subchains $\mathcal{C}_1$ and $\mathcal{C}_2$ have roughly the same complexity. Naturally, it must also run reasonably fast and use $O(s)$ variables.

In this section we propose two different methods to choose the partition vertex. The first one uses the approximate median finding algorithm of Raman and Ramnath [22]. The second one is randomized, and simply chooses a random reflex vertex among those lying in the cone $\Delta_{\mathcal{C}}$. We first show that, regardless of the partition method used, the visibility polygon will be correctly computed.

**Lemma 3.** *Algorithm 2 correctly reports the visibility polygon of an independent chain in counterclockwise order, using $O(s)$ variables.*

*Proof.* The divide-and-conquer approach repeatedly partitions the input into independent chains. Each subchain will eventually be reported. By Observation 2, the union of reported vertices is equal to the visibility polygon, hence correctness is derived from the correctness of Algorithm 1.

Regarding space, the subroutines called in this algorithm (findPartitionVertex and RayShooting) use $O(s)$ and $O(1)$ variables, respectively. Once the procedure finishes, their working space can be reused for further calls, hence we never use more than $O(s)$ working space at the same time. It remains to consider the memory used implicitly for handling the recursion.

Since each step of the algorithm needs a constant number of variables, the total memory needed will be proportional to the recursion depth. Since $h(s) \in O(s)$, the claim is shown. □

In what follows we present two implementations of FindPartitionVertex.

## 5.1 Deterministic variant

We start by giving a deterministic algorithm for FindPartitionVertex($\mathcal{C}$) using $O(s)$ extra variables. For this purpose, we will use the algorithms by Raman and Ramnath presented in [22], which compute an approximation of the median value of a given set using reduced workspace.

Given a chain $\mathcal{C}$, let $\Theta = \{\theta_q(v_i) : v_i \text{ is a reflex vertex of } \mathcal{C} \text{ lying inside } \Delta_{\mathcal{C}}\}$. An element $\theta_q(v_i) \in \Theta$ is called a 2/3-*median of* $\Theta$ if there are at most $2|\Theta|/3$ elements in $\Theta$ smaller that $\theta_q(v_i)$ and at most $2|\Theta|/3$ greater that $\theta_q(v_i)$. Given two elements $z, z'$ of $\Theta$ such that $z < z'$, we say that $z, z'$ is an *approximate median pair* if at most $|\Theta|/2$ elements of $\Theta$ are smaller than $z$, at most $|\Theta|/2$ lie between $z$ and $z'$, and at most $|\Theta|/2$ elements are greater that $z'$. Notice that if $z, z'$ is an approximate median pair, then either $z$ or $z'$ is a 2/3-median of $\Theta$. Moreover, we can determine which of the two is a 2/3-median with one scan of $\mathcal{C}$. Thus, we say that every approximate median pair induces a 2/3-median of $\Theta$.

Raman and Ramnath [22] presented two algorithms to find an approximate median pair. The first one is used whenever $s \in o(\log r)$, and allows us to find a 2/3-median of $\Theta$ in $O(snr^{1/s})$ time using $O(s)$ variables (Lemma 5 of [22], assigning their parameter $p$ to $r^{1/s}$). For the case $s \in \Omega(\log r)$, they propose another algorithm (stated in Lemma 3 of [22]) that computes an approximate median pair in $O(n \log r)$ time, using $O(\log r)$ variables. We note that Raman and Ramnath used these algorithms to afterwards obtain the exact median, but in here we only need an approximation.

FindPartitionVertex($\mathcal{C}$) will execute the approximate median pair algorithm, and return the reflex vertex $v$ that induces a 2/3-median of $\Theta$. By construction, each of the two cones obtained from $\Delta_{\mathcal{C}}$, by shooting a ray through $v$, will contain at most 2/3 of the reflex vertices in $\Delta_{\mathcal{C}}$.

Let $P(n, r)$ be the running time of the approximate median finding algorithm on a chain $\mathcal{C}$ of length $n$ with $r$ reflex vertices inside $\Delta_{\mathcal{C}}$ (that is, $P(n, r) = O(snr^{1/s})$ if $s \in o(\log r)$, $P(n, r) = O(n \log r)$ otherwise). We set $h(s) = s \log_{3/2} 2 \approx 1.71s$ (if $s \in o(\log r)$) or $s = \log_{3/2} r$ (otherwise). Observe that in both cases we have $h(s) \in O(s)$. We now prove an upper bound on the running time of Algorithm 2 with this implementation of FindPartitionVertex($\mathcal{C}$).

**Theorem 2.** *For any $s \in O(\log r)$, $\mathrm{Vis}_{\mathcal{C}}$ can be computed in $O(\frac{nr}{2^s} + n \log^2 r)$ time using $O(s)$ variables.*

*Proof.* Consider any node $u_i$ of the recursion tree of the algorithm and let $\mathcal{C}_i$ be the chain processed at this node. Let $n_i$ and $r_i$ be the size of $\mathcal{C}_i$ and the number of reflex vertices lying inside $\Delta_{\mathcal{C}_i}$, respectively.

If $u_i$ is a non-terminal node of the recursion, then the running time at $u_i$ is bounded by $O(P(n_i, r_i))$. Note that a vertex of the input can only appear in at most two chains of the same depth. Hence, the total cost of all non-terminal nodes of a fixed level $j$ in the recursion tree is bounded by $\sum_{u_i} O(P(n_i, r_i)) \leq O(P(n, r))$. By definition, there are at most $h(s)$ levels of recursion, hence the time spent in all the non-terminal executions of the algorithm is bounded by $O(P(n, r)h(s))$.

It remains to consider the time spent in the terminal nodes. Each terminal node $u_i$ will need $O(n_i \overline{r}_i)$ time, where $\overline{r}_i$ denotes the number of visible reflex vertices on $\mathcal{C}_i$. Recall that terminal nodes are only executed whenever either $\mathcal{C}_i$ has a constant number of reflex vertices or we have reached $h(s)$ levels of recursion. Further note that, at each level of recursion at least a third of the reflex vertices are discarded. In particular, we have that $\overline{r_i} \leq r(\frac{2}{3})^{h(s)}$.

Similar to non-terminal nodes, vertices cannot be present in more than two terminal nodes. Thus, the total time spent in the terminal nodes is bounded by

$$\sum_{u_i} O(n_i \overline{r_i}) \leq \sum_{u_i} O(n_i r(2/3)^{h(s)})) \leq O(nr(2/3)^{h(s)}).$$

Therefore, the total time spent by the algorithm becomes $O(P(n,r)h(s) + nr(2/3)^{h(s)})$.

This expression can be simplified by distinguishing between different workspace sizes.

**Case** $s \in o(\log r)$**.** In this case we have $P(n,r)h(s) = O(s^2 nr^{1/s})$, and in particular $O(s^2 nr^{1/s}) \in O(nr^{1/3})$ (since $s$ is a parameter that can be chosen to be at least 3). Further recall that $h(s) = s\log_{3/2} 2$, and in particular $(2/3)^{h(s)} = 2^{-s}$. Thus, the second term simplifies to $\frac{nr}{2^s}$. Since $s \leq \log_2(r)/3$ and the running time decreases as $s$ grows, we have $\frac{nr}{2^s} \geq \frac{nr}{r^{1/3}} \in \Omega(n\sqrt{r})$.

That is, the running time of our algorithm is expressed as the sum of two terms that, when $s \in o(\log r)$, the first one is at most $O(nr^{1/3})$ whereas the second one is at least $\Omega(n\sqrt{r})$. Asymptotically speaking, the first one can be ignored, and the running time is dominated by $O(\frac{nr}{2^s})$ (i.e., applying Algorithm 1 to each terminal node).

**Case** $s \in \Theta(\log r)$**.** In this case we can use the faster method of Raman and Ramnath (i.e. $P(n,r) = O(n\log r)$). Recall that in this case we set $h(s) = \log_{3/2} r$, hence $(2/3)^{h(s)} = 1/r$.

In particular, the running time of the second term simplifies to $O(nr(2/3)^{h(s)}) = O(n)$. Since $s \in \Theta(\log r)$, the running time is dominated by the first term (i.e., finding the split direction), which is $O(P(n,r)h(s)) = O(n\log^2 r)$.

Observe that in both cases the running time is bounded by $O(\frac{nr}{2^s} + n\log^2 r)$, hence the claim holds. □

**Remark** Note that, although we only consider algorithms that use up to $O(\log r)$ variables, one could study what happens whenever more space is allowed. However, we note that increasing the size of our workspace will not reduce the running time, since the time bottleneck of this approach is determined by the approximate median method of Raman and Ramnath.

## 5.2 Randomized approach

Whenever $s \in \Theta(\log n)$, the running time of the previous algorithm is dominated by the FindPartitionVertex procedure. Motivated by this, in this section we consider a faster (albeit randomized) partition method.

The randomized method proceeds as follows: let $k$ be the number of reflex vertices of $\mathcal{C}$ lying inside $\Delta_{\mathcal{C}}$ (note that $k$ can be computed in linear time by scanning $\mathcal{C}$). Select a random number $i$ between 1 and $k$ (uniformly at random). The idea is to output the $i$-th reflex vertex inside $\Delta_{\mathcal{C}}$ (computed by walking counterclockwise along $\mathcal{C}$). However, we must first check that this vertex will make a balanced partition. In order to check so, we make another scan of $\mathcal{C}$, and we count the angular rank of the $i$-th reflex vertex (among the reflex vertices in $\Delta_{\mathcal{C}}$). If its rank is between $k/3$ and $2k/3$, we use it as partition vertex. Otherwise, we pick another random number and repeat the process until such a vertex is found.

**Lemma 4.** *The randomized version of* FindPartitionVertex *has expected running time* $O(n)$.

*Proof.* The probability that, when choosing a reflex vertex uniformly at random, we pick one whose rank is between 1/3 and 2/3 is exactly 1/3. If each time we make the choices independently, we are performing Bernoulli trials whose probability of success is 1/3. Hence, the expected number of times we have to choose a random index is a constant (three in this case). For each try we only need to check its rank (which can be done in $O(n)$ time by performing two scans of the input). Therefore we conclude that the expected running time of FindPartitionVertex is $O(n)$. □

**Theorem 3.** *For any* $s \in O(\log r)$*,* $\mathrm{Vis}_{\mathcal{C}}$ *can be computed in* $O(\frac{nr}{2^s} + n\log r)$ *expected time using* $O(s)$ *variables.*

*Proof.* The proof is identical to the proof of Theorem 2, just taking into account that now $P(n,r) = O(n)$, hence the running time of the second term is decreased by a $\log r$ factor. □

Observe that this algorithm is faster than the previous one when $s \in \Theta(\log r)$. We conclude by summarizing the different algorithms presented in this paper.

**Theorem 4.** *Given a polygon $\mathcal{P}$ of $n$ vertices, out of which $r$ are reflex, the visibility polygon of a point $q \in \mathcal{P}$ can be computed in $O(n\bar{r})$ time using $O(1)$ variables (where $\bar{r}$ is the number of reflex vertices of $\mathrm{Vis}_{\mathcal{P}}$) or $O(\frac{nr}{2^s})$ time using $O(s)$ variables (for any $s \in o(\log r)$). If $\Omega(\log r)$ variables are available, the running time decreases to $O(n \log^2 r)$ time or $O(n \log r)$ randomized expected time.*

# References


[1] O. Aichholzer, M. Korman, A. Pilz, and B. Vogtenhuber. Geodesic order types. In *COCOON*, pages 216–227, 2012.

[2] S. Arora and B. Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[3] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *CoRR*, abs/1112.5904, 2011.

[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] T. Asano, W. Mulzer, and Y. Wang. Constant-work-space algorithm for a shortest path in a simple polygon. In *WALCOM*, pages 9–20, 2010.

[6] T. Asano and G. Rote. Constant-working-space algorithms for geometric problems. In *CCCG*, pages 87–90, 2009.

[7] L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space-time trade-offs for stack-based algorithms. Unpublished manuscript (currently under review), 2012.

[8] L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing the visibility polygon using few variables. In *ISAAC*, pages 70–79, 2011.

[9] P. Bose, P. Carmi, F. Hurtado, and P. Morin. A generalized Winternitz theorem. *Journal of Geometry*, 100:29–35, 2011.

[10] P. Bose, E. D. Demaine, F. Hurtado, J. Iacono, S. Langerman, and P. Morin. Geodesic ham-sandwich cuts. *Discrete & Computational Geometry*, 37(3):325–339, 2007.

[11] T. Chan and E. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.

[12] T. M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6:26:1–26:16, April 2010.

[13] M. De, A. Maheshwari, and S. C. Nandy. Space-efficient algorithms for visibility problems in simple polygon. *CoRR*, abs/1204.2634, 2012.

[14] G. N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987.

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

[16] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, pages 58–66, 2001.

[17] 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.

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

[19] J. Munro and M. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.

[20] J. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165:311–323, 1996.

[21] J. O'Rourke. Visibility. In *Handbook of Discrete and Computational Geometry*, chapter 28, pages 643–664. CRC Press, Inc., 2nd edition, 2004.

[22] V. Raman and S. Ramnath. Improved upper bounds for time-space tradeoffs for selection with limited storage. In *SWAT*, pages 131–142, 1998.