Contents lists available at SciVerse ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

# Space/query-time tradeoff for computing the visibility polygon [☆]

Mostafa Nouri Baygi [a,b,*], Mohammad Ghodsi [a,b]

[a] Computer Engineering Department, Sharif University of Technology, Tehran, Iran
[b] IPM School of Computer Science, Tehran, Iran

ARTICLE INFO

*Article history:*
Received 9 July 2011
Accepted 10 October 2012
Available online xxxx
Communicated by S. Suri

*Keywords:*
Visibility polygon
Logarithmic query time
Space/query-time tradeoff

ABSTRACT

In this paper, we consider the problem of computing the visibility polygon (VP) of a query point $q$ (or VP($q$)) in a scene consisting of some obstacles with total complexity of $n$. We show that the combinatorial representation of VP($q$) can be computed in logarithmic time by preprocessing the scene in $O(n^4 \log n)$ time and using $O(n^4)$ space. We can also report the actual VP($q$) in additional $O(|\mathrm{VP}(q)|)$ time. As a main result of this paper, we will prove that we can have a tradeoff between the query time and the preprocessing time/space. In other words, we will show that using $O(m)$ space, we can obtain $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$ query time, where $m$ is a parameter satisfying $n^2 \leqslant m \leqslant n^4$. For example, when $m = n^3$, the query time of our algorithm is $O(\sqrt{n} \log \sqrt{n})$, that improves the query time of the only available algorithm with this memory usage (Zarei and Ghodsi, 2008 [26]), which degrades to $O(n \log n)$ in the worst case. An elegant feature of our algorithm that makes it useful in many applications is that it can determine the properties of the VP without actually computing it.

## 1. Introduction

Visibility plays an important role in some problems in robotics and computer graphics. In robotics, for example, the efficient exploration of an unknown environment requires computing the visibility polygon of the robot. In some computer graphics applications, also, it is important to identify the objects in a scene that are illuminated by a light source.

The visibility polygon (VP) of a point $q$, denoted by VP($q$), is defined as the set of points of the scene that are visible from $q$. In this paper, we consider the problem of computing the VP of a query point $q$, when the scene is a fixed polygonal domain with total complexity of $n$. In the preprocessing phase, we construct some data structures based on the scene, from which we compute VP($q$) for any given $q$, quickly.

### 1.1. Related work

For simple polygons, this problem has been thoroughly investigated. ElGindy and Avis [11] and Lee [17] showed that VP($q$) can be computed in $O(n)$ time without any preprocessing. Later, Guibas et al. [13] and Bose et al. [4] proposed algorithms that use $O(n^3)$ space, and report VP($q$) in $O(\log n + |\mathrm{VP}(q)|)$ time. The preprocessing time in the former algorithm is $O(n^3)$, while it is $O(n^3 \log n)$ in the latter. Later, Aronov et al. [1] reduced the preprocessing time and space to $O(n^2 \log n)$ and $O(n^2)$, respectively, at the cost of increasing the query time to $O(\log^2 n + |\mathrm{VP}(q)|)$.

[☆] A preliminary version of this paper appeared in *Proceedings of the Third International Workshop on Frontiers in Algorithmics*, volume 5598 of *Lecture Notes in Computer Science*, pages 120–131, Springer-Verlag, 2009.
[*] Corresponding author at: Computer Engineering Department, Sharif University of Technology, Tehran, Iran.
*E-mail address:* nourybay@ce.sharif.edu (M. Nouri Baygi).

0925-7721/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comgeo.2012.10.004

Computing the VP is more challenging when the scene is a polygonal domain with total complexity of $n$, consisting of a simple polygon with $h$ holes. Without preprocessing, VP($q$) can be computed in $O(n \log n)$ time using the algorithms of Asano [2], and Suri and O'Rourke [24]. This was later improved to $O(n + h \log h)$ by Heffernan and Mitchell [14].

Asano et al. [3] showed that with a preprocessing in $O(n^2)$ time and using $O(n^2)$ space, VP($q$) can be reported in $O(n)$ time. It is important to note that the query time is always $\Theta(n)$ even if $|\mathrm{VP}(q)| = o(n)$. Later, Vegter [25] gave the first output sensitive algorithm which reports VP($q$) in $O(|\mathrm{VP}(q)| \log(n/|\mathrm{VP}(q)|))$ time, by preprocessing the scene in $O(n^2 \log n)$ time using $O(n^2)$ space. Pocchiola and Vegter [20] then showed that if the boundary polygon of the scene and its holes are convex, VP($q$) can be reported in $O(|\mathrm{VP}(q)| \log n)$ time, using the visibility complex and after a preprocessing in $O(n \log n + E)$ time and $O(E)$ space, where $E$ is the number of edges in the visibility graph of the scene. The visibility graph of a polygon $P$ is a graph with a node for each vertex of $P$, and an edge for each pair of nodes, if the segment connecting these vertices completely lies in $P$.

Zarei and Ghodsi [26] proposed an algorithm that reports VP($q$) in $O((1 + \min(h, |\mathrm{VP}(q)|)) \log n + |\mathrm{VP}(q)|)$ query time. The preprocessing takes $O(n^3 \log n)$ time and $O(n^3)$ space. They achieved this complexity by adding some diagonals to the polygon to make it simple, and then used the algorithm by Bose et al. [4] for the resulting simple polygon.

Recently, Inkulu and Kapoor [16] used another approach to reduce the preprocessing time while achieving a query time close to that of [26]. Their method is based on partitioning the polygon into some simple polygons, called corridors, using the technique used before for computing the shortest paths in polygons with holes. After partitioning the polygon, they use Aronov et al.'s algorithm [1] or the ray shooting algorithm by Hershberger and Suri [15] to compute the VP in each visible simple polygon.

Depending on the base algorithm to compute the VP in each corridor, they obtained different preprocessing and query complexities. If the algorithm by Aronov et al. [1] is used, the preprocessing time and space are $O(n^2 \log n)$ and $O(n^2)$ respectively, and the query time is $O((1+\min(h, |\mathrm{VP}(q)|)) \log^2 n + h + |\mathrm{VP}(q)|)$. If the ray shooting algorithm by Hershberger and Suri [15] is used, the preprocessing time and space are $O(T + E + n \log n)$ and $O(\min(E, hn) + n)$ respectively, and the query time is $O(|\mathrm{VP}(q)| \log n + h)$. Here, $T$ is the time to triangulate the given polygon, which is $O(n + h \log^{1+\epsilon} h)$ for a small positive constant $\epsilon > 0$. They showed how the $O(h)$ additive factor can be removed from the query time if we spend additional $O((\min(E, hn))^2)$ space and $O(h(\min(E, hn))^2)$ preprocessing time.

### 1.2. Our results

In this paper, we first introduce an algorithm which can be used to report VP($q$) in a polygonal scene in optimal time of $O(\log n + |\mathrm{VP}(q)|)$, using $O(n^4 \log n)$ preprocessing time and $O(n^4)$ space. The algorithm can be used to report the ordered visible edges and vertices of the scene around a query point, or the size of VP($q$) in optimal time of $O(\log n)$. The solution of the algorithm can also be used for further preprocessing to solve many other problems faster than before (e.g., testing line segment intersection with the VP).

Because the space used in the preprocessing is high, we have modified our algorithm to obtain a tradeoff between the space and the query time. With these modifications, using $O(m)$ space and $O(m \log(\sqrt{m}/n))$ time in the preprocessing phase, where $n^2 \leqslant m \leqslant n^4$, we can find the combinatorial representation (to be described later) of VP($q$) in $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$ time. If we need to report VP($q$), additional $O(|\mathrm{VP}(q)|)$ time is required in the query time.

We can summarize our results as follows:

- An algorithm with logarithmic query time to compute the combinatorial representation of the visibility polygon of a query point in a polygon with holes.
- An algorithm with the space/query-time tradeoff, to compute the combinatorial representation of the visibility polygon of a query point in a polygon with holes.
- These algorithms can report the visibility polygon of a query point in additional $O(|\mathrm{VP}(q)|)$ time.
- These algorithms have the capability to report other properties of the VP, e.g. the number of edges of VP($q$) for a query point $q$, without computing the VP, so the computation time of the properties is independent of the size of the VP.
- These algorithms can be combined with other algorithms to efficiently answer more complicated queries, e.g., reporting segments intersecting the VP of a query point, without first computing the VP (Section 4.3).

### 1.3. Comparison with previous results

One of the problems with the algorithms mentioned in Section 1.1 for polygonal scenes, is their large query time in worst cases. For example when $h = \Theta(n)$ and $|\mathrm{VP}(q)| = \Theta(n)$, the query time of the algorithms of Pocchiola and Vegter [20], Zarei and Ghodsi [26], and Inkulu and Kapoor [16] increases to $\Theta(n \log n)$ which is the running time of the algorithm without any preprocessing. For Vegter's algorithm, we can also generate instances that use more time than the required time to report VP($q$). Another problem with these algorithms, as far as we know, is that they cannot efficiently report other properties of the VP, like its size, without first computing the VP. For example, in Asano et al. [3], one should first compute VP($q$) to be able to determine its size.

Each of the algorithms for computing the VP is preferred in special cases. For example, if the arrangement of the obstacles in the scene is in such a way that all the scene is approximately visible from the query point, that is if $|\mathrm{VP}(q)| = \Theta(n)$,

**Table 1**
The preprocessing and query time and space usages of different algorithms for computing the VP of a query point.

| Preprocessing time | Space | Query time | Reference |
|---|---|---|---|
| $O(n^2)$ | $O(n^2)$ | $O(n)$ | Asano et al. [3] (1986) |
| $O(n^2 \log n)$ | $O(n^2)$ | $O(\lvert \mathrm{VP}(q)\rvert \log(n/\lvert \mathrm{VP}(q)\rvert))$ | Vegter [25] (1990) |
| $O(n \log n + E)$ | $O(E)$ | $O(\lvert \mathrm{VP}(q)\rvert \log n)$ | Pocchiola and Vegter [20] (1996) |
| $O(n^3 \log n)$ | $O(n^3)$ | $O((1+\min(h, \lvert \mathrm{VP}(q)\rvert)) \log n + \lvert \mathrm{VP}(q)\rvert)$ | Zarei and Ghodsi [26] (2008) |
| $O(n^2 \log n)$ | $O(n^2)$ | $O((1+\min(h, \lvert \mathrm{VP}(q)\rvert)) \log^2 n + h + \lvert \mathrm{VP}(q)\rvert)$ | Inkulu and Kapoor [16] (2009) |
| $O(T + \lvert E\rvert + n \log n)$ | $O(\min(\lvert E\rvert, hn) + n)$ | $O(\lvert \mathrm{VP}(q)\rvert \log n + h)$ | Inkulu and Kapoor [16] (2009) |
| $O(m \log(\sqrt{m}/n))$ | $O(m)$ | $O(n^2 \log(\sqrt{m}/n)/\sqrt{m})$ | This paper* |

* In these relations, $m$ is an arbitrary parameter such that $n^2 \leqslant m \leqslant n^4$.

and we need to report VP($q$), Asano et al.'s [3] algorithm and ours with $m = n^2$ are the best choices, since the preprocessing/query time are optimal. But if $|\mathrm{VP}(q)|$ is much smaller than $n$, that is $|\mathrm{VP}(q)| = o(n)$, the best choice depends on the preprocessing time and space that we can afford. When we can spend at most quadratic space, we can select either Pocchiola and Vegter's [20] or Vegter's [25] algorithms. If the number of holes is smaller than $|\mathrm{VP}(q)|/\log n$, Inkulu and Kapoor's [16] could be a good option too.

In cases that we can use more, but at most $O(n^3)$ space, we need to consider the number of holes in the scene, as well. If $h = \Theta(n)$, nothing will be gained using Zarei and Ghodsi's [26] and Inkulu and Kapoor's [16] methods compared to other algorithms. But in cases where $h = o(n)$, it may worth to use these algorithms. Unless the number of holes is much larger than $|\mathrm{VP}(q)|$, Inkulu and Kapoor's [16] outperforms Zarei and Ghodsi's [26].

Our algorithm, compared to Zarei and Ghodsi's [26] and Inkulu and Kapoor's [16], guarantees the worst case query time, whatever the arrangement of obstacles in the scene is. For example, if we use $O(n^3)$ space, we achieve $O(\sqrt{n} \log \sqrt{n} + |\mathrm{VP}(q)|)$ query time which is much better in the worst cases. In Table 1, the preprocessing and query times and spaces of the algorithms for computing the visibility polygon of a query point are summarized.

Recently, Gudmundsson and Morin [12] studied two visibility related problems. The first problem, visibility testing, is to preprocess a set of segments $S$ and a segment $s \in S$, so that given a query point $q$, we can quickly determine if $s$ is visible from $q$. They showed how to preprocess $S$ using $O(k)$ space, to answer queries in $O(m_s n^\epsilon / k)$ time. Here, $m_s$ is the number of critical constraints (to be described in the next section), that intersect $s$, which is $\Theta(n^2)$ in the worst case, $k$ is an arbitrary parameter such that $m_s \leqslant k \leqslant m_s^2$, and $\epsilon > 0$ is an arbitrary parameter that affects the constant factors in the time and space complexities.

The second problem studied in [12] is to count the number of visible segments of $S$ from a query point $q$. They gave an approximation algorithm with $O(En^\epsilon/\sqrt{k})$ query time, using $O(kn^\epsilon)$ space, where $E$ is the number of edges in the visibility graph of the scene, which is $\Theta(n^2)$ in the worst case, and $k$ is an arbitrary parameter such that $E \leqslant k \leqslant E^2$. The approximation factor of the algorithm is 2.

It can easily be checked that our algorithm can be used for both of these problems, with factor $n^\epsilon$ removed from the complexities.

### 1.4. *Paper organization*

The rest of this paper is organized as follows. In Section 2, we show how to compute the VP for a query point inside a polygonal scene in logarithmic time using a preprocessing step. In Section 3, we obtain a tradeoff between space and query time for this problem. Section 4 describes some of the immediate applications of the new algorithms, and finally, Section 5 summarizes the paper.

## 2. Logarithmic query time for computing the VP

Let $P$ be a polygonal domain with $n$ vertices and $h$ holes. Also, let $S = \{s_1, \ldots, s_n\}$ be the set of all edges of $P$, which may intersect each other only at their end-points. We denote the complement of the set of holes inside $P$ as *free space*. Let $q$ be a query point in the free space. The goal is to efficiently compute VP($q$) using as little preprocessing time and space as possible.

The idea is to partition the free space into a set of disjoint regions, totally covering the space, such that for any point $p$ inside a region $r$, the angularly ordered sequence of the visible edges and vertices of the scene around $p$ is fixed. Given a point $p$, we call the ordered sequence as *combinatorial structure* of VP($p$) and denote it by $\mathcal{VP}(p)$.

By a simple investigation, it can be verified that if $p$ moves inside the free space of $P$, $\mathcal{VP}(p)$ changes only when $p$ moves across extensions of segments connecting any pairs of visible vertices in the scene. These extensions are called *critical constraints*. Formally, for any pair $p_i$ and $p_j$ of distinct and mutually visible vertices of $P$, a critical constraint $c_{ij}$ is the ray emanating from $p_i$ in the direction that points away from $p_j$ until it hits an edge of the scene. By definition, any pair of mutually visible vertices results in at most two critical constraints (see Fig. 1). When $p$ crosses $c_{ij}$, the visibility state of $p_j$ from $p$ changes, that is, if it was invisible, it becomes visible and vice versa. This event also adds an invisible edge to $\mathcal{VP}(p)$ or removes a visible edge from $\mathcal{VP}(p)$. The following lemma summarizes this observation, which is an adaptation of a lemma proved by Aronov et al. [1].

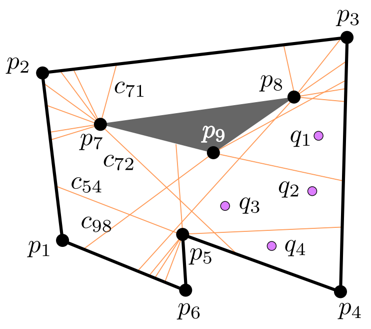

**Fig. 1.** Visibility decomposition for a polygonal domain. $q_1$, $q_2$ and $q_4$ have different combinatorial visibilities, while $q_2$ and $q_3$ have similar combinatorial visibilities.

**Lemma 2.1.** *Let $S = \{s_1, \dots, s_n\}$ be a set of segments in the plane and let $p$ and $q$ be two points not on any critical constraints of points of $P$. The ordered sequences of the visible edges and vertices of $P$ around $p$ and $q$, i.e., $\mathcal{VP}(p)$ and $\mathcal{VP}(q)$, differ iff they are on the opposite sides of a critical constraint, or a segment $s_i \in S$.*

Let $C$ denote the set of all segments in $S$ and all critical constraints $c_{ij}$, for $1 \leqslant i, j \leqslant n$, and $\mathcal{A}(C)$ denote the arrangement of lines, rays, and segments in $C$, i.e., the set of edges and vertices produced by the intersections of the critical constraints and the segments in $S$. The arrangement $\mathcal{A}(C)$ is called the *visibility decomposition* of $P$. Fig. 1 illustrates the visibility decomposition of a polygon with a hole. In the figure, the combinatorial structure of the visibility polygons of $q_2$ and $q_3$, denoted by $\mathcal{VP}(q_2)$ and $\mathcal{VP}(q_3)$ respectively, are equal, because both $q_2$ and $q_3$ see the following edges and vertices in the counterclockwise order: $(p_4p_3, p_3, p_3p_2, p_8, p_8p_9, p_9, p_9p_7, p_7, p_2p_1, p_1, p_1p_6, p_5, p_5p_4, p_4)$.

The complexity of the visibility decomposition is not greater than the complexity of the arrangement of $\binom{n}{2}$ lines, so $|\mathcal{A}(C)| = O(n^4)$. This implies the following corollary:

**Corollary 2.2.** *In a polygonal scene with complexity of $n$, the free space can be decomposed into $O(n^4)$ regions such that all points in a region have equal $\mathcal{VP}$'s. Moreover, for any two points $p$ and $q$ in two adjacent regions, $\mathcal{VP}(p)$ can be computed from $\mathcal{VP}(q)$ by $O(1)$ number of changes.*

Using the visibility decomposition, we obtain an approach for computing $\mathcal{VP}(q)$ for any query point. But before we proceed and to take advantage of the low change rate in $\mathcal{VP}$ between adjacent regions, we construct a tour of size $O(n^4)$ that visits all regions in the visibility decomposition. We build this tour using a depth-first traversal of the regions in the dual graph of $\mathcal{A}(C)$. Because this tour traverses each edge of $\mathcal{A}(C)$ at most twice, its size is $O(n^4)$. Based on this tour, we use a persistent data structure, such as a persistent red–black tree and store $\mathcal{VP}$ of all regions in this tree.

A persistent red–black tree, or PRBT, introduced by Sarnak and Tarjan [22], is a red–black tree capable of remembering its earlier versions, i.e., after $m$ updates into a set of $n$ linearly ordered items, any item of the version $t$ of the red–black tree, for $1 \leqslant t \leqslant m$ can be accessed in $O(\log n)$ time. In fact, after a sequence of $m$ updates, we will have $m$ roots of $m$ trees, each shares some nodes with other trees. PRBT can be constructed in $O((m+n)\log n)$ time using $O(m+n)$ space.

We now describe the preprocessing phase of our method. Initially, for an arbitrary region in the tour, $\mathcal{VP}$ is computed from scratch. Because $\mathcal{VP}$ is an ordered sequence, it can be stored in PRBT. We keep a pointer to the root of the current state of the tree as $\mathcal{VP}$ of the initial region. For the subsequent regions in the tour, we update PRBT accordingly by simply inserting or removing a point and/or an edge in the tree and save a pointer to the root of the current version of the tree as $\mathcal{VP}$ of the region. Finally, when all regions in $\mathcal{A}(C)$ are visited, we will have, for each region, a pointer to the root of a red–black tree that determines the order of the visible edges and vertices of $P$ from any point $p$ in that region, i.e., $\mathcal{VP}(p)$. Because the size of $\mathcal{VP}$ is always $O(n)$, the height of PRBT is logarithmic.

Constructing $\mathcal{A}(C)$ and the tour visiting all the regions takes $O(n^4)$ time. Computing $\mathcal{VP}$ for the first region needs $O(n\log n)$ time and $O(n)$ space, and the subsequent updates for $O(n^4)$ regions need $O(n^4\log n)$ time and $O(n^4)$ space. A point location data structure for $\mathcal{A}(C)$ is also needed that can be created in $O(n^4)$ time and space [5]. Totally, the preprocessing phase is completed in $O(n^4\log n)$ time using $O(n^4)$ space.

At the query time, we need to locate the region $r$ containing the query point $q$, and then find the precomputed $\mathcal{VP}$ for $r$. Locating $r$ is achieved using the point location data structure. Finally, we return the root of the tree associated with $r$, because the root specifies $\mathcal{VP}$ completely. Each of these tasks can be accomplished in $O(\log n)$ time.

If $\mathrm{VP}(q)$ (all vertices and edges of the VP) is to be computed, additional $O(|\mathrm{VP}(q)|)$ time is needed. This is done by examining the vertices and edges of $\mathcal{VP}(q)$ and finding the constructed edges and vertices of $\mathrm{VP}(q)$. The constructed edges (respectively, vertices) are edges (respectively, vertices) in VP that are not edges (respectively, vertices) of $P$ (see Fig. 2).

**Theorem 2.3.** *A planar scene consists of $n$ segments can be preprocessed in $O(n^4\log n)$ time and $O(n^4)$ space, such that for any query point $q$, a pointer to a red–black tree which stores $\mathcal{VP}(q)$ can be returned in $O(\log n)$ time. Furthermore,* $\mathrm{VP}(q)$ *can be reported in $O(\log n + |\mathrm{VP}(q)|)$ time.*

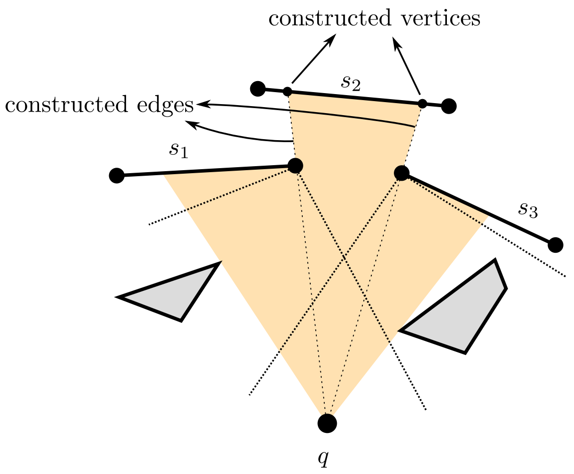

**Fig. 2.** Parts of the VP of a point. Colored region denotes VP($q$). The constructed edges and constructed vertices of the VP are marked. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

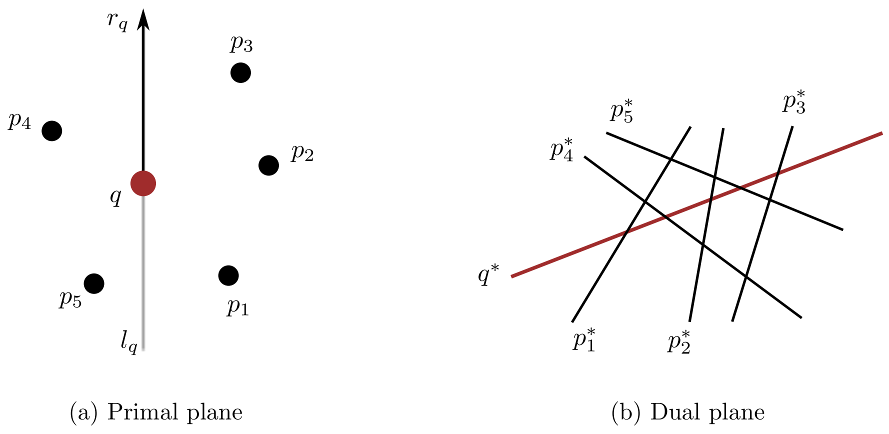

**Fig. 3.** (a) A set of points $P = \{p_1, p_2, p_3, p_4, p_5\}$ and a point $q$ in the primal plane. (b) The dual lines of points of $P$ and $q$.

It is easy to arrange a scene in such a way that the number of combinatorially different visibility polygons is $\Theta(n^4)$ (see e.g., [26]). Therefore, it seems likely that the above result is the best we can achieve for the optimal query time.

## 3. Space/query-time tradeoff

In Section 2, an algorithm with logarithmic query time for computing the VP is presented that uses $O(n^4)$ space and $O(n^4 \log n)$ preprocessing time. Asano et al. [3] have shown that in a polygonal scene VP($q$) can be computed in $O(n)$ time by preprocessing the scene in $O(n^2)$ time using $O(n^2)$ space. In this section, these results are combined and a tradeoff between the memory usage and the query time for computing the VP is obtained. Before describing the details of our method, we first review the technique used by Asano et al. [3] and then explain the construction of a regular cutting which is used to achieve the tradeoff.

### *3.1. Linear time algorithm for computing the VP*

To be able to use the technique used by Asano et al. [3], let us review their method with some modifications. Consider the duality transform that maps the point $p = (p_x, p_y)$ into the line $p^* : y = p_x x - p_y$ and the line $l : y = mx + b$ into the point $l^* = (m, -b)$. This transformation preserves incidence and order between points and lines [9], i.e., $p \in l$ iff $l^* \in p^*$ and $p$ lies above $l$ iff $l^*$ lies above $p^*$.

Let $P = \{p_1, \ldots, p_n\}$ be a polygonal domain with $n$ vertices as before. Let also $\mathcal{A}(P^*)$ denote the arrangement of the set of lines $P^* = \{p_1^*, \ldots, p_n^*\}$. For a query point $q$, the line $q^*$ can also be inserted in the arrangement in $O(n)$ time [7]. Let $r_q$ denote the vertical upward ray from $q$ and $l_q$ be its supporting line. Fig. 3(a) shows a sample set of points and Fig. 3(b) shows the dual lines of these points.

When $r_q$ rotates around $q$ counter-clockwise 180°, it touches all the half-plane to the left of $q$, and $l_q^*$ slides on the line $q^*$ from $-\infty$ to $+\infty$. Whenever $r_q$ reaches a point $p_i$, $l_q^*$ lies on $p_i^*$. Thus, the angular order around $q$ of all points $p_i$ in the left half-plane of the vertical line through $q$ is the same as the order of the intersection points between $q^*$ and the dual of those points according to $x$-coordinate. The same statement holds for the right half-plane of the vertical line through $q$.

Using the analogy between the orders of the points and the orders of the dual lines, we can compute the angular order of the points in $P$ around $q$ in linear time. Based on these orders, the visible edges and vertices from $q$ can easily be found in linear time.

**Lemma 3.1.** *(See Asano et al. [3].) Using $O(n^2)$ time and $O(n^2)$ space for preprocessing the scene to construct $\mathcal{A}(P^*)$ (by an algorithm due to Chazelle et al. [7]), for any query point $q$, we can find the angular sorted list of $p_i$, $1 \leqslant i \leqslant n$, in $O(n)$ time.*

## 3.2. Regular $(1/r)$-cutting

Before we describe the algorithm with the space/query-time tradeoff, we explain a concept usually used in algorithms with a divide-and-conquer approach. Let $L$ be a set of $n$ lines in the plane and $r \leqslant n$ be a given parameter. A $(1/r)$-cutting for $L$ is a collection of (possibly unbounded) triangles with disjoint interiors, which covers all the plane and the interior of each triangle intersects at most $n/r$ lines of $L$. The first (though not optimal) algorithm for constructing a cutting is given by Clarkson [8]. He showed that a random sample of size $r$ of the lines of $L$ can be used to produce an $O(\log r/r)$-cutting of size $O(r^2)$. Efficient construction of $(1/r)$-cuttings of optimal size $O(r^2)$ can be found in [6] and [10]. The set of triangles in the cutting together with the collection of lines intersecting each triangle can be found in $O(nr)$ time.

In our algorithm, we need a special type of the cutting, with this additional property: any arbitrary line intersects $O(r)$ triangles of the cutting. This property is needed to guarantee the worst case query time of our algorithm. We call this property the *regularity* of the cutting. In the following, we prove that the algorithm by Matoušek [18], has the regularity property.

Let us first review the algorithms by Chazelle and Friedman [6] and Matoušek [18]. Chazelle and Friedman [6] gave a randomized algorithm that computes a $(1/r)$-cutting in $O(nr)$ expected time. They further showed that using the method of Raghavan [21] and Spencer [23], they could make the algorithm deterministic, although the running time would be polynomial time. Matoušek [18] used the algorithm by Chazelle and Friedman [6] as the base, and by a recursive algorithm, computed a $(1/r)$-cutting in $O(nr)$ time.

The base algorithm used by Matoušek is as follows. For a set $L$ of $n$ lines, we want to find a $(1/r)$-cutting $\Xi$, with size $O(r^2)$. In the first step of the algorithm, we draw a sample $R \subset L$ by choosing each line in $L$ with probability $r/n$. We then construct the arrangement $\mathcal{A}(R)$ of lines of $R$ and the *canonical triangulation* of the arrangement. The canonical triangulation or *bottom-vertex triangulation* of an arrangement is obtained by drawing, for each cell $c$ of the arrangement of $\mathcal{A}(R)$, all the diagonals of $c$ incident to the lowest vertex of $c$.

Let $\mathcal{T}(R)$ denote the canonical triangulation of the arrangement of $R$. $\mathcal{T}(R)$ is too rough to guarantee a cutting factor of order $r$. But, we can refine this cutting by applying another level of partitioning. For each triangle $t \in \mathcal{T}(R)$, which is called a *first-generation triangle*, let $L_t$ denote the set of lines in $L$ intersecting $t$ and $r_t = |L_t|.r/n$, i.e., the factor by which $|L_t|$ exceeds the quantity $n/r$. For each triangle $t \in \mathcal{T}(R)$, we compute a $(1/r_t)$-cutting $\Xi_t$ for $L_t$, using any weaker method with $r_t^C$ sub-triangles, for some constant $C$. The union of all second-generation triangles is the resulting cutting $\Xi$. The following lemma gives a bound on the number of triangles in $\Xi$, which is proved in [18].

**Lemma 3.2.** *The expected number of triangles in $\Xi$ is $O(r^2)$.*

The intuition behind the above lemma is that the expected number of triangles with $r_t \geqslant k$ decreases exponentially with $k$.

Using the above cutting, we now attempt to find the expected number of triangles intersected by any arbitrary line. The following lemma shows that this value is linear.

**Lemma 3.3.** *For any line $l$, the expected number of triangles $t \in \Xi$ intersected by $l$ is $O(n)$.*

**Proof.** The expected number of lines of $R$ is $\Theta(r)$. By the zone theorem [7], the expected number of the first-generation triangles intersected by $l$ is also $\Theta(r)$. Let $\mathcal{T}_l(R)$ denote the set of triangles in $\mathcal{T}(R)$ intersected by $l$ and $\Xi_l$ denote the set of triangles in $\Xi$ intersected by $l$. We have

$$|\Xi_l| = \sum_{t \in \mathcal{T}_l(R)} r_t^C.$$

Taking expectation from both sides,

$$\begin{aligned}
\mathrm{E}\big(|\Xi_l|\big) &= \mathrm{E}\Bigg(\sum_{t \in \mathcal{T}_l(R)} r_t^C\Bigg) \\
&= \mathrm{E}\big(\big|\mathcal{T}_l(R)\big|\big)\mathrm{E}\big(r_t^C\big) \\
&= \mathrm{E}\big(\big|\mathcal{T}_l(R)\big|\big)\frac{\mathrm{E}(\sum_{t' \in \mathcal{T}(R)} r_{t'}^C)}{\mathrm{E}(|\mathcal{T}(R)|)}.
\end{aligned}$$

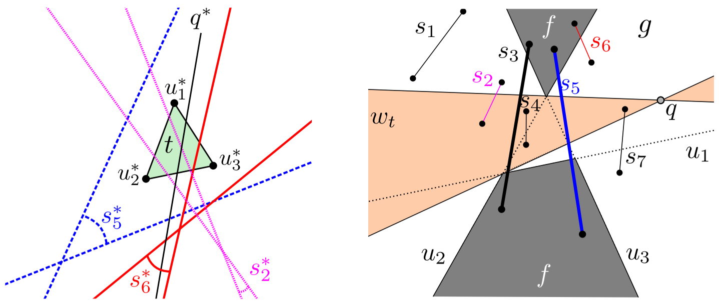

**Fig. 4.** (a) In the dual plane, the triangle $t$, is intersected by $q^*$. The dual of some of obstacle segments are also shown. (b) The arrangement of $f$, $g$, $q$ and segments of $S$ in the primal plane. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

Since $\mathrm{E}(\sum_{t'\in\mathcal{T}(R)} r_{t'}^C)$ is the expected number of triangles in $\Xi$, which is proved to be $O(r^2)$ by Matoušek, and $\mathrm{E}(|\mathcal{T}(R)|)$ is the expected number of the first generation triangles, which is $\Theta(r^2)$, we have

$$\begin{aligned}\mathrm{E}\big(|\Xi_l|\big) &= \mathrm{E}\big(\big|\mathcal{T}_l(R)\big|\big)\frac{\mathrm{E}(\sum_{t'\in\mathcal{T}(R)} r_{t'}^C)}{\mathrm{E}(|\mathcal{T}(R)|)}\\ &= O(r)\frac{O(r^2)}{\Theta(r^2)}\\ &= O(r),\end{aligned}$$

which proves the claim. □

Intuitively, since the expected number of the second-generation triangles inside each first-generation triangle is $O(1)$ and $l$ intersects $O(r)$ first-generation triangles, the expected number of the second-generation triangles is $O(r)$, that is $\mathrm{E}(|\Xi_l|) = O(r)$. Lemma 3.3 shows that the cutting produced by the algorithm of Chazelle and Friedman [6] has the regularity property. Since the algorithm is used as the base by Matoušek [18] to create the cutting, we can prove the regularity of his cutting, too. The following lemma states the existence of a regular cutting and is used in the next section to get the space/query-time tradeoff.

**Lemma 3.4.** *Given a set $L$ of $n$ lines, and a parameter $r \leqslant n$, we can compute a $(1/r)$-cutting for $L$ of size $O(r^2)$ and the set of lines intersecting each triangle of the cutting in $O(nr)$ time.*

### 3.3. Space/query-time tradeoff for computing the VP

We here show how a tradeoff between the space usage and the query-time for computing the VP can be achieved. As before, $P$ is a polygon with holes and $S$ is the set of edges of $P$. Let $P^*$ denote the set of dual lines of vertices of $P$ and $\mathcal{A}(P^*)$ denote the arrangement of lines of $P^*$. Based on the lines of $P^*$, we compute a regular $(1/r)$-cutting as described in Section 3.2 and denote it by $\mathcal{R}$. By definition, $\mathcal{R}$ decomposes the dual plane into $O(r^2)$ triangles such that each triangle is intersected by at most $O(n/r)$ lines of $P^*$, and the dual line of any arbitrary point intersects at most $O(r)$ triangles of the cutting.

Let $q^*$ be the dual line of a query point $q$ in the plane. The intersection points of $q^*$ with the edges of the cutting, break $q^*$ into $O(r)$ segments. These segments in the primal plane are $O(r)$ disjoint double wedges, centered at $q$, totally covering all the plane. Let $t$ be a triangle in the cutting intersected by $q^*$ (see Fig. 4(a)). Let $w_t$ denote the double wedge whose dual is the intersection of $q^*$ and $t$ in the dual plane (Fig. 4(b)). Let $\mathcal{VP}(q,t)$ denote $\mathcal{VP}(q)$ restricted to $w_t$. In order to compute $\mathcal{VP}(q)$, we can compute $\mathcal{VP}(q,t)$ for each triangle $t$ intersected by $q^*$ separately, and join them sequentially.

Let $u_1$, $u_2$ and $u_3$ be the three lines, whose dual points, $u_1^*$, $u_2^*$ and $u_3^*$, are the vertices of $t$, respectively. The arrangement of the lines, decomposes the plane into 7 regions, two of which are shaded in Fig. 4(b). Any point in the shaded regions is either above or below all $u_1$, $u_2$ and $u_3$. Let $f$ denote the union of the two shaded regions and $g$ denote the complement of $f$. The dual line of any point in $f$ does not intersect $t$, because it is either above or below all vertices of $t$. In contrast, the dual line of any point in $g$, which is lighter in the figure, intersects $t$. An immediate result is that $q$ lies in $g$. Moreover, $w_t$ is the double wedge, centered at $q$ with the two boundaries tangent to $f$ as shown in Fig. 4(b).

There are two sets of segments in $S$ whose dual double wedges intersect $t$. The first set contains segments with one or both end-points in $g$. We call these segments the *closed segments* of $t$, and denote the set by $CS_t$. The second set contains segments intersecting $g$ with both end-points in $f$, i.e., segments with one end-point above $u_1$, $u_2$ and $u_3$, and the other end-point below $u_1$, $u_2$ and $u_3$. We call these segments the *open segments* of $t$, and denote the set by $OS_t$.

Fig. 4(a) shows a triangle $t$ in the dual plane which is intersected by $q^*$. In Fig. 4(b), $f$, $g$, and a set of segments intersecting $g$ are shown. The region $g$ contains $q$ and at least one end-point of each closed segment (e.g., $s_2$, $s_6$). In contrast, $f$, which is shaded, contains at most one end-point of each closed segment and both end-points of the other segments of $S$. If an end-point of a segment lies in a shaded region and the other end-point lies in the other shaded region, that segment is an open segment (e.g., $s_5$). As can be seen in Fig. 4(a), the double wedges dual to the open segments completely surround the triangle, while the double wedges dual to the closed segments only intersect parts of the triangle.

By the above observations, and the fact that the segments of $S$ may intersect each other only at their end-points, we conclude that the segments in $OS_t$ play roles of infinite walls for $g$ and partition it into $|OS_t|+1$ subregions. These segments are drawn by thicker lines in Fig. 4(b). Let $R_t = \{r_t^1, \dots, r_t^{|OS_t|+1}\}$ denote the set of subregions. In each region $r_t^k$, a set of segments $CS_t^k \subset CS_t$ is contained. Since $\mathcal{VP}(q,t)$ is limited to the subregion $r_t^k$ in which $q$ lies, to compute $\mathcal{VP}(q,t)$, we first find $r_t^k$, and then compute $\mathcal{VP}(q)$ restricted to $r_t^k$ and the double wedge $w_t$.

If $q$ lies in $r_t^k \in R_t$, $q$ can only see a subset of segments in $CS_t^k$ and at most two walls of $OS_t$ separating $r_t^k$ from $r_t^{k-1}$ and $r_t^{k+1}$. If in the preprocessing phase, for $r_t^k$ we use the method of Section 2 with $CS_t^k$ as the set of obstacles, we can find $\mathcal{VP}(q,t)$ restricted to $w_t$ in $O(\log(|CS_t^k|)) = O(\log(n/r))$ time. We need to do the preprocessing with precaution here, because the size of $R_t$ for each triangle could be linear. To avoid the additional linear space needed in the preprocessing, which makes our method inefficient for $r = \omega(n^{\frac{3}{4}})$, we only preprocess the non-empty regions of $R_t$. If we do not preprocess the empty regions $r_t^k$, i.e., $CS_t^k = \varnothing$, the preprocessing time and the space will be $O((n/r)^4 \log(n/r))$ and $O((n/r)^4)$, respectively. Whenever a query point $q$ is received, we first find the region $r_t^k$ in which $q$ lies. If $CS_t^k \neq \varnothing$, we use the attached data structures to compute $\mathcal{VP}(q,t)$, otherwise $r_t^k$ consists only of at most two walls of $r_t^k$, $s_i, s_j \in OS_t$, and we can easily compute $\mathcal{VP}(q)$ restricted to $w_t$ in $O(1)$ time.

### *3.3.1. Details of data structures and query operation*

We need to compute three data structures, $CS_t$, $R_t$ and $CS_t^k$, to be able to compute $\mathcal{VP}(q)$. The first data structure, $CS_t$ is the set of segments with an end-point in $g$. Equivalently, $CS_t$ is the set of segments with an end-point whose dual intersects $t$. The set of lines intersecting each triangle of the cutting $\mathcal{R}$ was computed during the construction of $\mathcal{R}$. Therefore, we compute $CS_t$ of size $O(n/r)$, while constructing the cutting in $O(nr)$ time.

In contrast, computing $R_t$, the set of subregions for each triangle $t$, of size $O(n)$, takes much space and time and we cannot explicitly compute it for all the triangles of $\mathcal{R}$. We first compute $R_t$ for unbounded triangles in the cutting, and then for other triangles, we compute the differences between $R_t$ and $R_{t'}$ where $t'$ is a triangle adjacent to $t$.

The last data structure, $CS_t^k$, is the set of closed segments of subregion $r_t^k$ in $R_t$. We compute it by considering each end-point $p$ of each segment $s_i \in S$ and add $s_i$ to $CS_t^k$ for each region $r_t^k$ that $p$ lies in.

**Storing $R_t$:** We have two methods to store $R_t$, depending on the type of $t$. First, note that the segments of $OS_t$ and the regions in $R_t$ are totally ordered. We use this order to store the regions in a data structure.

For unbounded triangles in $\mathcal{R}$, we store a partially sorted list of regions in $R_t$. The list is maintained this way: we decompose $R_t$ into $O(n/r)$ collections $R_t(i)$, each of size $r$, such that any region in collection $R_t(i)$ is to the left of any region in collection $R_t(j)$, where $i < j$. The list can be created in $O(n\log(n/r))$ time using $O(n)$ space. Because the number of unbounded triangles is $O(r)$, we use $O(rn\log(n/r))$ time and $O(rn)$ space for all unbounded triangles. A search in a partially sorted list takes $O(r + \log(n/r))$ time.

For the other (bounded) triangles of $\mathcal{R}$, we cannot store $R_t$ explicitly, because of the limitations in the space and the preprocessing time. Therefore, for all (bounded or unbounded) pairs of adjacent triangles $t$ and $t'$, we store the differences of $R_t$ and $R_{t'}$, so that when moving from $t$ to $t'$, we are able to find the new position of a point in $R_{t'}$, given its previous position in $R_t$. For any two adjacent triangles $t$ and $t'$, $M_{t\to t'}$ is a binary search tree that stores new regions in $R_{t'}$ created by merging or splitting some regions in $R_t$ when moving from $t$ to $t'$.

Assume that we know the point $p$ lies in $r_t^k \in R_t$ and need to find the region $r_{t'}^l \in R_{t'}$ that contains $p$. To find $r_{t'}^l$, we search in $M_{t\to t'}$ for the region that contains $p$. If the search is successful, $r_{t'}^l$ is found, otherwise, we know that the region $r_t^k$ is left unchanged in $R_{t'}$ and the same region in $R_{t'}$ contains $p$.

For any pairs of adjacent triangles $t$ and $t'$, only closed segments of $t$ and $t'$ add regions to $M_{t\to t'}$. For each $s_i \in CS_t$ such that $s_i \in OS_{t'}$, $s_i$ is a wall in $R_{t'}$, while it is not a wall in $R_t$, therefore, a split is taken place, when moving from $t$ to $t'$. The region that is split is inserted into $M_{t'\to t}$ and two new regions are inserted into $M_{t\to t'}$. The same procedure should be performed for each segment $s_i \in CS_{t'}$ such that $s_i \in OS_t$ to complete the data structures. Since $|M_{t\to t'}|$ is not greater than $|CS_t| + |CS_{t'}| = O(n/r)$, the total complexity of the memory usage and time to construct all $M_{t\to t'}$ are $O(rn)$ and $O(rn\log(n/r))$ respectively. The needed time to find $r_{t'}^l$ from $r_t^k$ is also $O(\log(n/r))$.

**Storing $CS_t^k$:** Let $p$ be an end-point of a segment $s_i \in S$ and $t$ be a triangle intersected by $p^*$. We know that $s_i$ should be added to $CS_t^k$ of the region $r_t^k$ containing $p$. If we do this for all end-points of segments of $S$, $CS_t^k$ are computed for all $t \in \mathcal{R}$ and all $r_t^k \in R_t$.

For an end-point $p$ of a segment $s_i$, we start from an unbounded triangle, $t_1$, that is intersected by $p^*$. Using $R_{t_1}$, we can identify $r^k_{t_1}$ that contains $p$ in $O(r+\log(n/r))$ time and add $s_i$ to $CS^k_{t_1}$. Next, we move to a neighbour triangle of $t_1$, say $t_2$, that is also intersected by $p^*$. Using $M_{t_1 \to t_2}$, $r^l_{t_2}$ that contains $p$ is found in $O(\log(n/r))$ and consequently $s_i$ is added to $CS^l_{t_2}$.

The procedure is continued for all such triangles intersected by $p^*$ and it takes $O(r\log(n/r))$ time to insert $s_i$ into all $CS^k_t$ for all regions $r^k_t$ containing $p$. We repeat the method for both end-points of all segments of $S$, and when it is finished, $CS^k_t$ for all non-empty regions $r^k_t$ will be computed in $O(rn\log(n/r))$ time.

**Query processing:** In order to compute $\mathcal{VP}(q)$ for a query point $q$, for all triangles $t$ intersected by $q^*$, we find the region $r^k_t$ that contains $q$, in total $O(r\log(n/r))$ time, using $R_t$ and $M_{t'\to t}$. Inside each $r^k_t$, we compute $\mathcal{VP}(q,t)$ in $O(\log(n/r))$ time, using the data structures attached to $r^k_t$. We compute $\mathcal{VP}(q)$ by joining sequentially the bounded visibilities, $\mathcal{VP}(q,t)$, for each triangle $t$ intersected by $q^*$. Because the total number of triangles intersected by $q^*$ is $O(r)$, $\mathcal{VP}(q)$ is computed in $O(r\log(n/r))$ time.

In summary, the total preprocessing time and space are $O(n^4\log(n/r)/r^2)$ and $O(n^4/r^2)$, respectively, and the query time is $O(r\log(n/r))$. If the space complexity is denoted by $m$, we conclude the following theorem.

**Theorem 3.5.** *A planar polygonal scene with complexity n can be preprocessed into a data structure of size $O(m)$, for $n^2 \leqslant m \leqslant n^4$, in $O(m\log(\sqrt{m}/n))$ time, such that for any query point q, in $O(n^2\log(\sqrt{m}/n)/\sqrt{m})$ time $\mathcal{VP}(q)$ can be returned. Furthermore,* VP$(q)$ *can be reported in $O(n^2\log(\sqrt{m}/n)/\sqrt{m}+|\mathrm{VP}(q)|)$ time.*

## 4. Applications

In this section we use the previous results for computing the VP, and apply them to some other related problems.

### *4.1. Maintaining the VP of a moving point*

Let $q$ be a point in a polygonal scene. The problem is to update VP$(q)$ as it moves along a line. We can use the technique used to compute the VP of a query point, to maintain the VP of a moving point in the plane. In the first case, we create a data structure of size $O(n^4)$ in $O(n^4\log n)$ time to decompose the plane into a collection of cells with fixed $\mathcal{VP}$. For a point $q$, we can compute VP$(q)$ in $O(\log n + |\mathrm{VP}(q)|)$ time by first locating the (convex) region $r$ in which $q$ lies and then reporting VP$(q)$. Now assume that $q$ moves along a line $l$ in the plane. We use a binary search to detect in $O(\log n)$ time the edge of $r$ that is intersected by $l$ and then move $q$ to the adjacent cell $r'$ which shares that edge with $r$. When $q$ moves from $r$ to $r'$, only $O(1)$ number of changes are made in VP$(q)$ and they can be applied to VP$(q)$ in $O(\log n)$ time. Therefore, we conclude the following result.

**Theorem 4.1.** *A planar polygonal scene with complexity n can be preprocessed into a data structure of size $O(n^4)$, in $O(n^4\log n)$ time, such that for any query point q which moves along a line,* VP$(q)$ *can be maintained in $O(k\log n)$ time, where k is the number of combinatorial visibility changes in* VP$(q)$*. Furthermore, the place where the first combinatorial change in* VP$(q)$ *happens can be computed in $O(\log n)$ time.*

We can combine the above result with the technique of the space/query-time tradeoff, to obtain a tradeoff for this problem too. Consider in the dual plane, the arrangement of lines dual to the end-points of the segments and the corresponding $(1/r)$-cutting of the lines. The line $q^*$ intersects $O(r)$ triangles in the cutting and for each triangle, when $q$ moves and consequently $q^*$ rotates, the partial visibility polygon needs to be updated. We maintain a priority queue that stores the first places where the visibility of $q$ changes for each triangle. These places can be computed and inserted into this priority queue using Theorem 4.1 in $O(r(\log(r)+\log(n/r))) = O(r\log n)$ time. Using the queue, we find in $O(\log r)$ time the first place where the visibility of $q$ changes.

We should also consider the cases that $q^*$ leaves triangles, but these events are not important when the vertices of the cutting are the vertices of the original arrangement, which is the case in our cutting. Therefore, the first time that $\mathcal{VP}(q)$ changes can be detected and the queue can be updated in $O(\log(r)+\log(n/r)) = O(\log n)$ time. This observation leads to the following tradeoff for maintaining the VP of a moving point.

**Theorem 4.2.** *A planar polygonal scene with complexity n can be preprocessed into a data structure of size $O(m)$, $n^2 \leqslant m \leqslant n^4$, in $O(m\log(\sqrt{m}/n))$ time, such that for any query point q which moves along a line,* VP$(q)$ *can be maintained in $O(\frac{n^2}{\sqrt{m}}\log n + k\log n)$ time, where k is the number of combinatorial visibility changes in* VP$(q)$.

It is remarkable that the moving path need not to be a straight line; it can be a polygonal path, in which case, the number of break points is added to the number of combinatorial changes in the above upper bounds.

### 4.2. The weak visibility polygon of a query segment

The weak visibility polygon of a segment $rs$ is defined as the union of all the points in the plane, that are visible from at least one point on the segment. Using the previous result on maintaining the VP of a moving point, the weak visibility polygon of a query segment can also be computed easily.

Assume that the query point $q$ of the previous section is on $r$, the left end-point of $rs$, and moves to the other end-point. Computing $\mathrm{VP}(r)$ needs $O(n^2\log(\sqrt{m}/n)/\sqrt{m}+|\mathrm{VP}(r)|)$ time and each change in $\mathrm{VP}(q)$ can subsequently be detected and applied in $O(\log n)$ time. Constructing the priority queue that gives the triangle with the first change in the visibility takes $O(n^2\log(n^2/\sqrt{m})/\sqrt{m})$ time.

We should only care about those changes that increase the VP of $rs$. When the motion path of $q$ crosses a critical constraint of the scene, the visibility increases in only one direction, and in the other direction the visibility decreases. As $q$ in this problem moves from left to right (assuming $rs$ is not vertical), only those critical constraints should be considered that increase the visibility when they are crossed from left to right. This way, the number of critical constraints is reduced and only the useful events are processed during the movement of $q$.

Whenever $q$ encounters a critical constraint, an increase event in the visibility occurred and the new visible edges and/or vertices are added to the visible set. This result can be summarized in the following theorem:

**Theorem 4.3.** *A planar polygonal scene with complexity n can be preprocessed into a data structure of size $O(m)$, $n^2\leqslant m\leqslant n^4$, in $O(m\log(\sqrt{m}/n))$ time, such that for any query segment rs, $\mathrm{VP}(rs)$ can be computed in $O(\frac{n^2}{\sqrt{m}}\log n+|\mathrm{VP}(rs)|\log n)$ time.*

### 4.3. Weak visibility detection between two query objects

In [19], Nouri et al. studied the problem of detecting the weak visibility between two query objects in a polygonal scene. Formally, a polygonal scene with total complexity $n$, should be preprocessed, such that given any two query objects, it can be determined if the two objects are weakly visible from each other. They proved that using $O(n^2)$ space and $O(n^{2+\epsilon})$ preprocessing time, for any $\epsilon>0$, the queries can be answered in $O(n^{1+\epsilon})$ time.

They mentioned that the bottleneck of the algorithm is the time needed to compute the VP of a query point, and if it can be answered in a time less than $\Theta(n)$ in the worst case, the total query time will be reduced. Here, we summarize the result of applying the new technique for computing $\mathrm{VP}(q)$ and defer the details to the full version of that paper.

**Theorem 4.4.** *A planar polygonal scene with complexity n can be preprocessed in $O(m^{1+\epsilon})$ time to build a data structure of size $O(m)$, where $n^2\leqslant m\leqslant n^4$, so that the weak visibility between two query line segments can be determined in $O(n^{2+\epsilon}/\sqrt{m})$ time.*

## 5. Conclusion

In this paper, we studied the problem of computing the VP. We presented a logarithmic query time algorithm, where we can use $O(n^4\log n)$ time and $O(n^4)$ space in the preprocessing phase. As the algorithm requires much space, the algorithm was modified to get a tradeoff between the space usage and the query time. With this tradeoff, the VP of a query point can be computed in $O(n^2\log(\sqrt{m}/n)/\sqrt{m})$ time using $O(m)$ space, where $n^2\leqslant m\leqslant n^4$.

This result may have many applications in other related problems. An interesting future work is to find more applications for the proposed algorithms. It is also an interesting open problem, whether our algorithms are optimal in terms of space usage and preprocessing time.

## References

[1] B. Aronov, L.J. Guibas, M. Teichmann, L. Zhang, Visibility queries and maintenance in simple polygons, Discrete & Computational Geometry 27 (4) (2002) 461–483.
[2] T. Asano, An efficient algorithm for finding the visibility polygon for a polygonal region with holes, Transactions of IECE of Japan E 68 (9) (1985) 557–559.
[3] T. Asano, T. Asano, L.J. Guibas, J. Hershberger, H. Imai, Visibility of disjoint polygons, Algorithmica 1 (1) (1986) 49–63.
[4] P. Bose, A. Lubiw, J.I. Munro, Efficient visibility queries in simple polygons, Computational Geometry: Theory & Applications 23 (3) (2002) 313–335.
[5] B. Chazelle, Cutting hyperplanes for divide-and-conquer, Discrete & Computational Geometry 9 (1993) 145–158.
[6] B. Chazelle, J. Friedman, A deterministic view of random sampling and its use in geometry, Combinatorica 10 (3) (1990) 229–249.
[7] B. Chazelle, L.J. Guibas, D.T. Lee, The power of geometric duality, BIT 25 (1) (1985) 76–90.
[8] K.L. Clarkson, New applications of random sampling in computational geometry, Discrete & Computational Geometry 2 (2) (1987) 195–222.
[9] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd edition, Springer-Verlag, 2008.
[10] M. de Berg, O. Schwarzkopf, Cuttings and applications, International Journal of Computational Geometry & Applications 5 (4) (1995) 343–355.
[11] H.A. ElGindy, D. Avis, A linear algorithm for computing the visibility polygon from a point, Journal of Algorithms 2 (2) (1981) 186–197.
[12] J. Gudmundsson, P. Morin, Planar visibility: Testing and counting, in: Proceedings of the 26th ACM Symposium on Computational Geometry, 2010, pp. 77–86.
[13] L.J. Guibas, R. Motwani, P. Raghavan, The robot localization problem, SIAM Journal on Computing 26 (4) (1997) 1120–1138.
[14] P.J. Heffernan, J.S.B. Mitchell, An optimal algorithm for computing visibility in the plane, SIAM Journal on Computing 24 (1) (1995) 184–201.
[15] J. Hershberger, S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, Journal of Algorithms 18 (May 1995) 403–431.

[16] R. Inkulu, S. Kapoor, Visibility queries in a polygonal region, Computational Geometry: Theory & Applications 42 (9) (2009) 852–864.
[17] D.T. Lee, Visibility of a simple polygon, Computer Vision, Graphics, and Image Processing 22 (2) (1983) 207–221.
[18] J. Matoušek, Cutting hyperplane arrangements, Discrete & Computational Geometry 6 (1991) 385–406.
[19] M. Nouri, A. Zarei, M. Ghodsi, Weak visibility of two objects in planar polygonal scenes, in: Proceedings of the 2007 International Conference on Computational Science and Its Applications – Part I, 2007, pp. 68–81.
[20] M. Pocchiola, G. Vegter, The visibility complex, International Journal of Computational Geometry & Applications 6 (3) (1996) 279–308.
[21] P. Raghavan, Probabilistic construction of deterministic algorithms: Approximating packing integer programs, Journal of Computer and System Sciences 37 (2) (1988) 130–143.
[22] N. Sarnak, R.E. Tarjan, Planar point location using persistent search trees, Communications of ACM 29 (7) (1986) 669–679.
[23] J. Spencer, Ten Lectures on the Probabilistic Method, SIAM, 1987.
[24] S. Suri, J. O'Rourke, Worst-case optimal algorithms for constructing visibility polygons with holes, in: Proceedings of the 2nd Annual Symposium on Computational Geometry, 1986, pp. 14–23.
[25] G. Vegter, The visibility diagram: A data structure for visibility problems and motion planning, in: Proceedings of the 2nd Scandinavian Workshop on Algorithm Theory, 1990, pp. 97–110.
[26] A. Zarei, M. Ghodsi, Query point visibility computation in polygons with holes, Computational Geometry: Theory & Applications 39 (2) (2008) 78–90.