Computing a visibility polygon using few variables

Luis Barba[1], Matias Korman[2], Stefan Langerman[⋆2], and Rodrigo I. Silveira[⋆⋆3]

[1] Universidad Nacional Autónoma de México (UNAM), Mexico D.F., Mexico.
l.barba@uxmcc2.iimas.unam.mx
[2] Université Libre de Bruxelles (ULB), Brussels, Belgium.
{mkormanc,slanger}@ulb.ac.be
[3] Universitat Politècnica de Catalunya (UPC), Barcelona, Spain.
rodrigo.silveira@upc.edu

Abstract. We present several algorithms for computing the visibility polygon of a simple polygon $\mathcal{P}$ from a viewpoint inside the polygon, when the polygon 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. The next two algorithms use $O(\log r)$ variables, and output the visibility polygon 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 $\mathcal{P}$.

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[10]. It computes the visibility polygon from a point in linear time and space. We refer the reader to the survey of O'Rourke [13] and the book of Gosh [8] for an extensive survey 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 algorithms not only provides an interesting trade-off between running time and memory needed, but also is very useful in portable devices where important hardware constraints are present (such as the ones found in digital cameras or portable phones).

A significant amount of research has focused on the design of algorithms that use few variables, some of them even dating from the 80s [11]. Although many

⋆ Maître de Recherches du FRS-FNRS.
⋆⋆ Funded by the FP7 Marie Curie Actions Individual Fellowship PIEF-GA-2009-251235.

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 logarithmic amount).

One of the most studied problems in this setting is that of selection. For any constant $\epsilon \in (0,1)$, Munro and Raman [12] give an algorithm that runs in $O(n^{1+\epsilon})$ time and uses $O(1/\epsilon)$ variables. Frederickson [7] improved this result with an algorithm whose running time is $O(n \log^* s + n \log n / \log s)$ when s working variables are available (and $s \in \Omega(\log n) \cap O(2^{\log n / \log^* n})$). Whenever only $O(\log n)$ variables are available, Raman and Ramnath[14] gave an algorithm whose running time is $O(n \log^2 n)$. More recently Chan [5] provided several lower bounds for performing selection with few variables and modified the $O(n \log \log_s n)$ expected time algorithm of Munro and Raman so that it works for any input array.

Recently, there has been an interest in finding algorithms for geometric problems that use a constant number of variables: Given a set of n points, the well-known gift-wrapping algorithm (also known as Jarvis march [15]) can be used to report the points on the convex hull in $O(n\bar{h})$ time using a constant number of variables, where $\bar{h}$ is the number of vertices on the convex hull. Asano and Rote [3] and afterwards Asano *et al.* [2] gave efficient methods for computing well-known geometric structures, such as the Delaunay triangulation, Voronoi diagram and minimum spanning tree using a constant number of variables (in $O(n^2)$, $O(n^2)$ and $O(n^3)$ time, respectively) Observe that, since these structures have linear size, they are not stored but reported.

To the best of our knowledge, there is no known method that computes the visibility polygon of a given point and uses few variables. The question of finding a constant workspace algorithm to compute the visibility polygon in a polygon of size n was explicitly posed as an open problem by Asano *et al.* [1]. A natural approach to the problem would be to transform the linear time-algorithm of Joe and Simpson [10] to a constant workspace algorithm. However, their algorithm cannot be directly adapted to our setting, since their method uses a stack which could contain up to $\Omega(n)$ vertices.

Results In this paper, we consider a completely different approach. It is easy to realize that the differences between $\mathcal{P}$ and the visibility polygon of a given point depends on the number of reflex vertices. For example, if $\mathcal{P}$ is a convex polygon, the two polygons will be equal, but we cannot expect this to hold when the number of reflex vertices grows. Therefore whenever possible we will 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 in the input and the number of them also present 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 [4] and references found therein for a deep review of existing similar results.

In Section 3 we give an output-sensitive algorithm that reports the vertices of the visibility polygon in $O(n\bar{r})$ time using $O(1)$ variables. In Section 4 we

show that if we are allowed to use $O(\log r)$ variables, the problem can be solved in $O(n \log r)$ randomized expected time or $O(n \log^2 r)$ deterministic time.

2 Preliminaries

Model definition and considerations on input/output precision We use a slight variation of the constant workspace model (sometimes also referred as *log space*), introduced by Asano [3]. In this model, an algorithm can use a constant number of variables and 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 a part of the workspace.

The input of the problem is a polygon $\mathcal{P}$ in a read-only data structure. In the usual constant workspace model, we are allowed to perform random access to any of the values of the input in constant time. However, in this paper we consider a weaker model in which the only allowed operation to the input is obtaining coordinates of the next counterclockwise vertex of a given vertex of $\mathcal{P}$. This is the case in which, for example, the vertices of $\mathcal{P}$ are given in a list in counterclockwise order.

Many other similar models exist in the literature. We note that in some of them (like the *streaming* [9] or the *multi-pass* model [6]) the values of the input can only be read once or a fixed number of times. We follow the model of constant workspace and allow scanning the input as many times as necessary.

The algorithm given in Section 3 uses a weaker model than the constant workspace model (since random access to the vertices of the input is not used). The methods given in Section 4 use a logarithmic number of variables, providing another compromise between the space and time complexity of this problem.

Let n be the number of vertices of $\mathcal{P}$. 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. Observe that 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 v_i and v_j and the line passing through v_k and v_l" (where v_i, v_j, v_k and v_l are vertices of the input).

Other definitions The boundary of $\mathcal{P}$ is denoted by $\partial\mathcal{P}$. We regard $\mathcal{P}$ as a closed subset of the plane. We are also given a point q inside $\mathcal{P}$, from where the visibility polygon needs to be computed. The segment connecting points p and q is denoted by pq. We say that a point $p \in \mathcal{P}$ is visible (with respect to q) if and only if $pq \subset \mathcal{P}$ (otherwise we say that p is *not visible*). The set of points that are visible from q is denoted by $\mathrm{Vis}_{\mathcal{P}}(q)$ and is called the *visibility polygon* of q. It

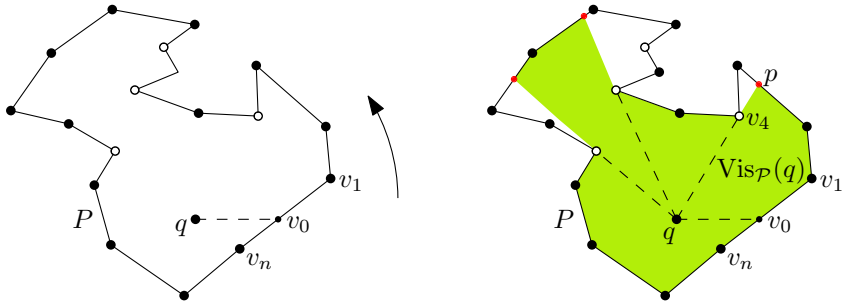

Fig. 1. Left: general setting, vertices that are reflex with respect to q are shown with a white point (black otherwise). Right: the visibility polygon $\mathrm{Vis}_{\mathcal{P}}(q)$ of q; point p is the *shadow* of vertex v_4.

is easy to see that the visibility polygon of q is a closed polygon whose vertices are either vertices of $\mathcal{P}$ or the intersection point between a segment of $\mathcal{P}$ and a ray emanating from q (see Figure 1).

From now on, for simplicity in the explanation, we assume that no line passes through q and two vertices of $\mathcal{P}$. We note, however, that our algorithms can be adapted to work in the general case with only minor changes. We also need to define what a *reflex* vertex is in our context. Given any vertex v_k, the line ℓ_k passing through v_k and q splits $\mathcal{P} \setminus \ell_k$ into disjoint components. We say that vertex v_k is *reflex with respect to* q if the angle at the vertex interior to $\mathcal{P}$ is more than π and the vertices v_{k-1} and v_{k+1} lie on the same connected component of $\mathbb{R}^2 \setminus \ell_k$ (see Figure 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.

Intuitively speaking, reflex vertices with respect to q are the vertices where topological changes might occur in the visibility polygon. That is, the positions where the polygon boundary can change between visible or invisible. 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. Let r be the number of reflex vertices of $\mathcal{P}$. We also define $\bar{r}$ as the number of reflex vertices of $\mathcal{P}$ that are present in $\mathrm{Vis}_{\mathcal{P}}(q)$. Observe that we always have $\bar{r} \leq r < n$.

Given any two points $p_1, p_2 \in \partial\mathcal{P}$, there is a unique path that travels counterclockwise along the boundary of $\mathcal{P}$ from p_1 to p_2. Let $\mathrm{Chain}(p_1, p_2)$ be the set of points traversed in this path, including both p_1 and p_2 (this set is called the *chain* between p_1 and p_2). We say that a chain is *visible* if all the points of the chain are visible from q. A visible chain $\mathcal{C} = \mathrm{Chain}(p_1, p_2)$ is *CCW-maximal* if it is visible and no other visible chain starting at p_1 strictly contains $\mathcal{C}$. We will use the term $\mathrm{Chain}(p, p)$ to denote the whole polygon boundary.

Given a point $p \in \mathcal{P}$, let $\theta(p)$ be the angle that the ray emanating from q and passing through p makes with the positive x-axis, $0 \leq \theta(p) < 2\pi$. Let v_0 be the closest point on $\partial\mathcal{P}$ to q, such that $\theta(v_0) = 0$. It is easy to see that v_0 is visible and can be computed in linear time. Without loss of generality, we will treat v_0

as a vertex (even though it does not need to be one). Moreover, we will assume that the vertices are numbered such that v_0 is on edge $v_n v_1$.

A point p on $\partial\mathcal{P}$ is the *shadow* of a reflex vertex v if p is collinear with q and v, and p is visible from q. Due to the general position assumption, v must be unique and p must be an interior point of an edge. As a result, each reflex vertex is uniquely associated to a shadow point (and *vice versa*).

In this paper, we will often use a ray shooting-like operation, which we call RayShooting(p). This is a basic operation that given a point $p \in \partial\mathcal{P}$, considers the ray $\rho(q, p)$ and reports the first point of $\mathcal{P}$ hit by the ray. This operation can be done in linear time, by scanning the edges of $\mathcal{P}$ one by one and reporting the point closest to q that intersects the ray. Another similar operation that our algorithms will use is computing the shadow of a given reflex vertex v. For that case, we will use the FindShadow(p) operation, which gives the shadow of p if p is a reflex vertex (p otherwise). Finally we note that, due to space constraints, some of the proofs of this paper have been omitted.

3 An $O(n\bar{r})$ algorithm using $O(1)$ variables

In this section we present an output-sensitive algorithm that spends linear time for each reflex vertex in the visibility polygon. The idea of the algorithm is to compute maximal visible chains as they appear on the boundary of $\mathcal{P}$. Each iteration of the algorithm starts from a point that is known to be visible, and finds the last point of the CCW-maximal chain that starts at that point. This is repeated until the initial vertex is found again.

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

Lemma 1. *Let $p \in \partial\mathcal{P}$ be a point visible from q, and let $\mathcal{C} = \text{Chain}(p, p')$ be a CCW-maximal chain. Then p' is either (i) equal to p, (ii) a reflex vertex of $\mathcal{P}$, or (iii) the* shadow *of some reflex vertex of $\mathcal{P}$.*

Based on the previous lemma, the algorithm will start from v_0 and walk on $\partial\mathcal{P}$ in counterclockwise direction, identifying the endpoint of the current CCW-maximal chain.

Let p be a visible point and let v_r be the first reflex vertex found on $\partial\mathcal{P}$ when going counterclockwise from p. $\text{Chain}(p, v_r)$ and the segments qp and qv_r define a region that we will denote by $\mathcal{R}(p, v_r)$. The following observation is crucial for the algorithm.

Lemma 2. *Let p be a visible point and let v_r be the first reflex vertex encountered on $\partial\mathcal{P}$ when going from p in counterclockwise direction. Then v_r is visible if and only if $\mathcal{R}(p, v_r)$ contains no vertex from $\mathcal{P}$, or equivalently, if and only if no edge of $\mathcal{P}$ crosses the segment qv_r.*

Let FindNextReflexVertex(p) be a routine that returns the first reflex vertex v_r found on $\partial\mathcal{P}$ counterclockwise from p in $O(n)$ time. If no such vertex exists the remaining portion of $\mathcal{P}$ is convex (and thus all vertices until v_0 can be

reported as visible). Otherwise, vertex v_r is identified and we need to find the end of the chain p', which starts at p. For that we will determine if $\mathcal{R}(p, v_r)$ is empty, or equivalently, if $\textsc{RayShooting}(v_r) = v_r$.

If so, we have $p' = v_r$ and we can report all vertices in the chain $\text{Chain}(p, v_r)$. Then the next visible chain must start at the shadow of v_r, which can be found with the $\textsc{FindShadow}(v_r)$ operation. Otherwise, from Lemma 1 we know that v_r is not visible and p' must be the shadow of some reflex vertex in $\text{Chain}(v_r, v_0)$. To find that reflex vertex we need one more observation.

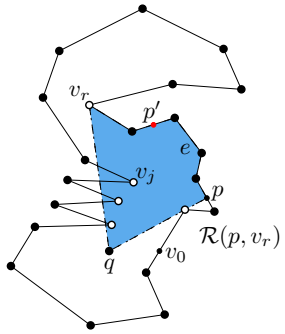

Fig. 2. When the next reflex vertex, v_r in the figure, is not visible, the end of the current visibility chain (p') can be found by walking on $\partial\mathcal{P}$. The shaded region is $\mathcal{R}(p, v_r)$.

Observation 1 *Let p be a visible point and let v_r be the first reflex vertex encountered on $\partial\mathcal{P}$ when going from p in counterclockwise direction. If v_r is not visible, then the CCW-maximal chain starting at p ends at the shadow of the reflex vertex in $\mathcal{R}(p, v_r)$ with smallest CCW-angle with respect to the segment qp.*

Therefore the algorithm only needs to find the reflex vertex with smallest angle that lies inside $\mathcal{R}(p, v_r)$, which can be done in $O(n)$ time by simply walking on $\partial\mathcal{P}$ and keeping track of the intersections with the line segment qv_r. Figure 2 illustrates a step of the algorithm.

Observe that in all cases, the total time spent in finding the end of the current chain and the beginning of the next one is $O(n)$. Hence, it follows that the total time of the algorithm is now $O(\bar{r}n)$, where $\bar{r}$ is the number of vertices reflex w.r.t. q in the visibility polygon of q. Moreover, this algorithm can be implemented such that only one pass through the polygon vertices is needed for each reflex vertex that appears in the output (plus one initial pass for finding v_0). Algorithm 1 sketches how the algorithm would look. Note also that such implementation requires only 3 variables and 1 bit of additional space.

Theorem 1. *The visibility polygon of a point q can be computed in $O(n\bar{r})$ time using constant workspace, where $\bar{r}$ is the number of reflex vertices that are part of the visibility polygon. More precisely, it can be computed in $(\bar{r}+1)$ passes through the input and using 3 variables and 1 bit.*

4 An $O(n \log r)$ algorithm using $O(\log r)$ variables

In this section we take a completely different approach to solve the problem. We consider the visibility problem in polygonal chains (instead of the whole polygon) and use a divide and conquer approach to split the problem into two subproblems. For this purpose, we must adjust the visibility definitions to make them consistent for chains.

Algorithm 1 Output-sensitive algorithm for visibility polygon

```
1: p ← v_0, r ← +∞
2: repeat
3:    Walk starting from p until next reflex vertex before v_0
4:    r ← index of reflex vertex found (or +∞ if none found)
5:    if r < +∞ then
6:       Walk from v_r until p, keep track of vertex with smallest angle inside R(p, v_r)
7:       j ← vertex with smallest angle in R(p, v_r) (j=+∞ if none found)
8:       if j = +∞ then
9:          (* v_r is visible *)
10:         Walk and report all points from p until finding v_r, including v_r (but not p)
11:         Let v_s be the shadow of v_r
12:         p ← v_s
13:      else
14:         (* v_j found, v_r is not visible *)
15:         Let p' be the shadow of v_j
16:         Walk and report all points from p until finding p', including p'
17:         p ← v_j
18:      end if
19:   end if
20: until r = +∞
21: Report all vertices between p and v_0
```

Let $\mathcal{C} = \{p_0, \ldots, p_i, \ldots, p_m\}$ be a polygonal subchain of the boundary of $\mathcal{P}$, such that p_0, p_m are both visible points on the boundary of $\mathcal{P}$ and p_i is a vertex of $\mathcal{P}$, $1 \leq i \leq m-1$. Assume without loss of generality that $\alpha = \theta(p_0) < \theta(p_m) = \beta$, and let $\mathcal{P}_\mathcal{C} = \{q, p_0, p_1, \ldots, p_m, q\}$ be the polygon contained in $\mathcal{P}$ obtained by joining q with both endpoints of $\mathcal{C}$; see Figure 3. We say that a point x on a polygonal chain $\mathcal{C}$ is $\mathcal{C}$-visible (from q), if the segment qx is completely contained in the polygon $\mathcal{P}_\mathcal{C}$. Let $\mathrm{Vis}_\mathcal{C}(q) = \{x \in \mathcal{C} : x \text{ is } \mathcal{C}\text{-visible}\}$ be the set of all $\mathcal{C}$-visible points of $\mathcal{C}$.

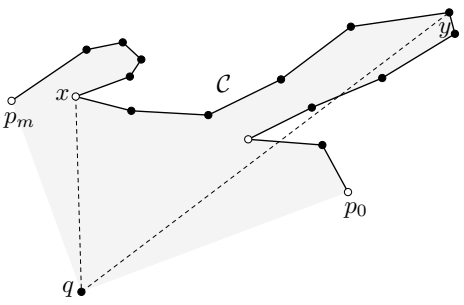

Fig. 3. Polygonal chain $\mathcal{C}$ and its associated polygon $\mathcal{P}_\mathcal{C}$. Point x is $\mathcal{C}$-visible, y is not.

Lemma 3. *Let $\mathcal{C} = \{p_0, \ldots, p_m\}$ be a polygonal chain such that p_0, p_m are both visible points on the boundary of $\mathcal{P}$, and let x be a visible point inside $\mathcal{C}$ lying on the edge $p_j p_{j+1}$. If $C_1 = \{p_0, \ldots, p_j, x\}, C_2 = \{x, p_{j+1}, \ldots, p_m\}$, then $\mathrm{Vis}_\mathcal{C}(q) = \mathrm{Vis}_{C_1}(q) \cup \mathrm{Vis}_{C_2}(q)$ and $\mathrm{Vis}_{C_1}(q) \cap \mathrm{Vis}_{C_2}(q) = x$.*

Using a similar argument it is easy to see that if x is $\mathcal{C}$-visible, and both p_0, p_m are visible from q, then x is a visible point of $\mathcal{P}$. Thus Lemma 3 allows us

to divide the problem of finding $\mathrm{Vis}_{\mathcal{P}}(q)$ into a series of subproblems that can be solved independently without compromising the output.

Let $\Delta(\mathcal{C})$ be the interior of the cone with apex q defined by the rays going from q and passing through both endpoints of $\mathcal{C}$ (and crossing the interior of the chain). Our divide and conquer algorithm is as follows. On each step of the algorithm we choose a random reflex vertex z inside the cone $\Delta(\mathcal{C})$, we then perform a ray shooting query to find the first point x on $\mathcal{C}$ in the direction of z. We split the polygonal chain $\mathcal{C}$ at x, thereby obtaining two subchains $\mathcal{C}_1, \mathcal{C}_2$; see Figure 4. We repeat the process recursively first on $\mathcal{C}_1$ and then on $\mathcal{C}_2$, until $\mathcal{C}$ is split into a series of subchains, each containing no reflex vertex inside their associated cone. Since the changes in visibility occur only at reflex vertices, the visible vertices inside each split subchain can be reported independently in the order in which they are found; see Figure 4.

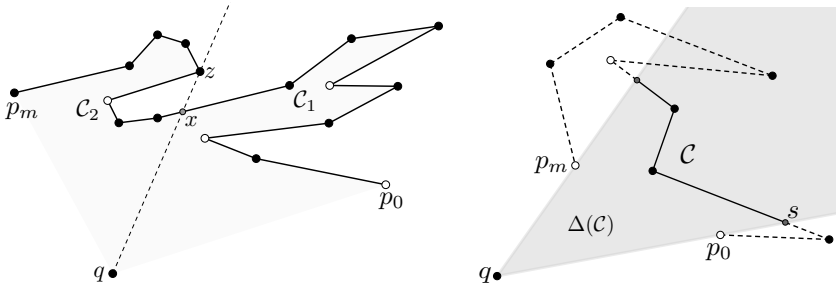

Fig. 4. Left: Split of $\mathcal{C}$ into two subchains $\mathcal{C}_1, \mathcal{C}_2$ using a visible point x in the direction of a random reflex vertex z. Right: A polygonal chain $\mathcal{C}$ with no reflex vertices inside the cone $\Delta(\mathcal{C})$, note that only one subchain of $\mathcal{C}$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

The main algorithm presented in this section is summarized in Algorithm 2. The subroutine ReportVisibleChain(x) takes a visible point x on $\mathcal{C}$, walks from x towards p_m and reports every vertex until finding an edge e of $\mathcal{C}$ intersecting the ray $\rho(q, p_m)$, once found that intersection is reported and the subroutine ends.

Lemma 4. *Algorithm 2 reports every visible edge of* $\mathrm{Vis}_{\mathcal{C}}(q)$ *in counterclockwise order.*

Lemma 5. *The expected running time of Algorithm 2 is* $O(n \log r)$.

Proof. (Sketch) The running time of Algorithm 2 can be analyzed in a similar way to the one of quicksort. On each step, a random pivot is chosen and it is used to split the chain $\mathcal{C}$, such that each subchain obtained contains in expectation a fraction of the reflex vertices. Thus if we look at the recursion tree, it is easy to see that on each level $O(n)$ operations are needed and the expected depth of the tree is $O(\log r)$, resulting in an expected running time of $O(n \log r)$. □

Algorithm 2 Given a polygonal chain $\mathcal{C} = \{p_0, \ldots, p_m\}$ such that p_0, p_m are both visible points of $\mathcal{P}$, algorithm to compute $\mathrm{Vis}_{\mathcal{C}}(q)$

```
1: k ← number of reflex vertices of C inside the cone Δ(C)
2: if k = 0 then
3:    s ← FindShadow(p_0)
4:    if s ≠ p_0 then
5:       Report p_0
6:    end if
7:    ReportVisibleChain(s)
8: else
9:    r ← random number in {1, ..., k}
10:   z ← the r-th reflex vertex of C inside Δ(C)
11:   x ← RayShooting(z)
12:   Call Algorithm 2 on C_1 = {p_0, ..., x}
13:   Call Algorithm 2 on C_2 = {x, ..., p_m}
14: end if
```

Theorem 2. *The visibility polygon of point q can be computed in $O(n \log r)$ expected time using $O(\log r)$ variables.*

Proof. (Sketch) In each step of the recursion a constant number of variables are needed. Hence, the number of words used by the algorithm is proportional to the depth of the recursion. In order to avoid using an excessive amount of memory, we use a standard trick of restarting the algorithm whenever the recursion tree becomes too deep. □

We now describe a deterministic variant of Algorithm 2 that runs on $O(n \log^2 r)$ time using $O(\log r)$ words. Let $\mathcal{R} = \{\theta(v_i) : v_i \text{ is a reflex vertex inside } \Delta(\mathcal{C})\}$ and let n, k be the number of vertices of $\mathcal{C}$ and the cardinality of $\mathcal{R}$ respectively. Recall that in steps 9 and 10 of Algorithm 2 we are choosing a random reflex vertex v inside $\Delta(\mathcal{C})$ and using it to split the chain $\mathcal{C}$ into two subchains $\mathcal{C}_1, \mathcal{C}_2$, such that the number of reflex vertices inside $\Delta(\mathcal{C}_1)$ and $\Delta(\mathcal{C}_2)$ is balanced in expectation. The next variant of Algorithm 2 replaces the random selection with a deterministic selection algorithm that guarantees a balanced split, albeit at a slight increase in the running time. The algorithm is based in the approximate median pair algorithm proposed by Raman and Ramnath [14], in which an approximation of the median of a set of m elements can be computed using $O(\log m)$ variables in $O(\log m)$ passes.

Theorem 3. *The visibility polygon of point q can be computed in $O(n \log^2 r)$ time using $O(\log r)$ variables.*

5 Closing Remarks

One expects that the running time of a constant workspace algorithm increases when compared to other models in which memory is not an issue. In most geometric problems, the increase in the running time is almost equal to the reduction

in memory space [2,3]. The algorithm given in Section 3 follows a similar pattern, since in the worst case the time-space product matches the one of Joe and Simpson [10] (from $O(n^2)$ to $O(n\bar{r})$). However, notice that the improvement of the algorithm given in Section 4 is much better, since the time-space product becomes $O(n \log n) \times O(\log r) = O(n \log n \log r)$.

If we compare the two methods given in this paper, we obtain a linear reduction in the running time by increasing the memory space by a logarithmic factor. This is due to the fact that this space allowed us to use more powerful techniques, such as divide and conquer. It would be interesting to study if the same result holds for other geometric problems (such as computing the Voronoi diagram, Delaunay triangulation or convex hull).

References

1. T. Asano, W. Mulzer, G. Rote, and Y. Wang. Constant-work-space algorithm for geometric problems. *Submitted to Journal of Computational Geometry*, 2010.
2. T. Asano, W. Mulzer, and Y. Wang. Constant-work-space algorithm for a shortest path in a simple polygon. In Md. S. Rahman and S. Fujita, editors, *WALCOM*, volume 5942 of *Lecture Notes in Computer Science*, pages 9–20. Springer, 2010.
3. T. Asano and G. Rote. Constant-working-space algorithms for geometric problems. In *CCCG*, pages 87–90, 2009.
4. P. Bose, P. Carmi, F. Hurtado, and P. Morin. A generalized Winternitz theorem. Journal of Geometry, In Press.
5. Timothy M. Chan. Comparison-based time-space lower bounds for selection. *ACM Trans. Algorithms*, 6:26:1–26:16, April 2010.
6. T.M. Chan and E.Y. Chen. Multi-pass geometric algorithms. *Discrete & Computational Geometry*, 37(1):79–102, 2007.
7. Greg N. Frederickson. Upper bounds for time-space trade-offs in sorting and selection. *J. Comput. Syst. Sci.*, 34(1):19–26, 1987.
8. S. Ghosh. *Visibility Algorithms in the Plane*. Cambridge University Press, New York, NY, USA, 2007.
9. M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD*, pages 58–66, 2001.
10. B. Joe and R. B. Simpson. Corrections to Lee's visibility polygon algorithm. *BIT Numerical Mathematics*, 27:458–473, 1987. 10.1007/BF01937271.
11. J.I. Munro and M. Paterson. Selection and sorting with limited storage. *Theor. Comput. Sci.*, 12:315–323, 1980.
12. J.I. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theor. Comput. Sci.*, 165:311–323, October 1996.
13. J. O'Rourke. Visibility. In *Handbook of Discrete and Computational Geometry*, chapter 28, pages 643–664. CRC Press, Inc., 2nd edition, 2004.
14. Venkatesh Raman and Sarnath Ramnath. Improved upper bounds for time-space tradeoffs for selection with limited storage. In *Proceedings of the 6th Scandinavian Workshop on Algorithm Theory*, SWAT '98, pages 131–142, London, UK, 1998. Springer-Verlag.
15. R. Seidel. Convex hull computations. In *Handbook of Discrete and Computational Geometry*, chapter 22, pages 495–512. CRC Press, Inc., 2nd edition, 2004.