Computational Geometry Algorithms for Visibility Problems

by

Yeganeh Bahoo

A Thesis submitted to the Faculty of Graduate Studies of
The University of Manitoba
in partial fulfillment of the requirements of the degree of

Doctor of Philosophy (Ph.D.)

Department of Computer Science
University of Manitoba
Winnipeg
September 2019

Thesis advisor

Stephan Durocher, Prosenjit Bose

Author

Yeganeh Bahoo

Computational Geometry Algorithms for Visibility Problems

Abstract

This thesis examines algorithmic problems involving k-crossing visibility. Given two points p and q and a set of polygonal obstacles in the plane, where p and q are in general position relative to the vertices of the obstacles, p and q are *k-crossing visible* to each other if and only if the line segment pq intersects obstacle boundaries in at most k points. Given a simple polygon P and a query point q, we show that region of P that is k-crossing visible from q can be calculated in $O(kn)$ time, where n denotes number of vertices in P. With preprocessing of the polygon P, and using a data structure of size $O(n^5)$, the k-crossing visible region for any query point q can be reported in $O(\log n+m)$ time, where m is the size of the output and q is given at query time. When there is a constraint on the amount of memory available, the k-crossing visible region of q in P can be determined in $O(cn/s+n\log s+\min\{\lceil k/s\rceil n, n\log\log_s n\})$ time, where s denotes the number of words available in the limited workspace model. Finally, given an x-monotone polygonal chain, i.e., a *terrain*, we present an $O(n^4\log n)$-time algorithm to determine the minimum height of a *watchtower* point, located above the terrain, such that any point on the terrain is k-crossing visible from that point. Additionally, we propose an $O(n^3)$-time algorithm for the discrete version of the problem, in which the watchtower is restricted to being positioned over vertices of T.

Contents

List of Figures

Acknowledgments

First and foremost, I must thank my advisors, Dr. Stephane Durocher, and Dr. Prosenjit Bose. This long path was not possible without their continuous help, support, and guidance.

I appreciate all the useful feedback I received on my thesis from Dr. Avery Miller, Dr. Stephen Kirkland, and Dr. William Evans. I must also thank Dr. Thomas Shermer, Dr. Wolfgang Mulzer and Dr. Barahreh Banyassady who were my collaborators and I learned a lot from them.

Last but not least, I would like to thank my parents and my brother who always support and motivate me in my journey. I would like to especially thank my partner Sasa Janjic who was always there for me, including giving feedback on my thesis.

My parents, my brother and my significant other, Sasa Janjic.

Chapter 1

Introduction

Wireless technologies have evolved to meet our needs, from transferring morse coded messages to transmitting images of the vistas of distant worlds. We can engage in real-time communication with friends and family across the world. As a result of the progress and success of the field of wireless communications, a new area of research has developed to address problems related to range, throughput, and reliability, called *k-crossing visibility* or *k-visibility*. The two parameters that most strongly affect the ability for two wireless devices to communicate successfully are the distance between the devices and the number of walls (or obstacles) that exists between these devices. As technological progress in this field is fast, transmission range has increased rapidly, allowing better connectivity across longer distances. The latter parameter is, however, still a constraint, as passing through barriers reduces the energy of the wireless signal; after a wireless signal passes some number of walls, it is impossible for a device to receive a wireless transmission. This area of research and related problems studying geometric properties and algorithms related to wire-

less communication across walls or barriers is referred to as *k-crossing visibility* or *k-visibility*. Informally, two points p and q in a polygon P are said to be k-crossing visible when the line segment pq intersects the interior of P in at most k times. A formal definition is provided in Chapter 2.

In the field of Computational Geometry, k-crossing visibility was initially studied by Mouawad and Shermer [81] and also Dean and Lingas [49] for the special case where k is 1. As wireless network access became more commonplace, the need for efficient algorithms for wireless communication was heightened as global adoption and reliance on the technologies increased [67]. At this point, the k-crossing visibility for the case where k can be any arbitrary positive integer has become a topic of interest [57; 13; 50; 29; 3]. This thesis studies some problems in this area.

The first problem studied is to calculate the region in which a wireless device can communicate in the presence of obstacles, when a given building is modeled as a two-dimensional polygon. This problem is studied under two different settings, with and without a constraint on memory, with corresponding results presented in Chapters 6 and 4, respectively. In Chapter 5, we also study the problem when the building is given as a fixed input, but the position of wireless device changes in each run of the algorithm. Each time the algorithm runs, for a given device location, the requested output is the region k-crossing visible for this new location. Last, in Chapter 7, algorithms are proposed to calculate the minimum height of towers located along a road; a road travels along a straight line horizontally while moving up and down vertically.

The application of this problem is in the case that some towers must be located along a road so that mobile devices are connected to at least one tower at all times.

We begin by providing necessary definitions, followed by a summary of related works. The subsequent chapters investigate the problems of visibility query without preprocessing, visibility query with preprocessing, visibility query with constrained memory, and selecting positions for watchtowers, respectively. Chapters 4 and 6 represent algorithms that compute a region as a function of the input, Chapter 5 proposes a data structure with an associated query algorithm, and Chapter 7 investigates a geometric optimization problem, all of which relate to k-crossing visibility. Finally, we conclude by going over some interesting open problems.

Some of the results presented in this thesis are published or submitted to the following conferences and journals:

1. Finding the k-visibility region of a point in a simple polygon in the memory-constrained model. In Proc. 32nd European Workshop Comput. Geom.(EWCG), 2016.

2. Time-space trade-off for finding the k-visibility region of a point in a polygon. In WALCOM: Algorithms and Computation: 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29-31, 2017, Proceedings, volume 10167, page 308. Springer, 2017.

3. A time-space trade-off for computing the k-visibility region of a point in a polygon. Theoretical Computer Science, 2018.

4. Computing the k-Visibility Region of a Point in a Polygon. To appear in IWOCA: 30th International Workshop on Combinatorial Algorithms, IWOCA 2019, Pisa, 23-25 July, 2019.

5. Watchtower for k-crossing Visibility. To appear in CCCG: 31st Canadian Conference in Computational Geometry, CCCG 2019, Edmonton, August 8-10, 2019.

Chapter 2

Background and Definitions

In this section, we give precise definitions for the k-visibility problems examined in this thesis.

A simple polygon (P) is defined as a sequence of points $v_0, v_1, ..., v_{n-1}, v_0$ in the plane, and a corresponding sequence of line segments or edges v_0v_1, $v_1v_2, ..., v_{n-2}$ $v_{n-1}, v_{n-1}v_0$, where non-consecutive edges in the sequence do not intersect. By this definition, a simple polygon P is a closed Jordan curve. A simple closed Jordan curve is a curve that does not intersect itself and divides the plane into three disjoint regions: the set of points inside P (the interior), the set of points outside P (the exterior), and the set of points on P (the boundary). ∂P denotes the set of points on the boundary of P. In this thesis P refers to the interior and boundary as the polygon. A polygon with holes is a simple polygon that excludes a given set of simple polygons that lie inside it.

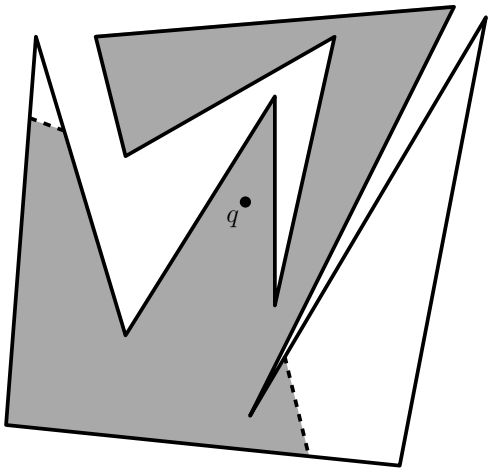


Figure 2.1: This figure illustrates one polygonal obstacle P in the plane. The shaded region is the region of P that is 2-visible from the query point q. The dotted line denotes the part of this boundary which is not an edge of the polygon.

2.1 Visibility

2.1.1 Visibility

Given a simple polygon P, two points p and q are said to be visible to each other if and only if the line segment pq does not intersect the exterior of P. If p and q are mutually visible and the line segment pq lies inside P (possibly overlapping the boundary of P), then they are said to be *internally visible* to each other. In contrast, if the line segment pq lies outside P, then they are said to be *externally visible*. Given a simple polygon P and a query point q inside P, the part of P visible from q is called the *visibility region* or *visibility polygon* of q. Visibility between p and q can also be considered among more general obstacles in the plane, such as a set of line segments in the plane. Various algorithmic problems have been studied related to visibility, such as those defined in Section 2.1.2.

2.1.2 k-Visibility

The geometric concept of visibility has been generalized in a variety of ways. The generalized visibility definition in this study is k-crossing visibility. First we define k-crossing visibility formally. Two paths P and Q are *disjoint* if $P \cap Q = \varnothing$. To provide a general definition of visibility requires a robust definition for a crossing between a line segment and a polygon boundary, in particular, for the case when points are not in general position.

Definition 1 (Weakly disjoint paths [Chang *et al.* (2014)[36]])**.** *Two paths P and Q are* weakly disjoint *if, for all sufficiently small $\epsilon > 0$, there are disjoint paths $\tilde{P}$ and $\tilde{Q}$ such that $d_{\mathcal{F}}(P, \tilde{P}) < \epsilon$ and $d_{\mathcal{F}}(Q, \tilde{Q}) < \epsilon$.*

$d_{\mathcal{F}}(A, B)$ denotes the Fréchet distance between A and B.

Definition 2 (Crossing paths [Chang *et al.* (2014)[36]])**.** *Two paths* cross *if they are not weakly disjoint.*

Definitions 1 and 2 apply when P and Q are Jordan arcs. We use Definition 2 to help to define k-crossing visibility.

Definition 3 (k-crossing visibility)**.** *Two Jordan arcs (or polygonal chains) P and Q* cross *k* times*, if there exist partitions $P_1, \ldots, P_k$ of P and $Q_1, \ldots, Q_k$ of Q such that P_i and Q_i cross, for all $i \in \{1, \ldots, k\}$. Points p and q in a simple polygon P are* k-crossing visible *if the line segment pq and the boundary of P do not cross $k+1$ times.*

Consequently, when p and q are in general position relative to the vertices of the obstacles (p, q and a vertex of the obstacles do not lie on the same line), p and q are k-crossing visible to each other if and only if the line segment pq crosses obstacle boundaries in at most k points. Figure 2.1 shows the 2-crossing visible region of the polygon for the given query point. When $k = 0$, k-crossing visibility and visibility are equivalent. Given a simple polygon P and a query point q, the k-crossing visibility region of q is the parts of P which is k-crossing visible from q; the k-crossing visible part of the plane is called the k-visibility polygon. Notice that when $k = 0$ and $q \in P$, the k-crossing visibility polygon and the k-crossing visibility region are the same.

Problem 1: Visibility Query without Preprocessing

Input: A simple polygon P, a point q in P, and a number k

Problem: Finding the k-visibility region of the plane for the point q.

We study visibility query without preprocessing in Chapter 4.

Problem 2: Visibility Query with Preprocessing

Input: A fixed polygon P, a query point q in P, and a fixed number k

Problem: Preprocessing P for the given integer k to construct a data structure to support efficient visibility queries where, for an arbitrary query point q, the k-visibility region of q in P must be returned.

Visibility query with preprocessing is studied in Chapter 5.

A polygon P is said to be *star-shaped* if and only if there exists a point p from

which the entire polygon P is visible. The polygon P is said to be *k-star-shaped* if there exists a point p from which the entire polygon P is k-crossing visible. The set of all such points is called the *k-kernel* of the polygon P. The 0-kernel is usually referred to as the *kernel*.

2.1.3 Constrained Memory

In real world settings, computers can be limited by the amount of memory available for computation. As a result, designing algorithms which can solve different problems with with a limited memory is of importance; such algorithms are called *memory-constrained algorithms*. The model in which researchers study these algorithms is described as follows: the input memory consists of a set of read-only words. The workspace that the algorithm uses for processing is a set of read-write words which generally can store $O(\log n)$ bits of information. The output is a set of write-only cells. The goal is to minimize the running time of algorithms while operating within the available memory.

Problem 3: Visibility Query with Constrained Memory

Input: A polygon P, a query point q in P, and a number k

Problem: Designing an algorithm to report the k-visibility polygon for q when constant or constrained memory space is available.

We study visibility query with constant and constrained memory space in Chapter 6.

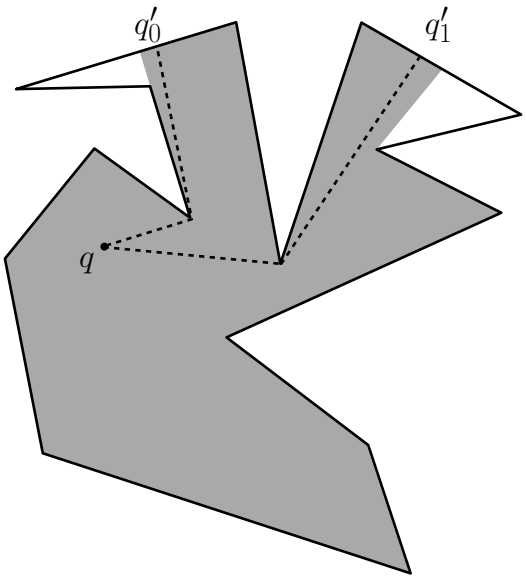


Figure 2.2: The gray region denotes the 2-link visible region of the polygon for a query point q. q'_0 and q'_1 are 2-link visible for q as the dotted lines show.

2.2 Guarding

2.2.1 Art Gallery

A set of points W in the polygon P is said to *guard* P if and only if every point in P is visible to at least one point in W. Each point in W is called a *guard*. The Art Gallery problem, introduced by Klee [93], seeks to identify a minimum cardinality set of points that guards a given polygon P. If each point in P must be visible from at least β guards, the problem is called β-*guarding* and P is called β-*guarded* [1]. When $\beta = 1$, Art Gallery and β-guarding problems are equivalent. In these problems the set of guards may be static or mobile. If static guards are located on vertices of P, they are referred to as *vertex guards*. A *mobile guard*, can patrol on an edge of the

[1]This problem is often called K-guarding in the literature. To avoid confusion, it is called β-guarding in this document.

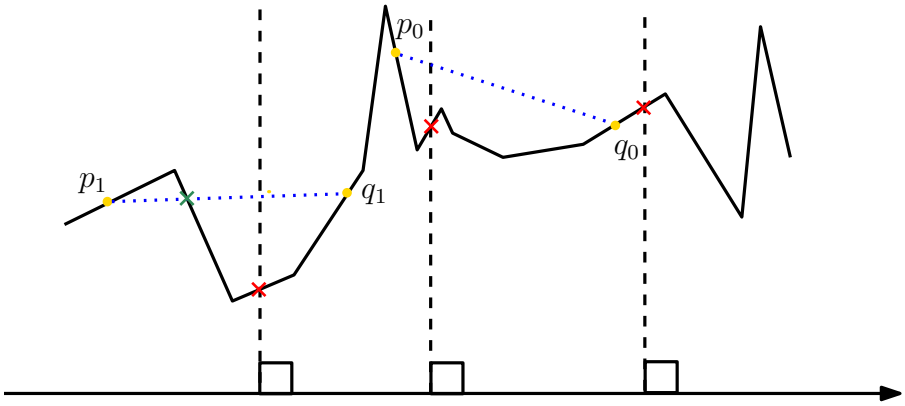


Figure 2.3: An x-monotone polygonal chain (terrain). p_0 and q_0 can see each other, while p_1 and q_1 can not.

polygon (*edge guard*), on a chord of the polygon (*chord guard*), or along a poly-line inside P (*poly-line guard*) [93]. Different kinds of visibility can be defined for the guard: the guard may see the entire region around itself, have a limited field of view such as α-radian (α-visibility), or have k-crossing visibility.

2.2.2 Guarding on the Terrain and Watchtower Problem

A terrain T is a 2-dimensional x-monotone polygonal chain, consisting of a set of points v_0, v_1,...,v_n, where the x-coordinate of $v_i < v_j$ for $i < j$; see Figure 2.3. In this thesis T always denotes an x-monotone polygonal chain with n vertices. A point p is said to lie below T if a vertical line through p intersects T at a point above p. Two points p and q on T are visible if and only if the line segment pq does not intersect the region below T; this is the usual definition of visibility, with the added constraint that the region below the terrain is opaque.

The Art Gallery problem for terrains is to find the minimum number of guards (W) needed on T such that each point on T is visible by at least one guard in W. If W is located above T, the problem is called the Watchtower problem. A *watchtower* is a point located above the terrain. The goal is to determine the minimum distance of one or a set of watchtowers above T from which the entire terrain T is visible. Watchtowers above a given terrain T can be either discrete, or continuous; in the discrete setting, the watchtowers' positions are restricted to points on the vertical lines emanating from vertices of P, while in the continuous version, watchtowers can be located at any point above T. Similar to the Art Gallery and β-guarding problems in polygons, many varieties of visibility can be studied for the Watchtower problem on terrains including k-crossing visibility.

Problem 4: Watchtower

Input: A terrain T, and a number k

Problem: Finding a watchtower with minimum height such that the entire terrain T is k-visible from the watchtower.

The algorithm for watchtower is proposed in Chapter 7.

Chapter 3

Related Work

Visibility problems have been of interest in Computational Geometry over the past few decades [86; 60]. In this section, we discuss previous results on problems related to visibility and the Art Gallery problem.

3.1 Visibility

3.1.1 Visibility

Given a polygon P with n vertices and a query point q inside P, a fundamental problem is to compute the visibility polygon for q: the region of P visible from q. This problem was first introduced by Davis and Benedikt [47], who gave an $O(n^2)$-time algorithm. However the number of vertices of the visibility polygon of q in P is proportional to the number of vertices of P in the worst case, i.e., $\Theta(n)$ [53; 75]. El Gindy and Avis proposed the first approach for finding the visibility polygon in $O(n)$ time without preprocessing [53], followed by Lee's work [75]. However, the approach

was in error as it was shown not to be applicable to all polygons, and a correction was provided by Joe and Simpson [69].

Given a polygon with n vertices, by using $O(n^3)$ space for preprocessing the polygon P, the visibility query can be answered more efficiently in time $O(\log n + m)$, where m refers to the number of vertices in the output polygon (the size of the output) [24]. Aronov *et al.* [6] introduced a new approach to answer the visibility query in $O(\log^2 n + m)$ time by using a $O(n^2)$-space data structure, constructed in the preprocessing step in $O(n^2 \log n)$ time.

3.1.2 k-Visibility

In recent years, research focus has turned to the application of k-crossing visibility in wireless networks. The concept of k-crossing visibility was first introduced by Dean *et al.* [49]. In [49], pseudo-star-shaped polygons in which each point was visible through one edge were studied, corresponding to k-crossing visibility where k is 1. Later, Mouad and Shermer [81] also studied the concept of k-crossing visibility, in what they originally called the *Superman problem*. Given a simple polygon P and a sub-polygon Q, the goal in this problem is to determine the minimum number of edges which must be made opaque so that the given point q located in the exterior of P cannot see any point in Q. Recently, the concept of k-crossing visibility was explored for arbitrary k, where an algorithm to construct the k-visible region from the query point q in $O(n^2)$ time was presented [12].

Other visibility problems that have been studied involve star-shaped and k-star-shaped polygons [79; 91; 55]. Evans and Sember [55] show how to calculate the k-kernel of a given polygon where the kernel of the polygon may lie outside of the polygon. Furthermore, the authors showed that the number of vertices of the k-kernel can be $O(n^4)$.

3.1.3 Constrained Memory

Given the proliferation of small mobile devices, a branch of algorithm design focuses on the development of algorithms for memory-constrained systems [82]. These algorithms are based around a variety of memory models and usage constraints. There are several notable variants of memory-constrained algorithms. *In-place algorithms* are the most basic. In this model, there is no limitation on the number of times the input can be read while there exist constraints on the available workspace [26; 27; 33; 34]. If a problem can be solved in $O(\log n)$ bits of workspace, it belongs to a class of problems called *LOGSPACE* [7]. Additionally, there are *streaming models* which are bounded by the number of times the input can be read, in addition to having a constraint on the workspace memory [84; 30; 14]. For solving problems for these different models, *succinct data structures* can be used to minimize the number of bits of space needed to store and represent the input [63; 85]. There have been a variety of problems in computational geometry which have been studied in this class of problems, such as computing the visibility region in a simple

polygon, among others. Perhaps the most well-studied problem in this area is sorting.

The optimal solution for reporting the 0-crossing visibility polygon from a given query point takes $O(n)$ time and $O(n)$ space [69] where n refers to the number of vertices of the polygon. Considering the in-place model, when the size of the workspace is $O(1)$, there exists an algorithm that needs $O(n\bar{r})$ time, where $\bar{r}$ refers to the number reflex vertices which are critical for the point q [16]. A vertex v_i is called critical for the point q when both edges of the polygon P adjacent to v_i lie one the same side of the line determined by qv_i.

If there exists $O(s)$ workspace, where s is $O(\log r)$ and r is the number of reflex vertices, Barba *et al.* [16] propose a solution for computing the 0-crossing visibility polygon from a given query point which runs in $O(nr/2^s + n\log r^2)$ and $O(nr/2^s + n\log r)$, deterministic and expected time respectively. Their algorithm uses a recursive approach where the constant memory algorithm is used as the base case. At each phase, the boundary of P is divided into two parts, such that the number of visible reflex vertices for q in these two sub-chains is roughly half of the visible reflex vertices in the original polygon. For stack-based algorithms, there exists a method proposed by Barba *et al.* [15] which gives a constrained memory algorithm for the 0-visibility query problem. This algorithm takes $O(n^2 \log n/2^s)$ time when s is $o(\log n)$.

In addition to computing the visibility polygon from a given query point q inside a simple polygon P with n vertices, an $O(mn)$-time algorithm is given to calculate

the weak visibility polygon from an edge of P using constant workspace [1], where m denotes to the size of the output polygon.

3.2 Guarding

3.2.1 Art Gallery

The original Art Gallery problem was introduced by Klee and Chvátal where the problem was to find the minimum number of guards needed such that each point within a given polygon is visible to at least one guard [93]. Chvátal and Fisk [42; 58; 93] proved that $\lfloor n/3 \rfloor$ guards are sufficient and sometimes necessary to guard a polygon P with n vertices. It has also been shown that for a given polygon P, determining the minimum number of point or edge guards needed is NP-hard [76]. For approximation algorithms, Eidenbenz *et al.* [52] showed that the Art Gallery problem cannot have any polynomial-time algorithm with an approximation factor better than a fixed constant unless $P = NP$. In addition to the above problems, some researchers have looked at approximation and randomized algorithms for locating a set of given guards g in a given polygon P to maximize the fraction of P that is guarded [62; 51].

The β-guarding problem has been studied for fixed β and different kinds of guards, such as vertex and edge guards. It has been shown that every simple polygon can be 2-guarded by at most $n-1$ edge guards [17]. Furthermore, every simple polygon can

be 1-guarded by at most $\lceil n/2 \rceil$ edge guards [17]. Belleville *et al.* also have shown that any polygon with holes can be 1-guarded by edge guards, while not all polygons with more than one hole can be 2-guarded [17]. Salleh proposed upper bounds of $\lfloor 2n/3 \rfloor$ and $\lfloor 3n/4 \rfloor$ for 2-guarding and 3-guarding simple polygons, respectively, by vertex guards [89]. The β-guarding problem has been shown to be NP-hard [28], and as a result, research has shifted to exploring approximation algorithms [28]; however for special polygons, such as spiral polygons, the 2-guarding problem can be solved in polynomial time [21].

3.2.2 k-Visibility

A guard g is called a k-modem (or k-transmitter) if it guards all points that are k-crossing visible from g. Recently Aichholzer *et al.* [3] have shown that $\lceil \frac{n-2}{2k+3} \rceil$ k-modems are sufficient, and in some cases necessary, for guarding monotone polygons. The authors also proved that a monotone orthogonal polygon can be guarded by $\lfloor n/(2k+4) \rfloor$ k-modems. Duque and Hidalgo-Toscano [50] showed that at most $O(n/k)$ k-modems are needed to guard a simple polygon P; however, given a polygon P, determining the minimum number of modems to guard P is an NP-hard problem, both for point k-modems (where $2 \leq k \leq n$) and edge 2-modems (an edge k-modem is an edge guard that is a k-modem) [29]. Given a set of line segments and a k-modem g_k, Fabila *et al.* investigated and proposed solutions for finding the minimum k needed such that the entire plane is k-crossing visible from g_k [57]. Additionally, k-crossing visibility can be considered in the plane with obstacles when the goal is to

guard the plane or boundary of geometric shapes. For instance, Ballinger *et al.* [13] developed upper and lower bounds on the number of k-modems needed to guard a set of orthogonal line segments, as well as for a few other special types of geometric objects.

3.2.3 Guarding on a Terrain

Chen *et al.* [40] claimed that Art Gallery problem on terrains is NP-hard, though the proof was later presented by King and Krohn [71]. As a result, research has since focused on approximation algorithms to solve the problem. In 2007, Ben-Moshe *et al.* [19] proposed the first constant-factor approximation algorithm forming the base of other algorithms in this area. A second constant-factor approximation algorithm was presented by Clarkson and Varadarajan where the more general class of problems were studied [43]. King introduced an approximation algorithm with constant factor five [70]. A 4-approximation algorithm was later presented by Elbassioni *et al.* [54]. Gibson *et al.* were also able to achieve a polynomial-time approximation scheme for a terrain (1.5D) by using a local search technique [61].

3.2.4 Watchtower Problem

The original terrain watchtower problem was introduced by Sharir for polyhedral terrains [92]. The minimum height for one watchtower can be found in $O(n \log n)$ time for both the continuous and discrete problems under 0-crossing visibility on an

x-monotone polyhedral terrain in $\mathbb{R}^3$ [96].

Bespamyatnikh *et al.* [22] proposed an $O(n^4)$-time algorithm for the discrete 2-watchtower problem under 0-crossing visibility on a terrain in $\mathbb{R}^2$. They also generalize their approach for the continuous version of the problem with assumptions on the time required to solve a specific cubic equation with three bounded variables. Under the assumption that the equation can be solved in $O(f_3)$ time, their approach takes $O(n^4 + n^3 f_3)$ time. Using parametric search, they show that the discrete and continuous versions of the problem can be solved in $O(n^3 \log^2 n)$ and $O(n^4 \log^2 n)$ time, respectively. Ben-Moshe *et al.* [18] improved the time to $O(n^{3/2}\sqrt{m'(n)})$ for the discrete 2-watchtower problem, where $m'(n)$ denotes the time required to multiply two $n \times n$ matrices, resulting in a time of $O(n^{2.88})$ using the current fastest matrix multiplication algorithm [44]. Using parametric search, Agarwal *et al.* [2] improved the time complexity of the discrete and continuous 2-watchtower problems for 0-crossing visibility to $O(n^2 \log^4 n)$ and $O(n^3 \alpha(n) \log^3 n)$ respectively, where $\alpha(n)$ denotes the inverse Ackermann function.

Chapter 4

Visibility Query without Preprocessing

Given a polygon P, an integer k, and a query point $q \in P$, we propose an algorithm that computes the region of P that is k-crossing visible from q in $O(nk)$ time, where n denotes the number of vertices of P. This is the first such algorithm parameterized in terms of k, resulting in asymptotically faster worst-case running time relative to previous algorithms when k is $o(\log n)$, and bridging the gap between the $O(n)$-time algorithm for computing the 0-visibility region of q in P [53; 75; 69] and the $O(n \log n)$-time algorithm for computing the k-visibility region of q in P [10].

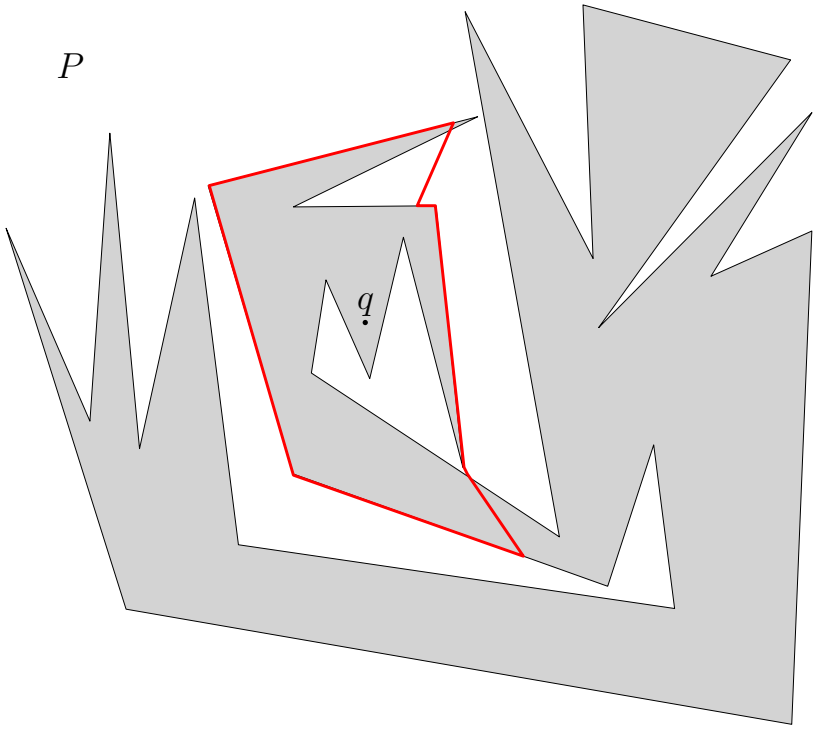


Figure 4.1: a polygon P, a point q, and the k-visibility polygon of q in P when $k = 2$

4.1 Introduction

Given a simple n-vertex polygon P, two points p and q inside P are said to be mutually visible when the line segment pq does not intersect the exterior of P. When p and q are in general position relative to the vertices of P (i.e., no vertex of P is collinear with p and q) p and q are mutually *k-crossing visible* when the line segment pq intersects the boundary of P in at most k points. For a formal definition of k-crossing visibility see Chapter 2. Various applications require computing the region of the plane that is visible or k-visible from a given query point q in P [4]. This region is called the *k-visibility polygon* of q in P. See Figure 4.1.

Our goal is to design an algorithm that reduces the time required for computing the k-visibility polygon for a given point q in a given simple polygon P. $O(n)$-time al-

gorithms exist for finding the visibility polygon of q in P (i.e., when $k = 0$) [53; 75; 69], whereas the best known algorithms for finding the k-visibility polygon of q in P require $\Theta(n \log n)$ time in the worst case for any given k [10]. A natural question that remained open is whether the k-visibility polygon of q in P can be found in $o(n \log n)$ time. In particular, can the problem be solved faster for small values of k? This chapter presents the first algorithm parameterized in terms of k to compute the k-visibility polygon of q in P. The proposed algorithm takes $O(nk)$ time, where n denotes the number of vertices of P, resulting in asymptotically faster worst-case running time relative to previous algorithms when k is $o(\log n)$, and bridging the gap between the $O(n)$-time for computing the 0-visibility polygon of q in P and the $O(n \log n)$-time algorithm for computing the k-visibility polygon of q in P.

Given a polygon P with n vertices and a query point q inside P, a fundamental problem in visibility is to compute the visibility polygon for q. Motivated by applications in wireless networks, where transmissions can pass through a number of obstacles before the signal degrades, this chapter focuses on finding the k-visibility polygon of q in P. Bajuelos *et al.* [12] subsequently explored the concept of k-crossing visibility for an arbitrary given k, and presented an $O(n^2)$-time algorithm to construct the k-crossing visible region of q in P for an arbitrary given point q. Recently, Bahoo *et al.* [10] examined the problem under the limited-workspace mode, and gave an algorithm that uses $O(s)$ words of memory and reports the k-visibility polygon of q in P in $O(n^2/s + n \log s)$ time. When memory is not constrainted (i.e., $\Omega(n)$ memory is available) their algorithm computes the k-visibility polygon in $O(n \log n)$ time.

The chapter begins with an overview of definitions, followed by the presentation of the algorithm, and an analysis of its running time.

4.2 Preliminaries and Definitions

4.2.1 Crossings and k-Visibility

In this chapter, the definition of k-visiblity is as in Chapter 2.

Given a simple polygon P, we refer to the set of points that are k-crossing visible from a point q as the *k-crossing visibility region of q with respect to P*, denoted $\mathcal{V}_k(P, q)$. When the polygon P is clear from the context, we simply refer to set as the k-crossing visibility region of q and denote it as $\mathcal{V}_k(q)$. Our goal is to design an efficient algorithm to compute the k-crossing visibility region of a point q with respect to a simple polygon P.

To simplify the description of our algorithms, we assume that the query point q and the vertices of the input polygon P are in general position, i.e., q, p_i and p_j are not collinear for any vertices p_i and p_j in P. Under the assumption of general position, two points p and q are k-crossing visible if and only if the line segment pq intersects the boundary of P in fewer than k points. That is, Definition 3 is not necessary under general position. All results presented in this chapter can be extended to input that is not in general position.

4.2.2 Trapezoidal and Radial Decompositions

A *polygon decomposition* of a simple polygon P is a partition of P into a set of simpler regions, such as triangles, trapezoids, or quadrilaterals. Our algorithm uses trapezoidal decomposition and radial decomposition. A *trapezoidal decomposition* (synonymously, *trapezoidation*) of P partitions P into trapezoids and triangles by extending, wherever possible, a vertical line segment from each vertex p of P above and/or below p into the interior of P, until its first intersection with the boundary of P. A *radial decomposition* of P is defined relative to a point q inside or outside P. Similarly, for each vertex p of P, a line segment is extended, wherever possible, toward/away from p into the interior of P on the line determined by p and q, until its first intersection with the boundary of P. A radial decomposition partitions P into quadrilateral and triangular regions. The number of vertices and edges in both decompositions is proportional to the number of vertices in P (i.e., $\Theta(n)$). Note that a trapezoidal decomposition corresponds to a radial decomposition when the point q has its y-coordinate at $\pm\infty$ (outside P). Chazelle [37] gives an efficient algorithm for computing a trapezoidal decomposition of a simple n-vertex polygon in $O(n)$ time.

4.2.3 Projective Transformations

Another topic we must cover in this section is *homogeneous coordinates* and its related concepts.

In computer graphics it is often necessary to perform operations such as rotation, shearing, scaling and translation on a two-dimensional image. A 3 by 3 matrix can

be used to perform a combination of these operations easily through a dot product with points of the image. To do so, each point x in the plane must be expressed with a 3D coordinate also referred to as homogeneous coordinates. The homogeneous coordinate of a point (x, y) is $(x, y, 1)$. The plane $Z = 1$ is called *real* plane in this setting. The points at infinity have representations in the homogeneous coordinates which lie in the plane $Z = 0$. Each set of parallel lines will meet at a point at infinity in the plane $Z = 0$.

For projecting an image from one plane to another plane in 3D, computer graphics researchers consider another geometry called *projective geometry*; a classical topic in mathematics. The projection of a point x of an image in the plane H to another plane H' from a point c ($c \notin H$), is the intersection of H' with the line cx. The point c is called the *central point*. If cx is parallel to H', the projective image of x appears at infinity and x is called a *vanishing point*. This transformation is called a *projective transformation*, and the plane to which the image projected called *projection plane*. After applying the projective transformation, we must bring back all points with $z \neq 0$ to the real plane $Z = 1$. As a result, a point (x, y, z) is represented as $(x/z, y/z, 1)$ in homogeneous coordinates and $(x/z, y/z)$ in Euclidean coordinates.

The transformation matrix can be shown as follows: $\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$.

Values in the first two rows will result in shearing, scaling, rotation, and transi-

tion. Values in the last row result in a projective transformation. When the rows are respectively [1 0 0], [0 1 0], and [0 0 1], the identity transformation is applied; no points will change under this transformation.

As mentioned, after projective transformation, each point must be transferred to the real plane. This is done by dividing all coordinates of each point to its z coordinate. As a result point (wx, wy, wz) and (x, y, z) represent the same point. Considering this, a point x in the real plane equals to a line in 3D which passes the plane $Z = 1$ at x. Expanding this discussion, each line L in the real plane is equal to a plane passing through the plane $Z = 1$ at L in 3D. In projective geometry, set of parallel lines meet at a point at infinity. This intersection point is a point in the plane $Z = 0$.

Suppose we have a simple polygon P and a point q in the Euclidian coordinates such that q lies below P. In the remainder of this section, we show how to define the matrix transformation so that the point q goes to infinity and the rest of P changes so that no point of P goes to infinity. Then, we explain how to use this transformation in order to report the radial decomposition of P from q.

Suppose a simple polygon P and a query point q are given so that q has the minimum y coordinate (as we want to move q to infinity but not the polygon P). Without loss of generality, suppose point $q = (0, 1)$. If not, we transfer P and q in the plane such that the x-coordinate and y-coordinate of q becomes 0 and 1, respectively. After

applying a projective transformation on P, if there is no point on the boundary of P that is transformed to the plane $Z = 0$, then the transformation of P remains a simple polygon. This is because in the projective transformation a point lies on a line if and only if the projective transformation of that point lies on the projective transformation of the line [23]. Also, notice that the projective transformation preserves lines [25]. So, if all vertices of the transformed P have positive z coordinates, P remains a simple polygon. The goal is to transform the query point q with coordinates (x_q, y_q, z_q) to $+\infty$ and the rest of P will change so that no point of P goes to infinity. After the transformation, the new coordinate of q must be $[x'_q \ y'_q \ 0]$. Suppose each vertex v_i of P with coordinates $[x_{v_i} \ y_{v_i} \ z_{v_i}]$ be projected to point v'_i with coordinate $[x'_{v_i} \ y'_{v_i} \ z'_{v_i}]$. If for each vertex v_i of P, z'_i remains positive (or negative), no point of the boundary of P will go to infinity and intersect with the plane $Z = 0$.

By considering these facts, there are two criteria which must be satisfied:

- $h_{31}x_q + h_{32}y_q + h_{33} = 0$
- $\forall v_i \in P : h_{31}x_{v_i} + h_{32}y_{v_i} + h_{33} > 0$

By satisfying the above conditions the point q will move to $+\infty$ and the rest of the polygon changes so that no point of P goes to infinity.

By defining the transformation matrix $\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & -1 \end{bmatrix}$ for transforming the polygon P and the point q, both the above criteria will be satisfied. Notice that the point

q will be transformed to the point $(0, 1)$ in the plane $Z = 0$. Let's call this matrix TM.

Lemma 1. *Given a simple polygon P and a point q where q lies below P, the rays emanating from q going through vertices of P are transferred to the vertical lines in the projective image under the transformation matrix TM and their ordering based on their x-coordinate is the polar ordering of the equivalent rays around q in the original image.*

Proof. Suppose x' and L' are the projective transformation of a point x and a line L. x' lies on L' if and only if x lies on L. When applying the projective transformation the point q will go to infinity. This point is the only common point between the rays emanating from q that were passing the vertices of P. This is because we are assuming that the vertices of P are in general position with respect to q. As a result, the transformation of these rays will never meet in the real plane. Consequently, they are parallel in the real plane.

Suppose the projective plane and the real plane intersect at a line called L. The rays emanating from q will intersect with L in the order they appear around q. These rays equivalent to parallel lines in the projective plane. L is also in the projective plane and the points of the intersection of the rays with L belong to the parallel lines which are equivalent to the rays emanating from q in the projective image. So, their ordering is the same as their polar order around q in P.

We now show that these parallel lines will be vertical lines in the transformed

image. Any set of parallel lines will intersect at a point at infinity in the plane $Z = 0$ in projective geometry. Each point at infinity can be shown as $(x, y, 0)$, which can be rescaled to the point $(x, 1, 0)$ where x is $1/m$ and m is the slope of a set of parallel lines. So, a set of vertical lines in the plane intersecting at $(0, 1, 0)$. $(0, 1, 0)$ is the transformed version of q where q goes to $+\infty$. q is the intersection of the set of parallel lines equal to the transformation of the rays passing through q and vertices of P. Consequently, these parallel lines are vertical. □

Using Lemma 1, we conclude that the radial decomposition of a simple polygon P around the point q corresponds to the trapezoidal decomposition of the transformed polygon P' under the transformation matrix TM, and can be calculated in $\Theta(n)$ time.

Lemma 2. *The radial decomposition of a simple n-vertex polygon P around a query point q can be calculated in $\Theta(n)$ time.*

Proof. Given a simple polygon P and a query point q, P can be transformed to a simple polygon P' in linear time by transformation matrix TM, where q goes to infinity. Chazelle showed that the trapezoidal decomposition can be found in linear time [37]. By Lemma 1, a vertical line segment of the trapezodial decomposition of P' corresponds to a ray emanating from q and passing a vertex of P. If this vertical line segment intersects edges of P', called e'_2 and e'_1, it must intersect the corresponding edges e_1 and e_2 in P; and it cannot intersect any other edges of P as a point lies on a line if and only if the transformation of that point lies on the transformation of the line [23]. So, the trapezoidal decomposition of P' equals to the radial decomposition of P around the point q. The trapezoidal decomposition can be transformed back by applying the transformation matrix TM^{-1} to obtain the radial decomposition of P.

This process needs $O(n)$ time complexity. As a result, the radial decomposition of P can be constructed in $O(n)$ time complexity.

□

4.3 k-Visibility Algorithm

4.3.1 Overview

Given as input an integer k, an array storing the coordinates of vertices whose sequence defines a clockwise ordering of the boundary of a simple polygon P, and a point q in the interior of P, our algorithm for constructing the k-visibility polygon of q in P executes the following steps, each of which is described in detail in this section:

1. Partition P into two sets of disjoint polylines, corresponding to the boundary of P above the horizontal line ℓ through q, and the boundary of P below ℓ.

2. Nesting properties of Jordan sequences are used to close each set by connecting disjoint components to form two simple polygons, P_a above ℓ and P_b below ℓ.

3. The two-dimensional coordinates of the vertices of P_a and P_b are mapped to homogeneous coordinates, to which a projective transformation, f_q, is applied, with q as the center of projection.

4. Compute the trapezoidal decompositions of $f_q(P_a)$ and $f_q(P_b)$ using Chazelle's algorithm [37].

5. Apply the inverse tranformation f_q^{-1} on the trapezoidal decompositions to obtain radial decompositions of P_a and P_b.

6. Merge the radial decompositions of P_a and P_b to obtain a radial decomposition of P.

7. Traverse the radial decomposition of P to identify the visibility of cells in increasing order from visibility 0 through visibility k, moving away from q and extending edges on rays from q to refine cells of the decomposition as necessary.

8. Traverse the refined radial decomposition to reconstruct and output the boundary of the k-visibility region of q in P.

Steps 1–6 can be completed in $O(n)$ time and Steps 7–8 can be completed in $O(nk)$ time.

4.3.2 Partitioning P into Upper and Lower Polygons

We begin by describing how to partition the polygon P into two pieces across the line ℓ, where ℓ denotes the horizontal line through q. We rotate P so that no vertices of P lie on ℓ. Let ϵ denote the minimum distance between any vertex of P and ℓ. Let $\{x_1, \ldots, x_m\}$ denote the sequence of intersection points of ℓ with the boundary of P, labelled in clockwise order along the boundary of P, such that x_1 is the leftmost point in $P \cap \ell$. This sequence is a *Jordan sequence* [65]. We now describe how to construct the upper polygon P_a and the lower polygon P_b. Notice that the following Lemma holds:

Lemma 3. *Suppose that the intersection points between the horizontal line and polygon P is labeled as $1, 2, \ldots,$ and m; then m is always an even number.*

Proof. P is a closed Jordan curve. The number of points of intersection of a closed Jordan curve and a line h is odd when the line is tangent to the curve. This occurs when there is a vertex of P on h so that h is tangent to P at that vertex. But we rotate P so that no vertex of P lies on h. As a result, the number of intersections is even. □

Between consecutive pairs (x_{2i-1}, x_{2i}) of the Jordan sequence, for $i \in \{1, \ldots, m/2\}$, the polygon boundary of P lies above ℓ. Notice that x_1 was leftmost in $P \cap \ell$; a walker walking along ∂P in clockwise order always traverses from the region below ℓ to the region above ℓ. Similarly, between pairs (x_{2j}, x_{2j+1}), for $j \in \{1, \ldots, m/2-1\}$, and between (x_m, x_1), the boundary of P lies below ℓ. We call the former *upper pairs* of the Jordan sequence, and the latter *lower pairs*. These pairs possess the *nested parenthesis* property [88]: every two pairs (x_{2i-1}, x_{2i}) and (x_{2j-1}, x_{2j}) must either *nest* or be *disjoint*. That is, x_{2j-1} lies between x_{2i-1} and x_{2i} in the sequence if and only if x_{2j} lies between x_{2i-1} and x_{2i}. This is because if they are not nested, the part of ∂P between x_{2j-1} and x_{2j} in clockwise order (which is above ℓ) must intersect the part of ∂P between x_{2i-1} and x_{2i} in clockwise order which is above ℓ. As P is a simple polygon such an intersection can not exist.

As shown by Hoffmann *et al.* [65], the nested parenthesis property for the upper pairs determines a rooted tree, called the *upper tree*, whose nodes correspond to pairs of the sequence. The nodes in the subtree rooted at the pair (x_{2i-1}, x_{2i}) consist of all nodes corresponding to pairs that are nested betweeen x_{2i-1} and x_{2i} in the Jordan sequence order. The leaves of the tree correspond to pairs that are consecutive in the

sorted order (sorted by their x-coordinates). If a node (x_{2j-1}, x_{2j}) is a descendant of a node (x_{2i-1}, x_{2i}) in the tree, then the points x_{2j-1} and x_{2j} are nested between x_{2i-1} and x_{2i}. The *lower tree* is defined analogously.

If the boundary of P intersects ℓ in more than two points, the resulting disconnected components must be joined appropriately to form the simple polygons P_a and P_b. To build the lower polygon P_b, we replace each portion of the boundary of P above ℓ from x_{2i-1} to x_{2i} with the following 3-edge path: x_{2i-1}, u, v, x_{2i}. The first edge (x_{2i-1}, u) is a vertical line segment of length $\epsilon/2d_i$, where d_i denotes the depth of the node (x_{2i-1}, x_{2i}) in the tree (we insert a dummy root vertex to ensure that no nodes of the tree has the depth of zero). Notice that if ℓ be the x-axis, all such u and v are located below (or above) ℓ while constructing P_b (or P_a). The next edge (u, v) is a horizontal line segment whose length is $||x_{2i-1} - x_{2i}||$. The third edge (v, x_{2i}) is a vertical line segment of length $\epsilon/2d_i$. Figure 4.2 illustrates this construction. Notice that all parts of these paths are located below the line ℓ.

The nesting property of the Jordan sequence ensures that all of the 3-edge paths are similarly nested and that none of them intersect. Consider two pairs (x_{2i-1}, x_{2i}) and (x_{2j-1}, x_{2j}). Either they are disjoint or nested. If they are disjoint, then without loss of generality, assume that $x_{2i-1} < x_{2i} < x_{2j-1} < x_{2j}$. Their corresponding 3-edge paths cannot cross since the intervals they cover are disjoint. If they are nested, then without loss of generality, assume that $x_{2i-1} < x_{2j-1} < x_{2j} < x_{2i}$. The only way that the two paths can cross is if the horizontal edge for the pair (x_{2j-1}, x_{2j}) is higher

than for the pair (x_{2i-1}, x_{2i}). However, since (x_{2j-1}, x_{2j}) is deeper in the tree than (x_{2i-1}, x_{2i}), the two paths do not cross. Thus, we form the simple polygon P_b by replacing the portions of the boundary above ℓ with these three edge paths. Sorting the Jordan sequence, building the upper tree, computing the depths of all the pairs and adding the 3-edge paths can all be achieved in $O(n)$ time using the Jordan sorting algorithm outlined by Hoffmann *et al.* [65]. The upper polygon P_a is constructed analogously. We conclude with the following lemma.

Lemma 4. *Given a simple n-vertex polygon P and a horizontal line ℓ that intersects the interior of P such that no vertices of P lie on ℓ, the upper and lower polygons of P with respect to ℓ can be computed in $O(n)$ time.*

4.3.3 Computing the Radial Decomposition

The two-dimensional coordinates of the vertices of each polygon P_a and P_b are mapped to homogeneous coordinates, to which a projective transformation, f_q, is applied with q as the center of projection. These transformations take constant time per vertex, or $\Theta(n)$ total time. Chazelle's algorithm [37] constructs trapezoidal decompositions of $f_q(P_a)$ and $f_q(P_b)$ in $\Theta(n)$ time, on which the inverse transformation, f_q^{-1} is applied to obtain radial decompositions of P_a and P_b. Merging the radial decompositions of P_a and P_b gives a radial decomposition of the original polygon P without requiring any additional edges. All vertices $x_1, \ldots, x_m$ of the Jordan sequence, all vertices of the three-edge paths, and their adjacent edges are removed. The remaining edges are either on the boundary of P, between two points on the boundary on

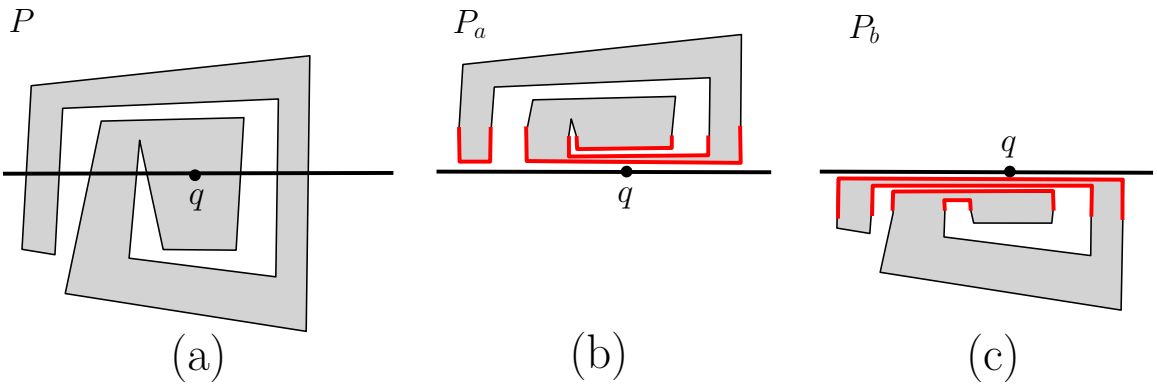


Figure 4.2: (a) a polygon P, a point q, and the horizontal line ℓ through q; (b)–(c) the upper polygon P_a and lower polygon P_b of P with the additional 3-edge paths highlighted.

a ray through q, or between the boundary and q. The entire process for constructing the radial trapezoidation takes $\Theta(n)$ time. This gives the following lemma.

Lemma 5. *The radial decomposition of a simple n-vertex polygon P around a query point q can be computed in $\Theta(n)$ time.*

4.3.4 Reporting the k-Visible Region

The 0-visibility region of q in P, denoted $\mathcal{V}_0(q)$, is a star-shaped polygon with q in its kernel. A vertex of $\mathcal{V}_0(q)$ is either a vertex v of P or a point x on the boundary of P that is the intersection of an edge of P with a ray emanating from q through a reflex vertex r of P. In the latter case, (r, x) is an edge of $\mathcal{V}_0(q)$ that is collinear with q, called a *window* or *lid*, because it separates a region in the interior of P that is 0-visible from q and an interior region that is not 0-visible. The reflex vertex r is the *base* of the lid and x is its *tip*. There are two types of base reflex vertices. The reflex vertex r is called a *left base* (respectively, *right base*) if the polygon edges

incident on r are to the left (respectively, right) of the ray emanating from q through r.

We now describe the algorithm to compute the k-visible region of q in P, denoted $\mathcal{V}_k(q)$. The algorithm proceeds incrementally by computing $\mathcal{V}_{i+1}(q)$ after computing $\mathcal{V}_i(q)$. We begin by computing $\mathcal{V}_0(q)$ in $O(n)$ time using one of the existing linear-time algorithms, e.g. [53; 75; 69]. Label the vertices of $\mathcal{V}_0(q)$ in clockwise order around the boundary as $x_0, x_1, \ldots, x_m$. Triangulate the visibility polygon by adding the edge (q, x_i) for $i \in \{0, \ldots, m\}$; this corresponds to a radial decomposition of $\mathcal{V}_0(q)$ around q.

If x_i is a left base vertex, then notice that the triangle $\triangle(qx_ix_{i+1})$[1] degenerates to a segment. Similarly, if x_i is a right base vertex, then $\triangle(qx_ix_{i-1})$ is degenerate. If we ignore all degenerate triangles, then every triangle has the form $\triangle(qx_ix_{i+1})$, where (x_i, x_{i+1}) is on the boundary of P. The union of these triangles is $\mathcal{V}_0(q)$. To compute $\mathcal{V}_1(q)$, we show how to compute a superset of triangles whose union is $\mathcal{V}_1(q)$.

We start with an arbitrary triangle $\triangle(qx_ix_{i+1})$ of $\mathcal{V}_0(q)$, where (x_i, x_{i+1}) is on the boundary of P. Note that (x_i, x_{i+1}) is either an edge of P or a segment within the interior of an edge of P, where each endpoint is either a vertex of P or on the interior of an edge of P. It is this segment (x_i, x_{i+1}) of the boundary that blocks visibility. We show how to compute the intersection of $\mathcal{V}_1(q)$ with the cone that has apex q and bounding rays $q\vec{x}_i$ and $q\vec{x_{i+1}}$, denoted $\mathcal{C}(q, x_i, x_{i+1})$. We call this process *extending the visibility* of a triangle. We have two cases to consider. Either one of x_i or x_{i+1} is a base vertex or neither is a base vertex. It is not possible for x_i and x_{i+1} to both be

[1] All indices are computed modulo the size of the corresponding vertex set: $m + 1$ in this case.

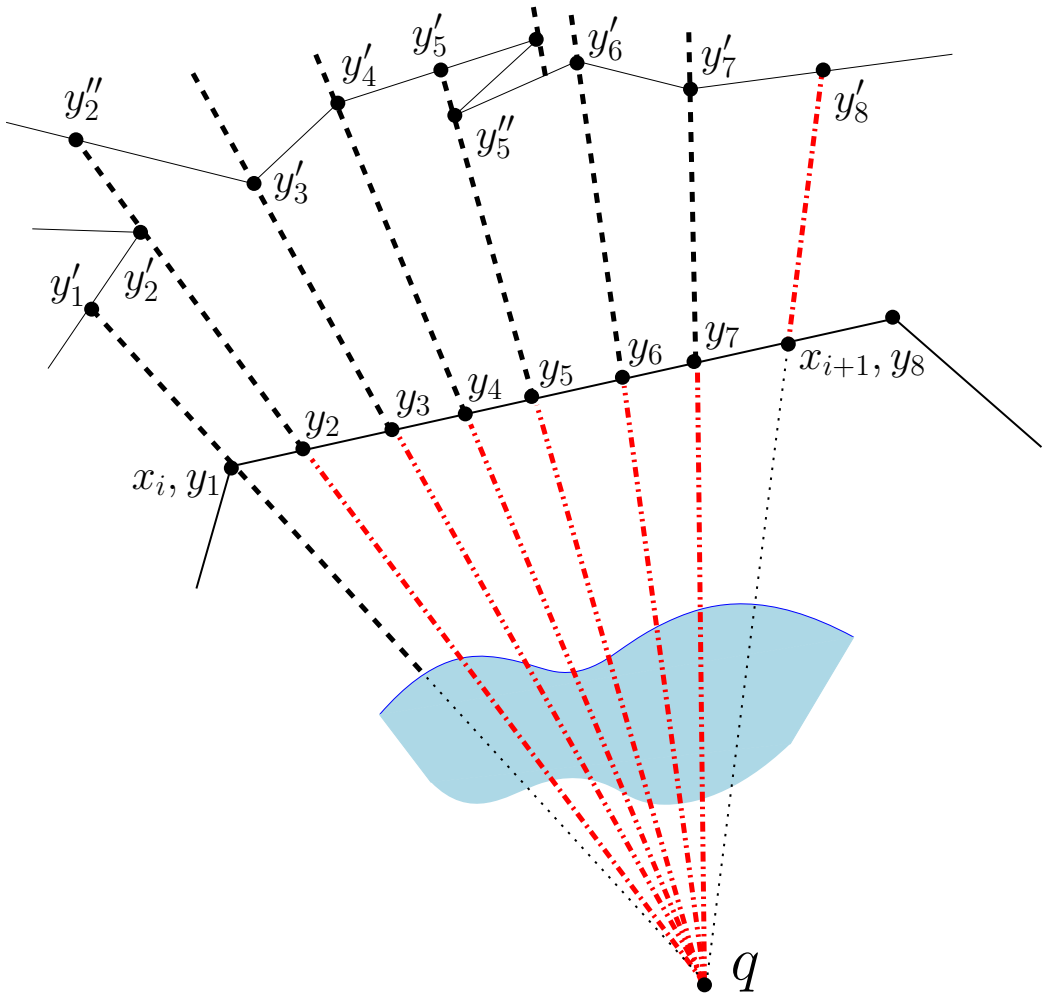


Figure 4.3: Edges of the radial decomposition are extended where critical vertices cast a shadow. Portions of the polygon in the blue region that were processed in previous iterations are omitted from the figure.

base vertices. We start with the latter case where neither is a base vertex.

Let Y be the set of vertices of the radial decomposition that lie on the edge (x_i, x_{i+1}). If Y is empty, then (x_i, x_{i+1}) lies on one face of the radial decomposition since neither x_i nor x_{i+1} is a base vertex. We show how to proceed in the case when Y is empty, then we show what to do when Y is not empty. Let f be the face of the

decomposition on the boundary of which (x_i, x_{i+1}) lies. By construction, this face is either a quadrilateral or a triangle. In constant time, we find the intersection of the boundary of f excluding the edge containing (x_i, x_{i+1}) with $q\vec{x}_i$ and $q\vec{x_{i+1}}$. Label these two intersection points as x'_i and x'_{i+1}. Extending the visibility of $\triangle(qx_ix_{i+1})$ results in $\triangle(qx'_ix'_{i+1})$. Note that $\triangle(qx'_ix'_{i+1})$ is the 1-visible region of q in $\mathcal{C}(q, x_i, x_{i+1})$ and (x'_i, x'_{i+1}) is on the boundary of P.

We now show how to extend the visibility of $\triangle(qx_ix_{i+1})$ when Y is not empty. Label the points of Y as y_j for $j \geq 1$ in the order that they appear on the edge (x_i, x_{i+1}) from x_i to x_{i+1}. Each y_j is an endpoint of an edge of the radial decomposition. Since y_j is a point on the boundary of P, there are 2 faces of the radial decomposition with y_j on the boundary. Let y'_j be the other endpoint of the face on the left of y_j and y''_j be the endpoint for the face on the right. Either $y'_j = y''_j$ or $y'_j \neq y''_j$. In the former case, we simply ignore y''_j. In the latter case, we note that either y'_j is a left base of $\mathcal{V}_0(y_j)$ or y''_j is a right base. See Figure 4.3 where y'_2 is a left base and y''_5 is a right base.

Thus, the edges of the radial decomposition that intersect segment (x_i, x_{i+1}) are of the form (y_j, y'_j) or (y_j, y''_j). Note that y_1 is either x_i or the point closest to x_i on the edge. For notational convenience, if $y_1 \neq x_i$, relabel x_i as y_0. Let f be the face of the radial decompostion on the boundary of which (y_0, y_1) lies. Let y'_0 be the intersection of $q\vec{y}_0$ with the boundary of f excluding the edge of f containing (y_0, y_1). We call this operation *extending* x_i. Similarly, for y_j that is closest point in Y to x_{i+1}, if $y_j \neq x_{i+1}$, relabel x_{i+1} as y_{j+1} and compute the edge (y_{j+1}, y'_{j+1}), i.e. *extend* x_{i+1}.

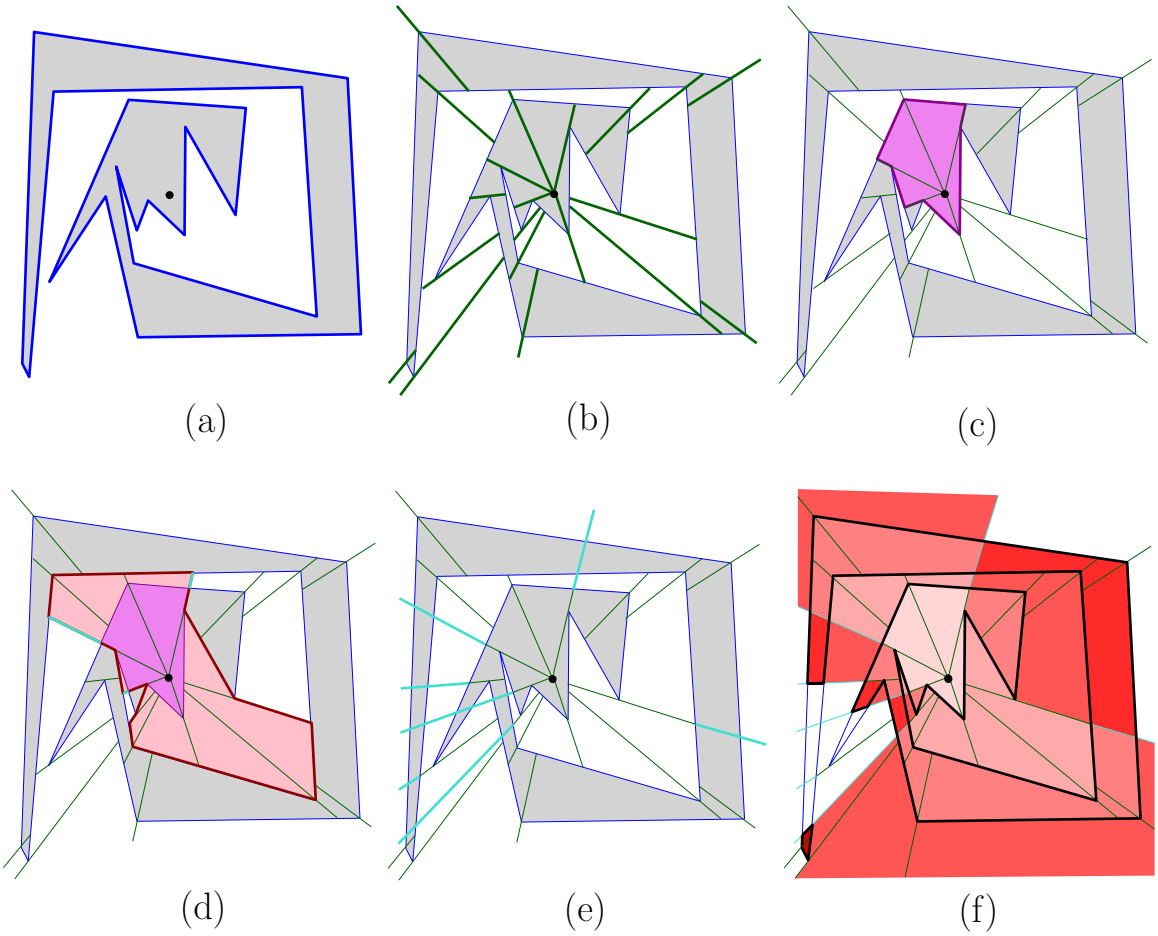


Figure 4.4: (a) a simple polygon P and a query point q; (b) the radial decomposition of P; (c) the 0-visibility polygon, $\mathcal{V}_0(q)$, of q in P computed in the first iteration; (d) the 1-visibility polygon, $\mathcal{V}_1(q)$, of q in P computed in the second iteration, with extended edges highlighted in light blue; (e) the refined radial decomposition, with extended edges highlighted in light blue; (f) the 4-visibility polygon, $\mathcal{V}_4(q)$, of q in P computed in the fourth iteration, with the algorithm's output highlighted in black (two components of the boundary of $\mathcal{V}_4(q) \cap P$), and cells of the decomposition with depth ≤ 4 coloured by depth, as computed by the algorithm.

We are now in a position to describe the extension of the visibility of triangle $\triangle(qx_ix_{i+1})$ when neither x_i nor x_{i+1} is a base vertex. The set of triangles are $\triangle(qy'_ky'_{k+1})$ and $\triangle(qy''_ky'_{k+1})$ (when y''_k exists). The union of these triangles is the 1-

visible region of q in $\mathcal{C}(q, x_i, x_{i+1})$. Furthermore, notice that each triangle $\triangle(qy'_k y'_{k+1})$ (respectively, $\triangle(qy''_k y'_{k+1})$) has the property that (y'_k, y'_{k+1}) (respectively, (y''_k, y'_{k+1})) is on the boundary of P. This is what allows us to continue incrementally since at each stage we extend the visibility of a triangle $\triangle(qab)$ where (a, b) is on the boundary of P.

Now, if x_i is a base vertex, then it must be a right base. Of the two edges of P incident on x_i, let e be the one further from q. The procedure to extend $\triangle(qx_i x_{i+1})$ is identical except that we only extend x_i when $x_{i+1} \in e$. Notice that e is defined as such due to the fact that when x_{i+1} is not located on e, the ray is already extended for the next step of the algorithm and if x_{i+1} is locate on e, we need to start extending the ray for the next step. Similarly, if x_{i+1} is a base vertex, then it must be a left base. Of the two edges of P incident on x_{i+1}, let e be the one further from q. Again, the procedure to extend $\triangle(qx_i x_{i+1})$ is identical except that we only extend x_{i+1} when $x_i \in e$.

The general algorithm proceeds as follows. At iteration i, the visibility region $\mathcal{V}_i(q)$ is represented as a collection of triangles around q with the property that the edge of the triangle opposite q is on the boundary of P and it is the edge blocking visibility. We wish to extend past this edge to compute $\mathcal{V}_{i+1}(q)$ from $\mathcal{V}_i(q)$. To do this, we extend each triangle in $\mathcal{V}_i(q)$. There are at most $O(n)$ triangles at each level. Therefore, the total time to extend all the triangles in $\mathcal{V}_i(q)$ is linear. Thus, we can compute $\mathcal{V}_{i+1}(q)$ from $\mathcal{V}_i(q)$ in $O(n)$ time and computing $V_k(q)$ takes $O(nk)$ time since we repeat this process k times.

The algorithm can report either only the subregion of P that is k-visible from q, i.e., $\mathcal{V}_k(q) \cap P$, or the entire region of the plane that is k-visible from q, including parts outside P. To obtain the region inside P, it suffices to traverse the boundary of P once to reconstruct and report portions of boundary edges that are k-visible. The endpoints of these sequences of edges on the boundary of P meet an edge of the refined radial decomposition through the interior of P that bridges to the start of the next sequence on the boundary of P. The entire boundary of P must be traversed since the k-visible region in P can have multiple connected components (unlike the k-visible region in the plane that is a single connected region). See Figure 4.4 for an example. We conclude with the following theorem.

Theorem 1. *Given a simple polygon P with n vertices and a query point q in P, the region of P that is k-crossing visible from q can be computed in $O(kn)$ time without preprocessing.*

Chapter 5

Visibility Query with Preprocessing

In this chapter, we examine the problem of preprocessing a given simple polygon P for a given integer k to construct a data structure to support efficient visibility queries, where each query consists of a point q inside P for which the k-visibility region of q in P must be returned. The objective is to balance the trade-off between the size of the data structure and the query time. Given a polygon P and an integer k, we describe how to preprocess P in $O(n^5 \log n)$ time to construct a data structure of size $O(n^5)$. Using this data structure the k-visibility region for any query point q given at query time can be found in $O(\log n + m)$ time, where m refers to the number of vertices of the k-visibility region.

5.1 Introduction

Given a simple polygon P with n vertices and a query point q inside P, a fundamental problem in visibility is to compute the visibility region for q: the region of the the polygon P 0-visible from q.

For a formal definition of k-crossing visibility see Chapter 2. Given a point q inside the polygon P, the goal in this chapter is to design a data structure that can determine the k-crossing visible part of the polygon for a query point q, the *k-visibility region*. To simplify the description of the presented algorithms, it is assumed that the query point q and the vertices of the input polygon P are in general position, i.e. q, p_i and p_j are not collinear for any vertices p_i and p_j in P.

Section 5.2 describes a data structure and a query algorithm for constructing the k-visibility region for the query point q. Section 5.2.1 proposes a query data structure and associated query algorithm for determining the k-visibility region from q by preprocessing P, for a point q and a positive number k given at query time.

5.2 Query Data Structure and Algorithm for a Fixed Polygon and a Fixed k

Given a polygon P, the goal is to preprocess P so that the k-crossing visibility region of a query point q given at query time is determined efficiently. The purpose

of this preprocessing step is to reduce the overall query time by shifting some of the computation specific to P and independent of q by constructing a query data structure. Note that the preprocessing step only has knowledge of the input polygon P and the number k; the query point q is provided only at query time.

To achieve this goal, the polygon can be decomposed into a set of cells; Figure 5.1 represents such a decomposition. For each cell, some information about the k-visibility region must be stored so that for a point lying in that cell, the k-visibility region can be quickly determined using the stored information. We denote such information as the combinatorial representation of the k-visibility region, henceforth simply referred to as the *combinatorial representation*. In this section, we first define the decomposition of P, provide a precise definition of the combinatorial representation, and then show how the proposed decomposition maintains the combinatorial representation in each cell. Finally, we present the process of constructing the k-visibility region of a given query point q by using the combinatorial representation stored in the cell containing q.

We decompose a polygon into a set of cells by considering the following:

1. The boundary of the *k-visibility region* of each vertex of P.
2. The boundary of the *v-region* of each vertex of P.
3. The *order lines*.

We describe each of these below.

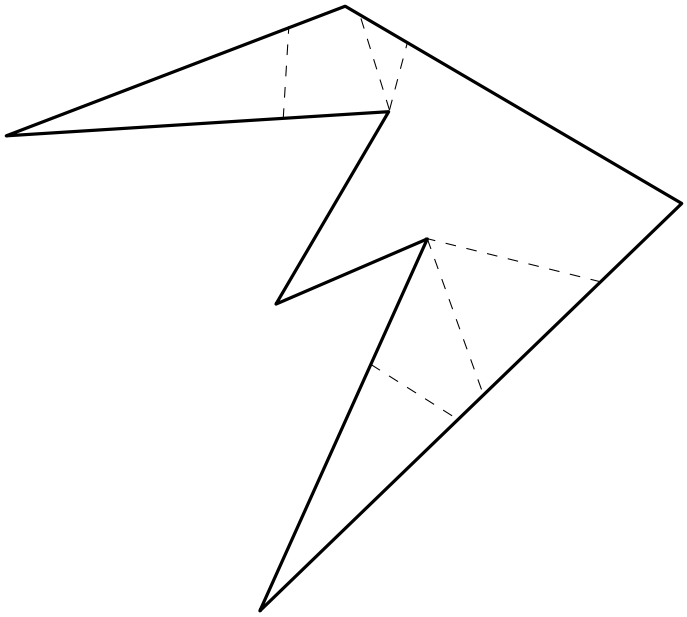


Figure 5.1: The combinatorial representation of the 2-visibility region remains the same at any point inside any given cell.

Consider the k-visibility region of a vertex in P. The boundary of this region is a set of line segments in P whose endpoints lie on the boundary of P, and are used to decompose the polygon. The k-visibility region of each vertex in P is calculated in $O(n \log n)$ time [10], as shown in Chapter 6. Hence, the k-visibility regions for all vertices of P can be determined in $O(n^2 \log n)$ time. As each k-visibility region has $O(n)$ vertices [10], this step introduces $O(n^2)$ line segments for the decomposition.

Next, we present the definition of the the *v-region* proposed by Evans and Sember [55].

Definition 4 (Evans and Sember [55])**.** *The v-region of a vertex v in P includes the points x where x is k-crossing visible for any point in P on the ray* $\overrightarrow{xv}$.

In other words, the v-region of a vertex v can be defined as any point x whose

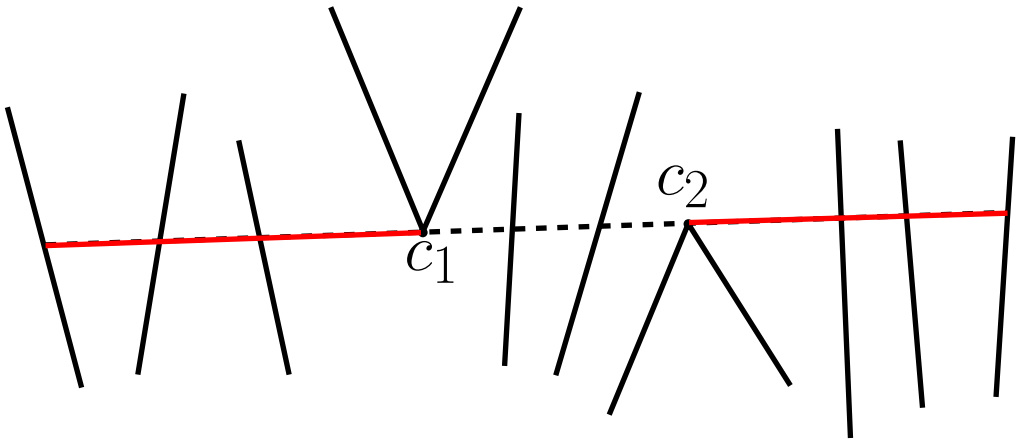


Figure 5.2: The bold red lines are added as order lines when $k = 3$.

emanating ray passing through v does not cross the edges of P more than k times. If both edges incident to v lie on the same side of the ray $\overrightarrow{xv}$, its edges are counted as two intersection points. We calculate the v-region of each vertex of P, where the boundary of each v-region is a set of rays and line segments. The boundaries of the regions are considered for the decomposition. Each v-region can be calculated by the approach proposed by Evans and Sember in $O(n \log n)$ time for each vertex of P, resulting a total time of $O(n^2 \log n)$ needed to calculate all regions [55]. As each v-region has $O(n)$ edges [55], this step introduces $O(n^2)$ new line segments and rays into the plane.

We define the vertex c to be *critical* for q when the edges incident to c lie on one side of the ray $\overrightarrow{qc}$. Suppose two vertices c_1 and c_2 are critical and k-visible from each other, and n' edges of P intersect the segment c_1c_2 where $n' \leq k$. Consider the two rays on the line c_1c_2, emanating from c_1 and c_2, respectively, traveling away from both c_1 and c_2 until encountering at most $k + 1 - n'$ edges of P (the two edges incident to c_1 are counted as two intersection points, as are the edges incident to c_2); this is

illustrated in Figure 5.2. If the number of intersection points of these rays with the polygon is less than $k+1-n'$, we consider the ray for the decomposition, otherwise the line segment is used. These line segments or rays used for the cell decomposition are referred to as *order lines*. As there are $O(n^2)$ order lines, and the intersection of one order line with the polygon can be found in $O(n)$ time, this step takes $O(n^3)$ time in total.

Lemma 6. *The decomposition of P, constructed by the boundary of the k-visibility region and v-region of each vertex of P, and order lines, has $O(n^4)$ cells.*

Proof. There are $O(n^2)$ line segments and rays considered for the decomposition, and the polygon P has $O(n)$ edges. Considering the entire plane, all these line segments and rays can intersect $O(n^4)$ times. As a result, they partition the entire plane into $O(n^4)$ cells. The polygon P is a subset of the plane, so there exist $O(n^4)$ cells in the decomposition of the polygon. □

The decomposition of P constructed by the boundary of the k-visibility region and v-region of each vertex of P, and order lines, is called *k- cell decomposition.*

Lemma 7. *There exists a polygon P whose k- cell decomposition has $\Theta(n^4)$ cells.*

Proof. The k-kernel is a region from which every point in P is k-crossing visible. Evans and Sember [55] demonstrated a case where the k-kernel has $\Theta(n^4)$ vertices, and $\Theta(n^4)$ components, each of which lies inside P. Each such component is the intersection of the v-regions of all the vertices of the polygon P. Furthermore, this example showed that there exists a case for the proposed cell decomposition in which

$\Omega(n^4)$ cells exist. Hence, $\Theta(n^4)$ is the tightest bound on the number of cells in the worst case.

□

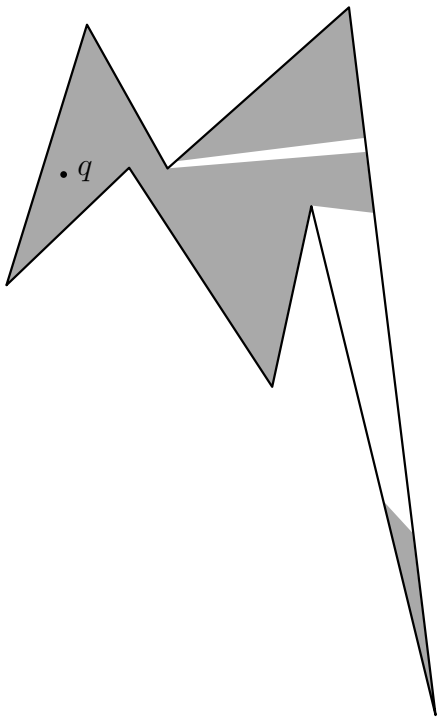


Figure 5.3: The gray polygons are the components of the 2-visibility region.

The vertices of the k-visibility region are either the vertices of the polygon P, or lie on an edge of P. The boundary of the k-visibility region is composed of some parts of the boundary of P and some line segments, which will be referred to as *windows*. Each window lies on a ray emanating from a point q, and passes through a critical vertex b k-visible from q so that the ray $\overrightarrow{qb}$ intersects the boundary of the polygon more than $k+1$ times (edges of b count as two intersection points). Such a critical vertex is called the *base* of the window. Also, notice that the k-visibility region of

a query point can be disconnected, consisting of a set of simple polygons called the *components* of the k-visibility region, as shown in Figure 5.3.

Observation 1. *A base vertex b creates a window whose endpoints are the $(k+2)^{th}$ and $(k+3)^{th}$ intersections of the ray $\overrightarrow{qb}$ with the polygon boundary (considering the edges of b as two intersection points), where there exists more than $k+2$ intersections between the edges of P and the ray $\overrightarrow{qb}$; see Figure 5.5.*

With the above definition of the k-cell decomposition, we can now define the combinatorial representation of the k-visibility region of a query point q as follows. Consider a component of the k-visibility region. This component can be represented by the vertices of the polygon P on the boundary of the component of the k-visibility region, and the endpoints of the windows on the boundary of the component. The endpoints of the windows can be represented by a single element defined by the base vertex of the window and the two edges that the window lies on; see the third element in Figure 5.4a. This information will be stored in the order that the vertices appear along the border of a component of the k-visibility region of q and represents that component of the k-visibility region of q. If the component includes any vertex of the polygon P, this list starts from the vertex with minimum index, and if there is no vertex of P in the component, the representative list starts from the base with minimum index. The set of such lists for all components of the k-visibility region is called the combinatorial representation. Figure 5.4 illustrates an example of the combinatorial representation.

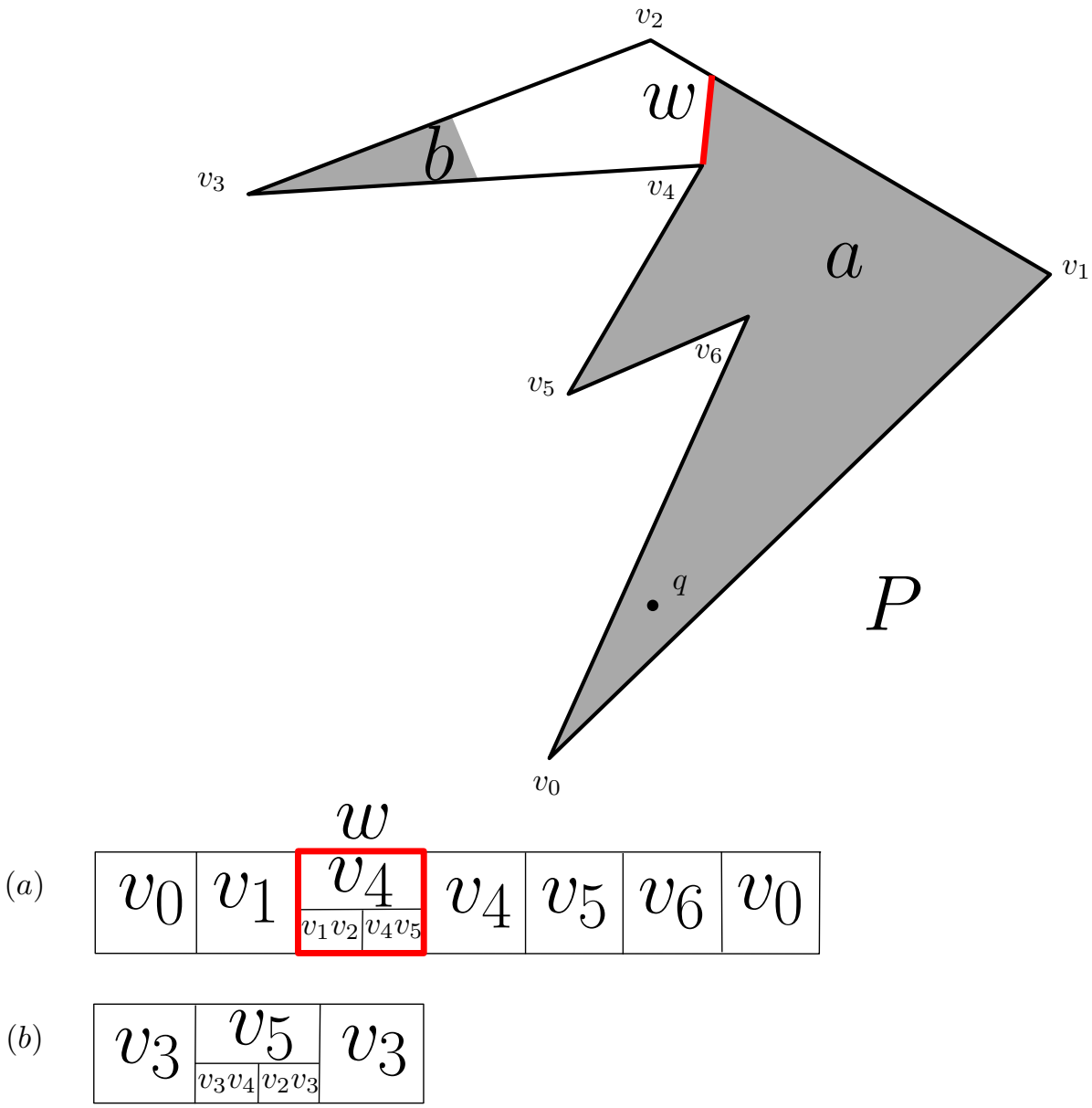


Figure 5.4: The gray region is the k-visible part of the polygon P for the point q. Figure (a) is the representation of the component a of the k-visibility region, and (b) shows the corresponding representation of the component b. These two representation together correspond to the combinatorial representation of the 2-visibility region for the point q. Notice that a window is shown by its base vertex and two edges on which its endpoints lie; see the third element in Figure (a) which corresponds to the window w in the k-visibility region.

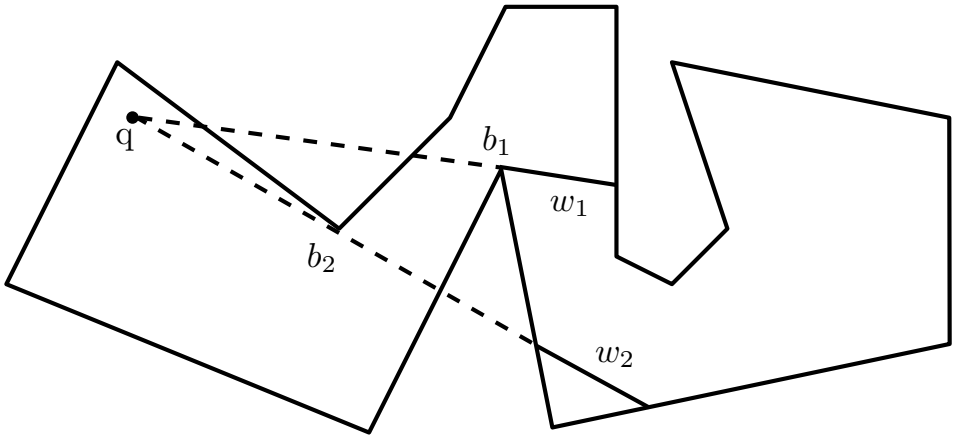


Figure 5.5: When $k = 2$, w_1 and w_2 are the windows of 2-visibility with base vertices b_1 and b_2 respectively.

We next show that all points inside a given cell of the k-cell decomposition have the same combinatorial representation.

Lemma 8. *Points lying in each face of the k-cell decomposition have the same combiniatorial representation for their k-visibility region.*

Proof. Suppose two points x_1 and x_2 are in a cell, and have different combinatorial representations. Notice that a cell of the k-cell decomposition is connected. So, without loss of generality, we can consider that the points x_1 and x_2 are some distance ϵ from each other. The difference in their combinatorial representation is one of the following cases:

1. Difference in a vertex of P that exists in the combinatorial representation of one, but not the other's.

2. The difference between a base vertex.

3. The changes of the endpoint of a base vertex.

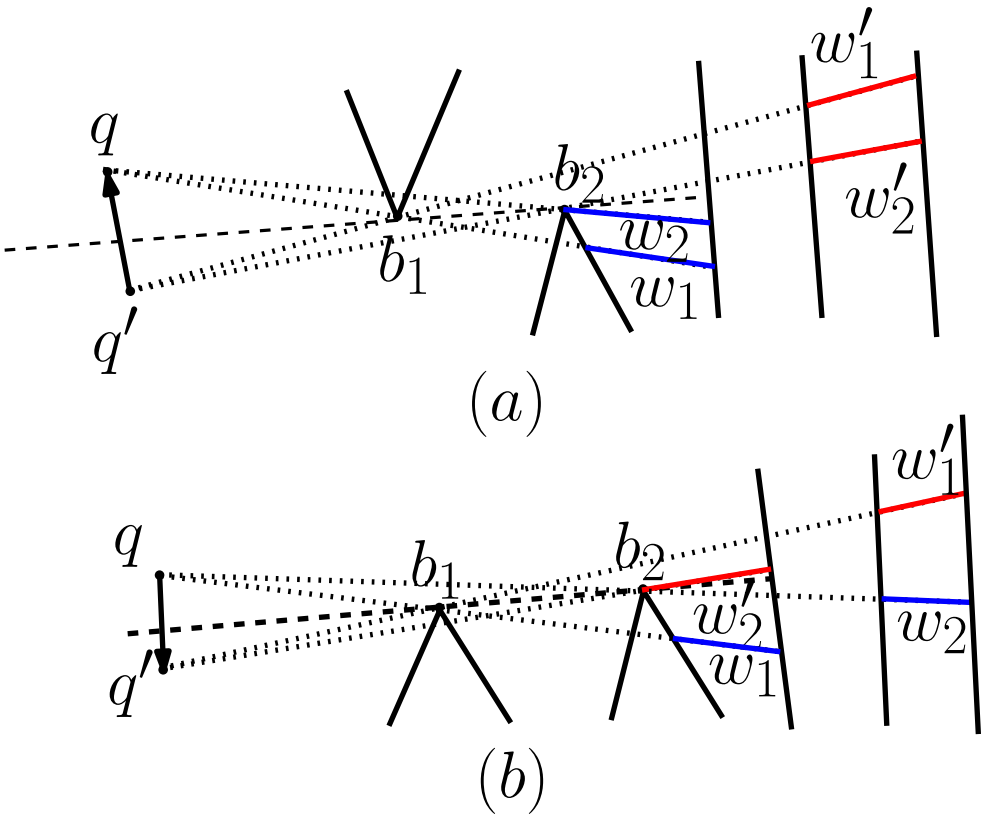


Figure 5.6: When $k = 3$, moving from q to q' changes the angular order of b_1 and b_2. As a result, the position of their equivalent windows change.

4. The changes of the location of a base vertex or the vertex of the polygon P in the list.

We explain each of the above cases in the following.

Suppose x_1 and x_2 lie in the same cell but their combinatorial representation is different at a vertex v_i. If v_i is visible from x_1 but it is not visible from x_2, then the k-visibility region of v_i divides x_1 and x_2 into two different cells. This is a contradiction.

Suppose x_1 and x_2 lie in the same cell but their combinatorial representation is

different at a base vertex b_i. If b_i is a base vertex for x_1, but not a base vertex for x_2, this in turn means that b_i acts as an obstacle of visibility for x_1, but that b_i does not block x_2. So, the v-region of b_i disconnects x_1 and x_2 in the k-cell decomposition. This is a contradiction.

Suppose for a base vertex b_1, that the endpoints of the window created by b_1 lie on different edges for some query points x_1 and x_2. Suppose there exist the same sets of vertices and base vertices appear in the combinatorial representations of x_1 and x_2. This is a valid assumption; if base vertices or vertices are different in the combinatorial representation of x_1 and x_2, then x_1 and x_2 must lie in two different cells as we explained above. Consider the ray x_1b_1. There must exist another base vertex b_2 that as a result of moving from x_1 to x_2, changes the angular order of b_2 and b_1 around the query point. This change introduces or removes an obstacle on the ray x_2b_1, and causes the location of the endpoints of the window created by b_1 from x_2 to change. See Figure 5.6. Consequently, the order line made by b_1 and b_2 must separate x_1 and x_2, a contradiction.

All the vertices of each component of the k-visible region in a given cell are always in order for each cell in the combinatorial representation, as the vertices appear in the same order as on the boundary of the polygon P. If the ordering of two base vertices in a component of the combinatorial representation changes, either their corresponding edges representing the location of the endpoints of their windows change or the edges remain the same. The first case is shown to be a contradiction above. Suppose

the edges representing the endpoints of their corresponding windows stay the same. Notice that while walking along the boundary of a component of the k-visibility region, the windows in this component appear in the angular order that their base vertices appear around the point q. This is because the windows all emanate from the query point q, and pass through the base vertices. The combinatorial representation of the k-visibility region is saved based on the ordering of the elements as they appear on the boundary of the k-visibility region. As moving from x_1 to x_2 causes the angular order of b_1 and b_2 to change around the query point, this in turn causes the order line made by b_1 and b_2 to separate x_1 and x_2, a contradiction. This is to say, all the elements representing windows (base vertices with the edges on which their endpoints lie) in a given cell are always in order for each cell in the combinatorial representation. As the combinatorial representation of the k-visibility region is saved based on the ordering of the elements as they appear on the boundary of the k-visibility region, if the location of a vertex of P changes with the location of a base vertex in the list, the corresponding edges for the base vertex must change as well; shown above to be a contradiction.

Hence, all points in a cell of the k-cell decomposition have the same combinatorial representation. □

By saving the corresponding combinatorial representation for each cell, we show that the k-visibility region of a query point in a given cell can be restored without loss.

Lemma 9. *Given the combinatorial representation of the cell containing the query point q, the k-visibility region of q can be restored in $O(m)$ time, where m is the*

number of vertices of the k-visibility region.

Proof. Looking at the given combinatorial representation, the k-visibility region can be reconstructed as follows. If the current element is a vertex v_i of the polygon P, v_i is reported in the output along with an edge between the previous element of the list (if v_i is not the first element) and v_i. If the current element is a base vertex b_i stored with edges e and e' in the list, the endpoints of its corresponding window can be calculated in constant time considering the intersection points of the ray $\overrightarrow{qb_i}$ and edges e and e'. Let these endpoints be x_i and $x_{i'}$. One of these endpoints must lie on the same edge of P that the previous element of the list lies on; let x_i be that endpoint. The edge between the previous reported element of the list (if b_i is not the first element) and x_i is reported. Following this edge, x_i, $x_ix_{i'}$, and $x_{i'}$ are reported in order.

There are $O(m)$ elements in the combinatorial representation. For each element, the corresponding vertices and edges of the k-visibility region can be found in constant time. So, by knowing the combinatorial representation of the k-visibility region, the region can be retrieved in $O(m)$ time. □

Next, we propose a query algorithm for preprocessing P to find k-visibility region of a given query point.

5.2.1 Preprocessing Steps and the Query Algorithm

In this section, the high-level description of the preprocessing and the query algorithm are presented.

Preprocessing

1. Construct the k-visibility region of each vertex of P.

   - This step takes $O(n \log n)$ time for each vertex [10], resulting in $O(n^2 \log n)$ total time.

2. Construct the v-region of each vertex of the polygon P.

   - This step takes $O(n \log n)$ time for each vertex [55], resulting in $O(n^2 \log n)$ total time.

3. Where possible, construct the order line between each pair of vertices of P.

   - For each pair v_i and v_j, $O(n)$ time is needed to find the intersection of the line segment $v_i v_j$ with the edges of the polygon in one pass over the polygon P. So, in $O(n)$ time it can be found whether v_i and v_j are k-visible from each other. If the point pair is k-visible, it can be determined in constant time if these vertices are critical for each other. Consider the ray $\overrightarrow{v_i v_j}$, the $(k+1)$ intersection points of this ray with the edges of the polygon can be found in $O(n)$ time. Simultaneously, the $(k+1)$ intersection point of the ray $\overrightarrow{v_j v_i}$ with the edges of the polygon P can be found in $O(n)$ time. So, for a pair of vertices of P, it takes $O(n)$ time to determine the order line, if necessary. As there are $O(n^2)$ different pairs of vertices, this step takes $O(n^3)$ time overall.

4. Construct the planar subdivision made by the above decomposition rays and line segments.

- By applying Bentley and Ottmann's algorithm [20], this step can be performed in $O(n^4)$ time as there are $O(n^2)$ rays and line segments and $O(n^4)$ points of intersection in the k-cell decomposition.

5. Process the above subdivision for the point location.

   - This can be done in $O(n^4 \log n)$ time [72; 77; 87; 90].

6. Assign the corresponding combinatorial representation to each cell.

   - Consider an arbitrary query point q in a cell. The k-visibility region of q can be calculated in $O(n \log n)$ time [10]. Notice that in the proposed algorithm by Bahoo *et al.* [10] the boundary of the k-visibility region is reported out of order. This output can be sorted in $O(n \log n)$ time as all the vertices of the k-visibility region are on the boundary of the polygon P given in counterclockwise order.

   - Passing over each component of the k-visibility region of q, if a vertex of P is encountered it will be saved in the corresponding list. If we pass over a window, the base vertex b_i of the window, and the edges of P the endpoints of the window lie on are stored. Notice that the base of each window can be stored when the k-visibility region is found. The size of the k-visibility region is $O(n)$, so storing the combinatorial representation of each cell takes $O(n)$ time.

   - There are $O(n^4)$ cell. As a result, it takes $O(n^5 \log n)$ time to assign the combinatorial representation to the cells of the k-cell decomposition.

As the number of vertices of the k-visibility region is $O(n)$ [9], for each cell $O(n)$ space is needed to save this data. Consequently, $O(n^5)$ space is required to store the entire k-cell decomposition and the information of each cell, and $O(n^5 \log n)$ time for preprocessing.

Query Algorithm

Given a query point q, the k-visibility region of q can be constructed by using the information stored in the preprocessing.

1. Query which cell of the k-cell decomposition q lies in.
    - This step takes $O(\log n)$ time, as the point location query algorithm takes $O(\log n)$ time [48].
2. Retrieve the k-visibility region k.
    - This step takes $O(m)$ time by Lemma 9 where m denotes the number of vertices on the boundary of the k-visibility region.

This data structure and its accompanying query algorithm can be modified to report the k-visibility polygon, the region of the plane which is k-crossing visible for the query point q. Also, this result can be generalized for arbitrary non-crossing line segments instead of a simple polygon. The same approach can be used to decompose the plane. In this case the endpoints of the line segments are processed as the vertices of the polygon and when two line segments have a common endpoint they may act like a critical vertex of the polygon P. As there exist $O(n^2)$ line segments which create the

cell decomposition, there exists $O(n^4)$ cells. As a consequence, $O(n^5)$ space is required for the cell decomposition of the plane with a set of n non-crossing segments, and the k-visibility region of the plane for a query point can be reported in $O(\log n + m)$ time, where m is the size of the output.

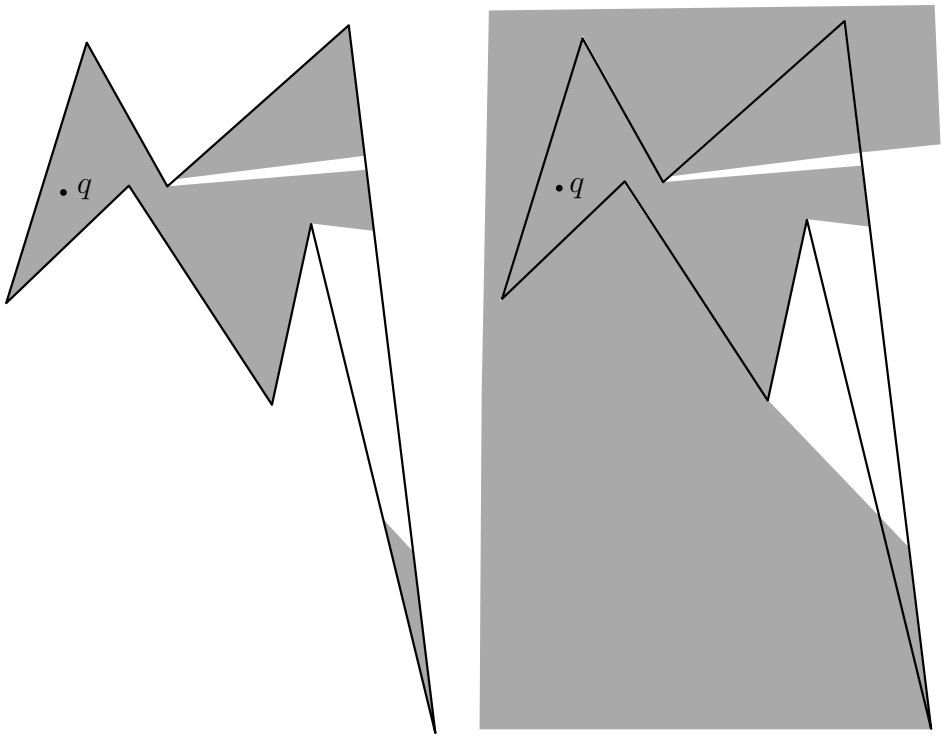
Suppose for each cell in the cell decomposition, the corresponding k-visible regions are stored for all $k \in \{0, \ldots, n\}$. The previous structure for fixed k has size $O(n^5)$. We store, for each value of k, one of these structures. This gives a structure of size $O(n^6)$ that can answer queries for arbitrary k by simply answering the query in the data structure constructed for the given value of k.

Chapter 6

Visibility Query with Constrained Memory

In this chapter, we investigate the problem of computing the k-visible parts of a given simple polygon P from a given query point q under constrained-memory model.

Motivated by the limited resources available to mobile and low-power devices, new categories of algorithms have emerged to address the problems related to these limitations. These algorithms are analyzed under the *limited workspace model* [8]. In this model, there is a read-only memory which stores the input consisting of $O(n)$ words where each word has $\Omega(\log n)$ bits. Memory required for algorithm's computation takes place in separate read-write memory, consisting of $O(s)$ words, where s is a parameter in the limited workspace model. Finally, there is a write-only memory for the purpose of writing the output.



(*a*) (*b*)

Figure 6.1: The 2-visible part of the polygon from q is disconnected (a), while the 2-visible part of the plane in connected, (b).

Given a simple polygon P and a query point q, the goal is to report the parts of P which are k-crossing visible from q, denoted by $V_k(P, q)$. The formal definition of k-crossing visibility and the problem definition are provided in Chapter 2.

Notice that given a simple polygon P and a query point q, the k-visibility region of the plane is always connected, while the k-visibility region of the polygon P from the query point q can be disconnected; see Figure 6.1. The approach proposed in this chapter will work for both the plane and the polygon.

As the properties of 0-visibility and k-visibility regions are quite different, none of the approaches for 0-visibility works for k-crossing visibility.

In the following sections, we show how to calculate the k-visibility region of P from q using $O(s)$ workspace. We present two algorithms, one for $s \in O(1)$, and another parameterized in terms of a general s. The algorithm proposed for constant workspace requires $O(kn+cn)$ time, where c refers to the number of critical vertices. Having $O(s)$ workspace, the other algorithm runs in $O(cn/s + n \log s + \min\{\lceil k/s \rceil\, n, n \log \log_s n\})$ expected time.

In Section 6.4, we will explain how to generalize these ideas to report the k-visibility region from a query point q in a polygon with holes, or in the plane with non-crossing line segments. If q is inside a polygon with holes, the algorithm needs $O(cn/s + n \log s + \min\{\lceil k/s \rceil\, n, n \log \log_s n\})$ expected time when the workspace is of size $O(s)$. If there are n non-crossing line segments in the plane, the k-visibility region from q requires $O(n^2/s + n \log s)$ deterministic time.

6.1 Preliminaries and Definitions

As mentioned, in this chapter, we are studying k-crossing visibility which is defined formally in Chapter 2.

Suppose there exist s words available in the workspace in our model where $s \in \{1, \ldots, n\}$. A simple polygon P which is represented in counterclockwise order is

given. Given a query point q inside P, and a parameter k, the goal is to determine the k-visible region of P from q, denoted by $V_k(P, q)$. As q is inside P, it can be assumed that k is always *even*. When k is odd, the k-visibility region of P corresponds to $(k-1)$-visibility region of P from q. It can also be assumed that the input is given in a *general position*: for any two vertices u and v of P, the three points u, v, and q are not collinear. The boundary of $V_k(P, q)$ includes a subset of the edges, as well as a set of chords of P. A chord is a segment that lies inside P and its endpoints lie on the boundary of P.

Let r_θ denote the ray emanating from q, and let θ be the angle of this ray with respect to the positive x-axis where q is the origin. The *intersecting edge* of r_θ is an edge of P that intersects r_θ. The intersecting edges are stored in the *edge list* of r_θ sorted by their increasing distance from q. Let $e_\theta(j)$ refer to the j^{th} element of this list, where j also denotes the *rank* of $e_\theta(j)$.

The positive angle θ of the ray qv with the positive x-axis is called the *angle* of the vertex v. A vertex v is called a *critical vertex* of q when both edges of v are on the same side of the line determined by qv; v is *non-critical* otherwise. Deciding whether a vertex is a critical vertex for a given q takes $O(1)$ time. If both edges of a critical vertex v lie on the left side of the ray qv, v is called a *start vertex*. If both edges of v lie to the right of the ray qv, v is considered to be an *end vertex*; see Figure 6.2. A minimal continuous set of edges of P with one start vertex and one end vertex at the opposite ends in P is referred to as a *chain*. Each ray r_θ intersects each chain at most once.

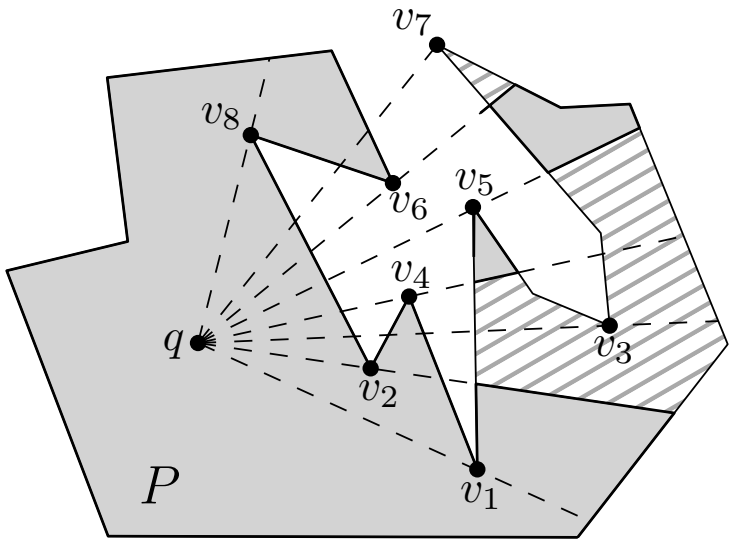


Figure 6.2: An example with $k = 2$. The hatched regions are not 2-visible for q. The vertices $v_1, \ldots, v_8$ are critical for q. More precisely, v_1, v_2, v_3, v_6 are start vertices, and v_4, v_5, v_7, v_8 are end vertices. ∂P is partitioned into 8 disjoint chains, e.g, the counterclockwise chain $v_3 v_5$.

Starting from r_0 and increasing the angle of r_θ continuously, the edge list of r_θ only changes when r_θ intersects with a vertex v of P. If v is a non-critical vertex, the edge of v which lies to the right of r_θ must be removed from the edge list. The other edge of v must be added to the edge list in the same position as the removed edge; the rest of the edge list is unchanged. If v is a critical vertex, there are two cases: either v is a start vertex, or is an end vertex. In the first case, edges of v must be added to the list, while in the latter the edges of v must be removed from it; the rest of the edge list is unchanged. As such, the edge list of r_θ is equivalent to the edge list of $r_{\theta+\epsilon}$ when v is a start vertex, and it is equivalent to the edge list of $r_{\theta-\epsilon}$ when v is an end vertex (ϵ is a small positive number).

When considering the edge list of r_θ, the first $k+1$ intersecting edges of this list are k-crossing visible from the given point q. Increasing r_θ continuously from $\theta = 0$, the chains intersecting r_θ are unchanged until r_θ encounters a k-visible critical vertex. In other words, intersecting chains change when the critical vertex along the ray r_θ is among the first $k+1$ elements of the intersecting edge of r_θ.

Lemma 10. *Let $\theta \in [0, 2\pi)$ such that r_θ contains a k-visible start or end vertex v. The segment on r_θ between $e_\theta(k+2)$ and $e_\theta(k+3)$ is an edge of $V_k(P, q)$, provided that these two edges exist.*

Proof. First, suppose v is a critical end vertex which is k-crossing visible from q. When r_θ intersects with the vertex v, the edges of v must be removed from the edge list of r_θ. The vertex v is k-crossing visible, so its edges must lie among the first $k+2$ elements of the edge list. The k-visibility region on the ray $r_{\theta-\epsilon}$ extends to $e_\theta(k+1)$; though on the ray $r_{\theta+\epsilon}$ the k-visible region extends to $e_\theta(k+3)$. This means that the segment with endpoints $e_\theta(k+2)$ and $e_\theta(k+3)$ must be a part of the boundary of the k-visibility polygon from q. In the case that v is a critical start vertex k-crossing visible from q, the lemma can be proven symmetrically. An example of this case is presented in Figure 6.3.

□

Let r_θ intersect a critical vertex which is either a start or end vertex. By Lemma 10, the segment with endpoints $e_\theta(k+2)$ and $e_\theta(k+3)$ is part of the boundary of $\partial V_k(P, q)$. Notice that this segment is not on the boundary of P. Such a segment is referred to as a *window*; see Figure 6.3.

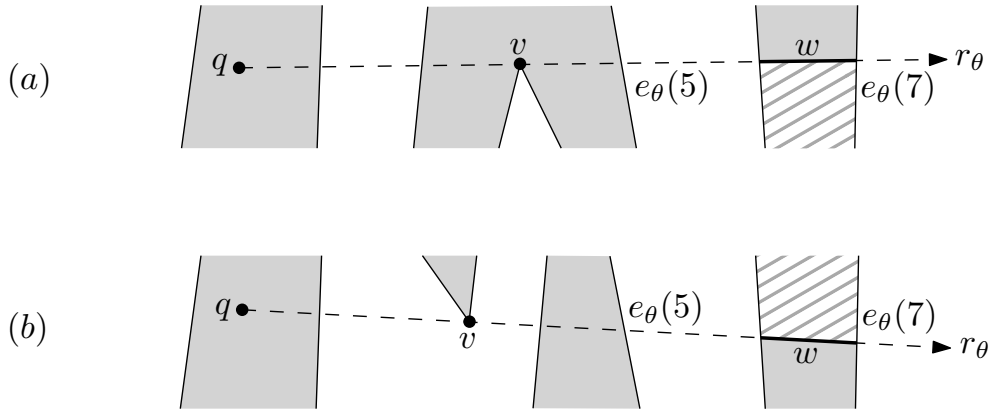


Figure 6.3: An example with $k = 4$. The hatched regions are not 4-visible for q. (a) The ray r_θ encounters the end vertex v. The 4-visibility region of q before v extends until $e_\theta(5)$ and after v extends until $e_\theta(7)$. (b) The ray r_θ encounters the start vertex v. The 4-visibility region of q before v extends until $e_\theta(7)$ and after v extends until $e_\theta(5)$. The segment w in both figures is the window of r_θ.

Observation 1. *The k-visibility region $V_k(P, q)$ has $O(n)$ vertices.*

Proof. $\partial V_k(P, q)$ consists of windows, and some subset of ∂P. As a result, a vertex of $\partial V_k(P, q)$ is either a vertex of P or an endpoint of a window. Each window has two endpoints, and lies on a ray emanating through q passing a critical vertex; see Lemma 10. Critical vertices are a subset of vertices of the polygon, there exist $O(n)$ critical vertices. Hence, the total number of vertices of $\partial V_k(P, q)$ is $O(n)$.

□

6.2 An Algorithm Using O(1) Words

In this section we present an algorithm that computes the k-visibility region of a given vertex q in a given polygon P when the algorithm's workspace is limited to $O(1)$ words each of length $\Omega(\log n)$. When there does not exist any critical vertex of P for the point q, $\partial V_k(P, q) = P$, there also exists no window. Checking whether

there exists a critical vertex for q takes $O(n)$ time. Let us assume there is at least one critical vertex, called v_0. With a scan on the vertices of P, v_0 can be found in $O(n)$ time using $O(1)$ words of memory. Next, we consider the ray qv_0 while q is the origin. Let $v_0, v_1, \ldots, v_{c-1}$ refer to the critical vertices in the order they are encountered by r_θ in counterclockwise order. In this section the notations r_i and $e_i(j)$ will refer to r_{θ_i} and $e_{r_{\theta_i}}(j)$ respectively, where θ_i refers to the angle of v_i.

Considering the ray r_0, the edge $e_0(k+1)$ can be found in $O(nk)$ time using $O(1)$ workspace perfoming a *selection subroutine* as follows: scan the input $k+1$ times, and at each iteration find the next intersecting edge with r_θ until $e_0(k+1)$ is reached. If v_0 is encountered among the first $k+1$ intersection, v_0 is k-crossing visible. As a result, the window which lies on r_0 , if it exists, must be reported based on Lemma 10. The window is the segment $e_0(k+2)e_0(k+3)$ which can be found by two more scans. This is because $e_0(k+2)$ and $e_0(k+3)$ can be found by two more scans as explained above.

In the next step, v_1 can be found in $O(n)$ time by a simple scan. Determining $e_1(k+1)$ can be found in $O(n)$ time by using $e_0(k+1)$ as follows: If v_0 is an end vertex, then the edges of v_0 vanish in the edge list of r_1. When v_1 is a start vertex, its edges are added to the edge list of r_1; the rest of the chains are unaffected and remain unchanged. $e_{\theta_0+\epsilon}(k+1)$ is either $e_0(k+1)$ or $e_0(k+3)$ based on the type and position of v_0, and can therefore be found in $O(n)$ time. Let this edge be referred to as e'. Scan along the boundary of the polygon P from e' until reaching $r_{\theta_1-\epsilon}$. This

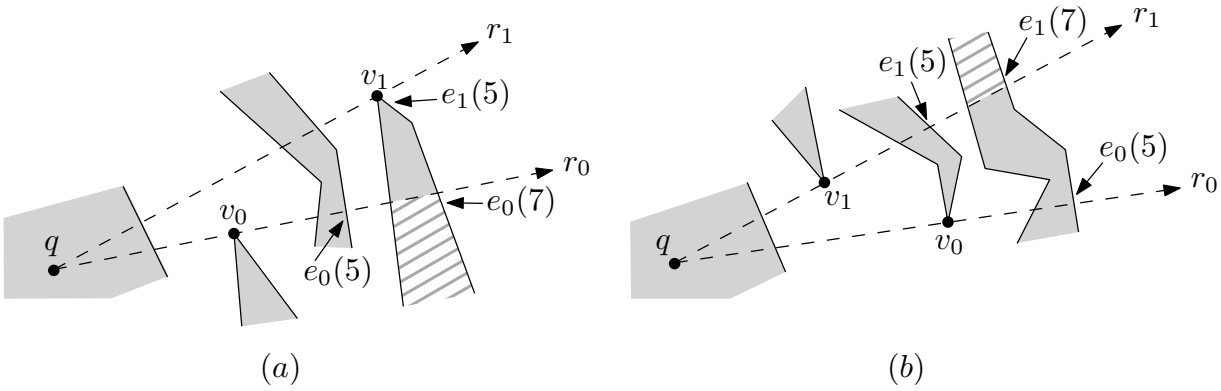


Figure 6.4: For the above examples, let $k = 4$. (a) Both v_0 and v_1 are end vertices. $e_0(5)$ is used to find $e_0(7)$ and follow the chain until $e_1(5)$. (b) Both v_0 and v_1 are start vertices. The chain of $e_0(5)$ can be followed until $e_1(7)$, which is then used to find $e_1(5)$. Finally, the window from $e_1(6)$ to $e_1(7)$ is reported.

can be found in $O(n)$ time. Denote by e'' the last edge encountered on this scan, and note that e'' belongs to the same chain as e'. Based on the type and position of v_1, e'' is either $e_1(k+1)$ or $e_1(k+3)$. As a consequence, $e_1(k+1)$ can be determined in $O(n)$ time by using e''. An example of this process is given in Figure 6.4.

If v_1 is k-crossing visible from q, there exists a window on r_1 which can be reported in $O(n)$ time by the above process. The subchains of $\partial V_k(P, q)$ between r_0 and r_1 must also be reported. This can be done by scanning ∂P by first traversing along the boundary of P counterclockwise. When entering the counterclockwise cone between r_1 and r_0, it must be checked whether the intersection of ∂P and r_1 or r_0 takes place at or before $e_1(k+1)$ or $e_0(k+1)$, respectively. If this is the case, the subchain of ∂P must be reported until the cone is exited.

This process is repeated until all critical vertices are encountered. The full pro-

Algorithm 6.2.1: The constant workspace algorithm for computing $V_k(P, q)$

input: Simple polygon P, point $q \in P$, $k \in \mathbb{N}$

output: The boundary of the k-visibility region of q in P, $\partial V_k(P, q)$

```
1  if P has no critical vertex then
2  |  return ∂P
3  v_0 ← a critical vertex of P
4  Find e_0(k + 1) using selection
5  i ← 0
6  repeat
7  |  if v_i lies on or before e_i(k + 1) on r_i then
8  |  |  Report the window of r_i (if it exists)
9  |  v_{i+1} ← the next counterclockwise critical vertex after v_i
10 |  Find e_{i+1}(k + 1) using e_i(k + 1)
11 |  Report the part of ∂V_k(P, q) between r_i and r_{i+1}
12 |  i ← i + 1
13 until v_i = v_0
```

cess is represented in Algorithm 6.2.1. In Algorithm 6.2.1, if there are less than $k+1$ elements in the edge list of r_i, the last edge in that list and its rank will be used in place of $e_i(k+1)$, to find $e_{i+1}(k+1)$ or the last element in the edge list of r_{i+1} and its rank. As there are c critical vertices which take $O(n)$ time to process, and processing v_0 takes $O(kn)$ time, the following theorem can be derived:

Theorem 2. *Given a simple polygon P with n vertices, a point $q \in P$, and a parameter $k \in \{0, \ldots, n-1\}$, the k-visibility region of q in P can be reported in $O(kn+cn)$ time using $O(1)$ words of workspace, where c is the number of critical vertices in P.*

6.3 Time-Space Trade-Offs

In this section we present an algorithm that computes the k-visibility region of a given vertex q in a given polygon P when the algorithm's workspace is limited to $O(s)$ words of length $\Omega(\log n)$ each, for a fixed $s \in [1, n]$. Using this workspace, there are faster algorithms than the previous one which reports the k-visibility region. First, we propose a simple algorithm that includes the main idea behind the trade-off. The second algorithm is more complicated, but with better time complexity. In the first algorithm, vertices are processed in angular order in continuous batches of size s. By using the edge list of the last processed vertex, a data structure can be created with which the windows of the batch can be reported. From the windows of each batch, $\partial V_k(P, q)$ can be output between the first and last rays, per batch. It can be noted that the edges of the k-visibility polygon are not reported in order, but all edges are reported in the algorithm. In the second approach, the running time is improved by focusing on critical vertices instead of processing all vertices. We process a batch of s critical vertices in each iteration. A data structure will be constructed in order to find the windows, though a more involved approach is needed to maintain this data structure. In the following Lemma, an efficient way to find the continuous batches of vertices in the angular order is presented. This procedure is based on Theorem 2.1 of Chan and Chen's work [32].

Lemma 11. *Suppose that we are given a read-only array A with n pairwise distinct elements from a totally ordered universe and an element $x \in A$. For any given parameter $s \in \{1, \ldots, n\}$, there is an algorithm that runs in $O(n)$ time and uses $O(s)$*

words of workspace that finds the set of the first s elements in A that follow x in the sorted order. This is applicable for finding the last s elements which appear before x in the sorted order.

Proof. Let $A_{>x}$ denote a subset of A which contains the elements of A larger than x. Notice that there is no need to calculate $A_{>x}$. We pass over A and skip the elements less or equal than x which corresponds to $A_{>x}$. The algorithm makes one pass over $A_{>x}$, where the elements are processed in batches as follows: the first $2s$ elements of $A_{>x}$ are stored in the workspace (without sorting). The median of these $2s$ elements can be found in $O(s)$ time by using $O(s)$ workspace [45]. After this, the s elements which are greater than the median are removed from the workspace using $O(s)$ time and space. The next s elements of $A_{>x}$ are then inserted into the workspace, and the same process is applied: the median of the $2s$ elements saved in the workspace is found and elements greater than the median are removed. This process is repeated until all of $A_{>x}$ is processed. This proccess results in the smallest elements of $A_{>x}$ remaining in the workspace. The above process is performed $O(n/s)$ times, where each iteration takes $O(s)$ time and workspace. Hence, this algorithm runs in $O(n)$ time, using $O(s)$ workspace. □

Lemma 12. *Suppose a read-only array A with n elements is given from a totally ordered universe and a number $k \in \{1, \ldots, n-1\}$. For any given parameter $s \in \{1, \ldots, n\}$, there is an algorithm that runs in $O\big(\lceil k/s \rceil n\big)$ time and uses $O(s)$ words of workspace and that finds the k^{th} smallest element in A.*

Proof. First, the first s smallest elements of A are found. This can be done in $O(n)$ time and $O(s)$ workspace by Lemma 11. If $k \leq s$, the k^{th} smallest element stored

in the workspace can be found in $O(s)$ time. Otherwise, we proceed as follows: the largest element in the workspace can be found in $O(s)$ time; let it be called x. Using Lemma 11, the first s elements greater than x in A can be determined. This process continues such that in step i, the i^{th} batch of s elements is found. If $k \leq s$, the $(k-(i-1)s)^{th}$ smallest element in the workspace is the output which can be found in $O(s)$ time. Otherwise, the largest element in the workspace is found and the process continues. The result will be found in the $\lceil k/s \rceil^{th}$ batch. So, the run time of the algorithm is $O(\lceil k/s \rceil\, n)$. □

There exist other selection algorithms in the read-only memory rather than the simple algorithm in Lemma 12; see Table 1 of [35]. Specifically, there is a randomized algorithm for selection which uses $O(s)$ words of the workspace with $O(n \log\log_s n)$ expected time [31; 83]. We must choose between the latter algorithm and the algorithm represented in Lemma 12 according to given k, s and n. The selection subroutine in this text refers to the chosen selection algorithm chosen. In this work, we represent the selection time by $T_{selection}$ which refers to $O(\min\{\lceil k/s \rceil\, n, n \log\log_s n\})$ expected time.

6.3.1 Processing All the Vertices

The process begins by first considering the ray emanating from q and parallel to the positive x-axis. By Lemma 11, the batch that consists of s vertices with the smallest angles can be found with $O(s)$ words of memory; and can be sorted in $O(s \log s)$ time. Let $v_0, v_1, v_2, \ldots, v_s$ be the vertices of P in the sorted order. By using the selection subroutine, $e_0(k+1)$ can be found. If v_0 is not after $e_0(k+1)$ on r_0, v_0 is

k-visible from q, and the appropriate window can be reported, if it exists. It should be noted that in case that there are less than $k+1$ intersecting edges on r_0, the last intersecting edge and its rank will be stored.

By applying Lemma 11 four times consecutively, $4s+1$ intersecting edges with rank $k-2s+1, \ldots, k+2s+1$ can be found. Notice that Lemma 11 can be applied as the edge $e_0(k+1)$ is already calculated. These edges are added to a balanced binary search tree T, sorted based on their rank on r_0. These stored edges are the candidates for $e_i(k+1)$ where $i \in \{1, \ldots, s\}$ (candidates for having the rank $k+1$ on the next s rays). As mentioned, if $e_i(k+1)$ is in the edge list of r_{i-1}, the edge list of r_{i-1} contains at most one edge between $e_{i-1}(k+1)$ and $e_i(k+1)$. So, in the case where $e_i(k+1)$ exists in the edge list of r_0, at most $2i-1$ edges can occur between $e_0(k+1)$ and $e_i(k+1)$ in the edge list of r_0. This is because for each critical vertex between r_0 and r_i two additional edges may be added to the edge list.

The algorithm then proceeds as follows: the next vertex v_1 will be processed and the tree T updated according to the type of v_0 and v_1. If v_0 is a non-critical vertex, one edge of v_0 may be replaced with another in T. If v_0 is an end vertex, we may remove its edges from T. Finally, if v_1 is a start vertex, its edges may inserted to T. For other cases, no action is needed. The insertion and deletion happen only for edges with a rank between the smallest and largest rank in T, with respect to r_1. Each update of the tree needs $O(\log s)$ time. After this update on T, $e_1(k+1)$, if it exists, can be reported in $O(1)$ time by using the position of $e_0(k+1)$ and its

neighbor in T. Based on the type of the vertex v_1, we need only to examine a constant number of neighbours around $e_0(k+1)$. An example of this is presented in Figure 6.5.

This procedure is repeated for $v_2, \ldots, v_s$. To determine $e_i(k+1)$ and the window on r_i for $i \in \{2, \ldots, s\}$, the tree T and $e_{i-1}(k+1)$ are used. This process uses $O(s \log s)$ time. When a window is discovered, its end-points will be inserted into another balanced binary search tree called W, requiring $O(\log s)$ time per window. Notice that the endpoints in W are sorted based on their counterclockwise order around q along ∂P. The part of $\partial V_k(P, q)$ which lies between r_0 and r_s consists of W and $e_i(k+1)$ for $i \in \{0, \ldots, s\}$. The set of these $e_i(k+1)$ for $i \in \{0, \ldots, s\}$ is called E.

Next we show how to report the k-visible part of the boundary of P in $O(n)$ time. Let a *0s-segment* of an edge e of P be the subset of e between r_0 and r_s. If the $0s$-segment does not include an end-point of a window, it must be either completely k-visible or not k-visible at all which can be found as follows: each endpoint of $0s$-segment can be checked in $O(1)$ time to determine whether it is k-visible; this can be done by using E. Also, by traversing W in parallel, it can determine whether there exists an end-point of a window which lies on e. This part can be done in $O(|w_e|)$ time, where $|w_e|$ refers to the number of end-points of windows which lie on e. This information is sufficient to report the k-visible part of a $0s$-segment. Since there exist $O(n)$ windows (Observation 1) and each one processed once, the k-visible part of ∂P between r_0 and r_s can be reported in $O(n)$ time.

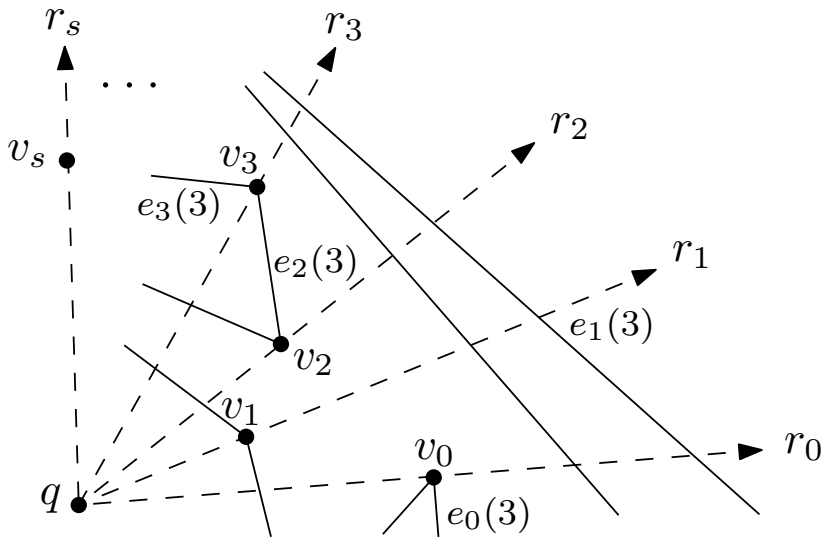


Figure 6.5: The first batch $v_0, v_1, \ldots, v_s$ of s vertices in angular order. The edge $e_1(3)$ is the second neighbor to the right of $e_0(3)$ on r_0, because v_0 is an end vertex. The edge $e_2(3)$ is the second neighbor to the left of $e_1(3)$ which is inserted in T before processing v_2. The edge $e_2(3)$ is exchanged with $e_3(3)$, after processing v_3, because v_3 is a non-critical vertex.

After processing $\{v_0, \ldots, v_s\}$, the next batch of s vertices after v_s is calculated according to their angular order. The same process as before will be applied in order to report the k-visible part of ∂P between $r_s, \ldots, r_{2s}$ where v_{2s} is the last vertex in the batch. Notice that the binary search tree constructed from the previous batch is out of use for the new batch as it does not necessarily include any right or left neighbour of $e_s(k+1)$ on r_s; consequently, a new binary search tree must be constructed. This will continue until all vertices are processed. Also, it should be noted that the last batch may include fewer than s vertices as n may not be fully divisible by s. The full procedure for all the vertices is given in Algorithm 6.3.1.

The process is repeated for $O(n/s)$ batches, where each batch takes $O(n + s \log s)$ time. Also, the selection subroutine runs for the first batch. Consequently, the algo-

Algorithm 6.3.1: Computing $\partial V_k(P, q)$ using $O(s)$ words of workspace

input: Simple polygon P, point $q \in P$, $k \in \mathbb{N}$, $1 \leq s \leq n$

output: The boundary of k-visibility region of q in P, $\partial V_k(P, q)$

```
1  v_0 ← a vertex of P
2  E ← ⟨e_0(k+1)⟩ (using the selection subroutine with O(s) workspace)
3  T, W ← an empty balanced binary search tree
4  i ← 0
5  repeat
6      v_{i+1}, ..., v_{i+s} ← sorted list of s vertices following v_i in angular order
7      T ← at most 4s+1 edges with rank in {k−2s+1, ..., k+2s+1} on r_i
8      for j = i to i+s−1 do
9          if v_j lies on or before e_j(k+1) on r_j then
10             Report the window of r_j (if it exists)
11             Insert the endpoints of the window into W (according to their
                position on ∂P)
12         Update T according to the types of v_j and v_{j+1}
13         E.append(e_{j+1}(k+1)) (find it using e_j(k+1) and T)
14     Report the part of ∂V_k(P, q) between r_i and r_{min{i+s,n}} (using W and E)
15     i ← i+s
16 until i ≥ n
```

rithm runs in $O((n/s)(n + s \log s)) + T_{selection}$. As $T_{selection}$ is dominated by the other terms, the following theorem holds.

Theorem 3. *Let $s \in \{1, \ldots, n\}$. Given a simple polygon P with n vertices in a read-only array, a point $q \in P$ and a parameter $k \in \{0, \ldots, n-1\}$, the k-visibility region of q in P can be reported in $O(n^2/s + n \log s)$ time using $O(s)$ words of workspace.*

6.3.2 Processing Only the Critical Vertices

In this section, we present an algorithm that processes only the critical vertices instead of all vertices. The algorithm is similar to the algorithm in the previous section, except that the intersecting edges will be handled differently. In each iteration, the next batch of s critical vertices is found and is sorted in $O(s \log s)$ while using $O(s)$ words of the workspace. As before, a balanced binary search tree T is constructed which includes the possible candidates for the $(k+1)^{th}$ intersecting edge that lies on the rays emanating from q and passing the s critical vertices of each batch. Then, the next critical vertex will be processed at each step. The tree T will be used to calculate the windows, and updated as necessary. After all the windows in a batch are reported, the k-visible part of ∂P between the first and last rays of the batch can be found. Notice that T can be updated efficiently by using an auxiliary data structure called T_{aux} which is explained below.

If P does not have any critical vertices, P is completely k-visible from q. This can be determined by scanning P once in $O(n)$ time. So, let v_0 be a critical vertex. The coordinate system is chosen so that P the ray qv_0 lies on the positive x-axis. First, the first critical vertices $v_1, \ldots, v_s$ which occur after v_0 are calculated and sorted in angular order. This process can be done in $O(n + s \log s)$ time using $O(s)$ words of workspace, by Lemma 11. Then, $e_0(k+1)$ can be calculated through the selection subroutine. Additionally, $4s+1$ intersecting edges $\{k-2s+1, \ldots, k+2s+1\}$ on r_0 are found if they exist. These intersecting edges will be inserted into the binary search tree T, where they are ordered based on their rank on r_0. This process up to now can

be done in $T_{selection}+O(n+s\log s)$ time. Each edge e of T is then checked to determine if it has a non-critical endpoint that lies between r_0 to r_s. The corresponding edges of such endpoints will be added to another balanced binary search tree called T_{aux} based on their angular order. Each member of T_{aux} has a cross-pointer to its equivalent edge in T. The tree T_{aux} can be constructed in $O(s\log s)$ time as T has $O(s)$ members by using $O(s)$ words of workspace. T_{aux} will be used to determine which edges in T must be updated between two critical vertices. This is shown in Figure 6.6.

In order to find $e_1(k+1)$, the next step is to update the tree T such that it contains the edge list on r_1. The update process happens as follows: for each non-critical vertex v stored in T_{aux} which is between the rays r_0 and r_1, we walk on the chain C to which v belongs until the edge e of C intersecting r_1 is visited. Notice that such an edge e exists as there is no critical vertex between r_0 and r_1 that can be the endpoint of C. If the endpoint of e which lies after r_1 is non-critical, it will be added to T_{aux}, and the corresponding edge of v in T will be replaced by e. This process can be done in $O(n_1\log s+n_1)$ time where n_1 is the number of non-critical vertices between r_0 and r_1. The trees T and T_{aux} must be updated accordingly based on the vertex type of v_0 and v_1. In case v_0 is an end vertex, the two edges incident to v_0 must be removed from T. If v_1 is a start vertex, its two edges must be added to T. This update takes $O(\log s)$ time. After doing this process, T includes at most $4s+1$ intersecting edges on the ray r_1. Using the chain of $e_0(k+1)$ and its neighbours in T, $e_1(k+1)$ can be found in $O(1)$ time as we just need to check constant number of neighbors of $e_0(k+1)$ in the tree. This process will be repeated for all the critical vertices in the

batch. As a result, updating T for a batch (for critical and non-critical vertices) takes $O(n' \log s + n' + s \log s)$ time where n' refers to the number of non-critical vertices between r_0 and r_s.

When processing a batch, all $e_i(k+1)$ will be stored in E; as before, E is the sequence $e_0(k+1), e_1(k+1), \ldots, e_s(k+1)$ of the edges of rank $k+1$. Additionally, when a window is discovered, its endpoint will be added to the binary search tree W, sorted based on their angular order around q in $O(\log s)$ time. After processing the entire batch, E and W can be used to report all parts of ∂P which are k-visible from q that lie between r_0 and r_s. The reporting process is the same as that in Section 6.3.1, though here we must keep track of the entire chains between r_0 and r_s instead of single edges. This takes $O(n)$ time.

In the next iteration, the following batch of s critical vertices will be repeated. This process will be repeated until all critical vertices are processed; see Algorithm 6.3.2. Notice that each non-critical vertex is handled in exactly one iteration. As there exist $O(c/s)$ iterations, and the update of T takes $O(n \log s)$ time, the overall time complexity of the algorithm is $O(cn/s + n \log s)$ plus $T_{selection}$, which runs for the first batch. As a result, we have the following Theorem.

Theorem 4. *Let $s \in \{1, \ldots, n\}$. Given a simple polygon P with n vertices in a read-only array, a point $q \in P$ and a parameter $k \in \{0, \ldots, n-1\}$, the k-visibility region of q in P can be reported in $O(cn/s + n \log s + \min\{\lceil k/s \rceil n, n \log \log_s n\})$ expected time*

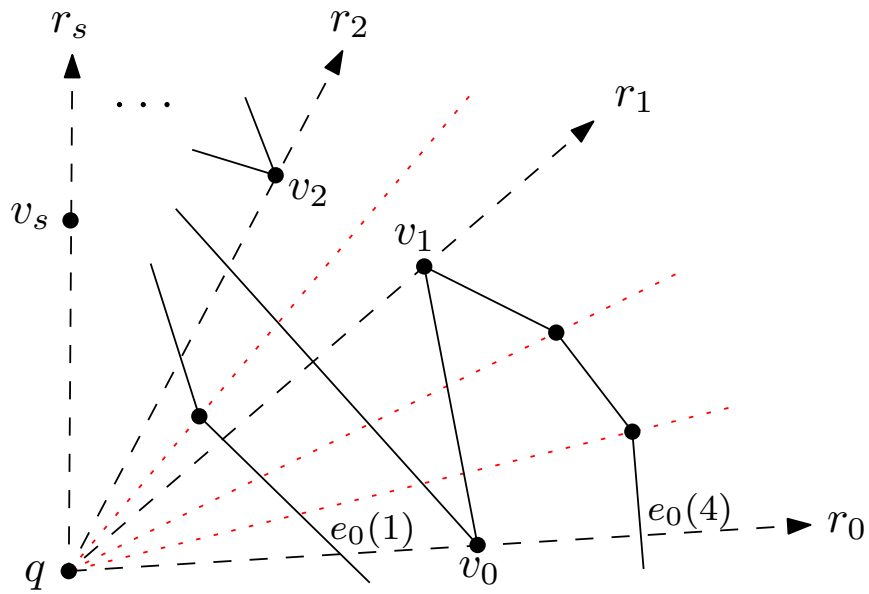


Figure 6.6: The first batch $v_0, v_1, \ldots, v_s$ of s critical vertices in angular order. The non-critical endpoint of $e_0(1)$ is between r_1 and r_2, so $e_0(1)$ will be replaced in T right before processing v_2. The non-critical endpoint of $e_0(4)$ is between r_0 and r_1, so $e_0(4)$ will be replaced in T right before processing v_1.

using $O(s)$ words of workspace, where c is the number of critical vertices of P for q.

6.4 Variants and Extensions

The results presented in this chapter can be expanded in different ways, such as calculating the k-visibility region inside a polygon P when P may include holes, or calculating the k-visible region for a point q which lies in the planar arrangement of non-crossing line segments inside a bounding box (this bounding box is only used to bound the k-visible region). In the case of a polygon with holes, all previous properties for a simple polygon will be applied except the use of ∂P for reporting the k-visible segment of ∂P. For a polygon with holes, after processing the outer part of ∂P, the boundary of each hole will be processed sequentially. The same procedure

Algorithm 6.3.2: Computing $\partial V_k(P, q)$ using $O(s)$ words of workspace

input: Simple polygon P, point $q \in P$, $k \in \mathbb{N}$, $1 \leq s \leq n$

output: The boundary of k-visibility region of q in P, $\partial V_k(P, q)$

$v_0 \leftarrow$ a critical vertex of P

$E \leftarrow \langle e_0(k+1) \rangle$ (using the selection subroutine with $O(s)$ workspace)

$T, T_{\text{aux}}, W \leftarrow$ an empty balanced binary search tree

$i \leftarrow 0$

repeat

  $v_{i+1}, \ldots, v_{i+s} \leftarrow$ sorted list of s critical vertices following v_i in angular order

  $T \leftarrow$ at most $4s+1$ edges with rank in $\{k-2s+1, \ldots, k+2s+1\}$ on r_i

  $T_{\text{aux}} \leftarrow$ for each edge in T, its non-critical endpoint between r_i and r_{i+s} (if it exists)

  for $j = i$ *to* $i+s-1$ **do**

    if v_j *lies on or before* $e_j(k+1)$ *on* r_j **then**

      Report the window of r_j (if it exists)

      Insert the endpoints of the window into W (according to their position on ∂P)

    for *any* $v \in T_{aux}$ *between* r_j *and* r_{j+1} **do**

      Find the edge e on v's chain that intersects r_{j+1}

      Exchange the corresponding edge of v in T with e

      If e has a non-critical endpoint between r_{j+1} and r_{i+s}, insert it into T_{aux}

    Update T according to the types of v_j and v_{j+1}

    E.append($e_{j+1}(k+1)$) (find it using $e_j(k+1)$ and T)

  Report the part of $\partial V_k(P, q)$ between r_i and $r_{\min\{i+s,n\}}$ (using W and E)

  $i \leftarrow i+s$

until $i \geq n$

will be applied while walking on the boundary of each hole. Notice that if there does not exist any window on the boundary of a hole, it is either completely k-visible or it is completely not k-visible. Each hole is checked to determine if it is k-visible, and holes that are k-visible will be reported completely. So, the following corollary holds.

Corollary 1. *Let $s \in \{1, \ldots, n\}$. Given a polygon P with $h \geq 0$ holes and n vertices in a read-only array, a point $q \in P$ and a parameter $k \in \{0, \ldots, n-1\}$, the k-visibility region of q in P can be reported in $O(cn/s + n \log s + \min\{\lceil k/s \rceil n, n \log \log_s n\})$ expected time using $O(s)$ words of workspace. Here, c is the number of critical vertices of P for the point q.*

For the case that the query point q is in a planar arrangement of non-crossing segments inside a bounding box, the output consists of the parts of the segments which are k-visible from q. Notice that all endpoints of the segments are critical and must be processed. Instead of performing a walk on the boundary of the polygon, we apply a sequential scan of the input, as this outputs a similar result. Also, there may exist some segments such that there is no window endpoints on them. For such a case, it can be determined whether the segment is either completely k-visible or completely not k-visible simply by checking the visibility of an endpoint. So, we have the following corollary.

Corollary 2. *Let $s \in \{1, \ldots, n\}$. Given a set S of n non-crossing planar segments in a read-only array that lie in a bounding box B, a point $q \in B$ and a parameter $k \in \{0, \ldots, n-1\}$, there is an algorithm that reports the k-visible subsets of segments*

in S from q in $O(n^2/s + n \log s)$ time using $O(s)$ words of workspace.

Notice that the same approach can be applied to report the slightly different definition of k-visibility proposed by Bajuelos *et al.* [12]. As a consequence, the running time of the algorithm of Bajuelos *et al.* [12] can be reduced from $O(n^2)$ time to $O(n \log n)$ time by applying an algorithm analogous to that described in this section, using $O(n)$ words of workspace.

Chapter 7

Watchtower

Given a 1.5D terrain T, consisting of an x-monotone polygonal chain with n vertices in the plane, and a positive integer k, we propose an algorithm to place one point, called a *watchtower*, whose vertical height above T is minimized, such that every point x on T is k-crossing visible from the watchtower w. That is, the line segment from w to any point x on T crosses T at most k times. Our algorithm runs in $O((n^2 + m) \log n)$ time, where m denotes the number of vertices on the boundary of the k-kernel of T. For arbitrary k, $m \in O(n^4)$, and for $k = 2$, $m \in O(n^2)$. When the watchtower is restricted to being positioned over a vertex of T, we can improve the time complexity. We show this improvement step by step in this chapter. We start from the $O(n^4)$-time algorithm and end with an $O(n^3)$-time algorithm.

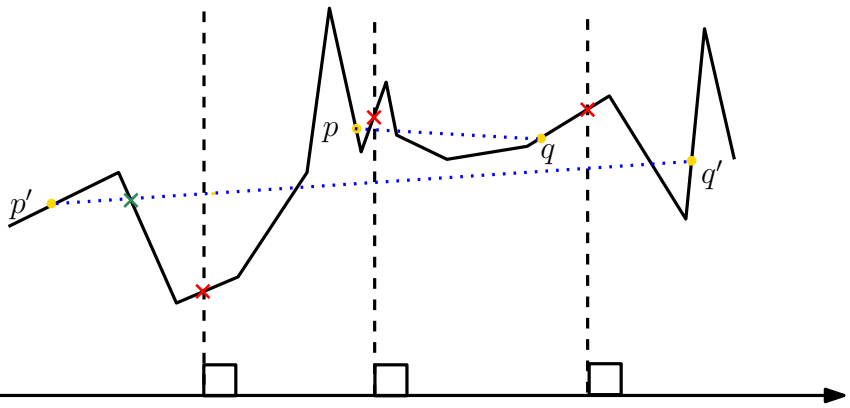


Figure 7.1: The points p and q mutually 2-crossing visible, while p' and q' are not.

7.1 Introduction

A *terrain* T in $\mathbb{R}^2$ is an x-monotone polygonal chain consisting of a sequence of vertices $v_0, v_1, \ldots, v_{n-1}$, each of which is a point in $\mathbb{R}^2$, such that v_i is to the left of v_j for all $i < j$ and $v_i v_{i+1}$ is an edge for $i \in 0, \ldots, n-2$. See Figure 7.1. Two points p and q are *k-crossing visible* if and only if the line segment pq crosses T at most k times. For a formal definition of k-crossing visibility see Chapter 2.

A *watchtower* w is a point on or above T. Given a terrain T and a positive integer k, the goal in the *1-watchtower* problem is to place a watchtower w with minimum height on or above T (length of the vertical line segment from w to T) such that the entire terrain T is k-crossing visible from w. This definition can be generalized to the *M-watchtower* problem where the goal is to assign positions to a set $W = \{w_1, \ldots, w_M\}$ of M watchtowers, such that each w_i is a point on or above T, and for each point p on T, there exists a watchtower $w \in W$ such that p is k-crossing visible from w.

The watchtower problem presents itself in two forms: discrete and continuous. In the discrete version, the watchtower must be located on a vertical line through a vertex of the terrain, while in the continuous version the watchtower can be located anywhere above the terrain. Solutions to the discrete and continuous watchtower problems can vary significantly. Figure 2 shows an instance for which the solution to the continuous 1-watchtower problem has height zero (on the terrain), whereas the solution to the discrete 1-watchtower problem on the same terrain requires a watchtower to be positioned significantly higher.

The watchtower problem generalizes to the setting of k-crossing visibility for any k. We consider the problem of placing one watchtower. In Section 7.2, we present an algorithm for the continuous problem, and then propose an algorithm for the discrete problem in Section 7.3. For both algorithms we describe how the running time can be decreased when $k = 2$ and $k = 0$.

7.2 k-Kernel Algorithm

In this section, we solve the continuous 1-watchtower problem under k-visibility for general k, and then describe how the running time can be reduced when $k = 2$ and $k = 0$.

Consider a simple polygon T', bounded from above by a horizontal line segment h_P that lies above T, and on its sides by vertical line segments aligned with the respective left and right endpoints of T; see Figure 7.3. Since the watchtower must be

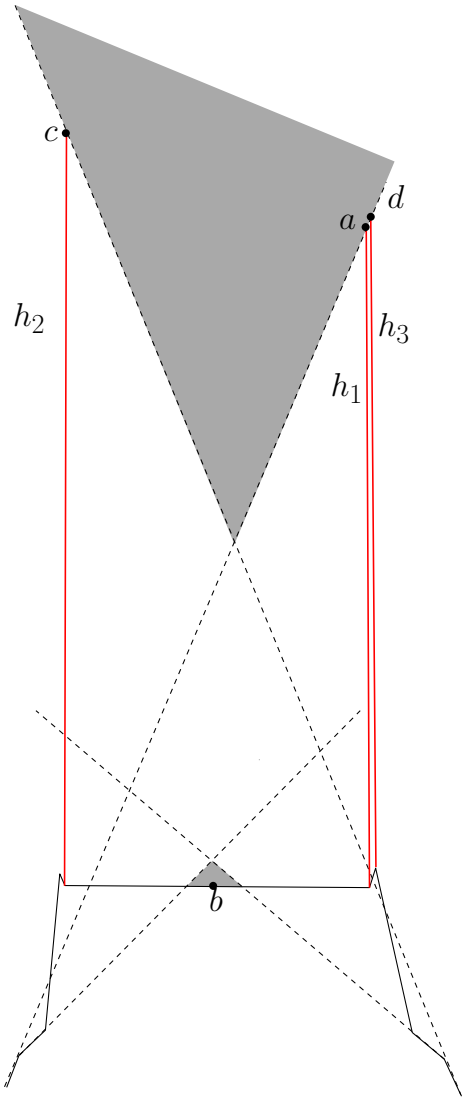


Figure 7.2: When $k = 2$, the solution to the 1-watchtower problem for the continuous problem can have much lower height than that for the discrete problem on the same terrain. The points b and d represent the locations of the watchtower in the continuous and discrete versions, respectively (suppose $h_3 < h_1$). In the continuous version, the tower is located on the edge of the terrain with height zero, while in the discrete version it must be located above the terrain with height h_3, significantly larger than zero. Notice that the points below d cannot see the edges adjacent to the vertex v.

located above the terrain, it must be inside T'.

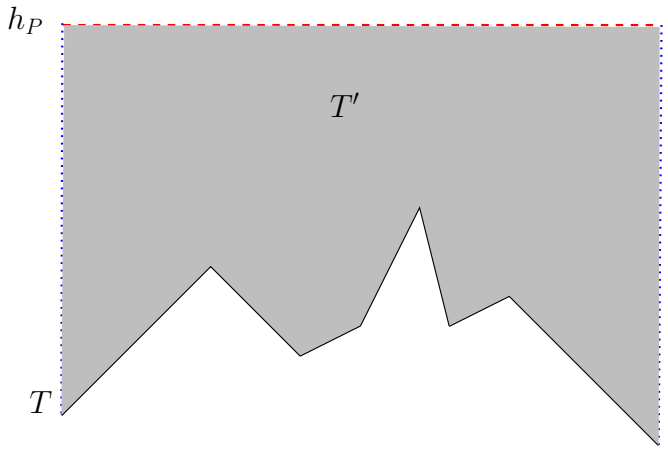


Figure 7.3: The shaded region is a simple polygon P constructed for a given terrain.

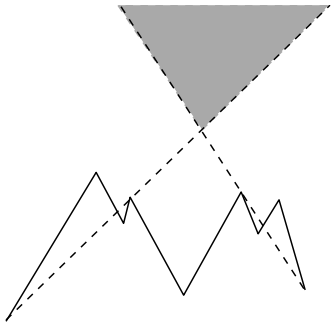


Figure 7.4: 2-kernel

We first find the k-kernel of T'. The k-kernel of a given polygon P is the set of all points p such that every point in P is k-crossing visible from p; see Figure 7.4. The algorithm of Evans and Sember [56] finds the k-kernel of T' in $O(n^2 \log n + m)$ time, where m denotes the complexity (the number of boundary vertices) of the k-kernel. The k-kernel consists of $O(n^4)$ disjoint simple polygons. The worst-case number of vertices of the k-kernel is $\Theta(n^4)$. For $k = 2$, the complexity of the k-kernel is $\Theta(n^2)$, and for $k = 3$, the complexity of the k-kernel is $O(n^4)$ and $\Omega(n^2)$ [56].

The lower envelope of the portion of the k-kernel above T is the locus of feasible locations for the top of the watchtower from which the entire terrain T is k-crossing visible. Finding the minimum-length vertical line segment between this lower envelope and T yields the optimal solution for the 1-watchtower problem; see Figure 7.6. Notice that given line segments s_1 and s_2 that intersect a vertical line, the distance between s_1 and s_2 along the vertical line is minimized at a vertex of s_1 or a vertex of s_2. Hence, to find the optimal height for the continuous 1-watchtower problem, it suffices to examine vertical line segments from the vertices of the lower envelope of the k-kernel to T, and vertical line segments from the vertices of T to the lower envelope of the k-kernel. The minimum length of these line segments is the minimum height of the continuous 1-watchtower problem.

The minimum height of a watchtower can be found by partitioning the edges of the k-kernel into those that lie above T and those that lie below T. Following this partition, the lower envelope of the edges above T is computed. By sweeping a vertical line across T and the lower envelope, we stop at all vertices to evaluate the distance on the sweep line between these two x-monotone chains, maintaining the minimum distance thus far. These steps can be implemented in a single sweep using a modification of the algorithm of Bentley and Ottmann [20]. At each event during the sweep, it suffices to measure the distance along the sweep line between T and the closest line segment above T. If this distance is less than the previously recorded minimum, we update the minimum distance and the current x-coordinate of the sweep line. Observe that no two edges of the k-kernel cross, and that no two edges of T cross. Furthermore,

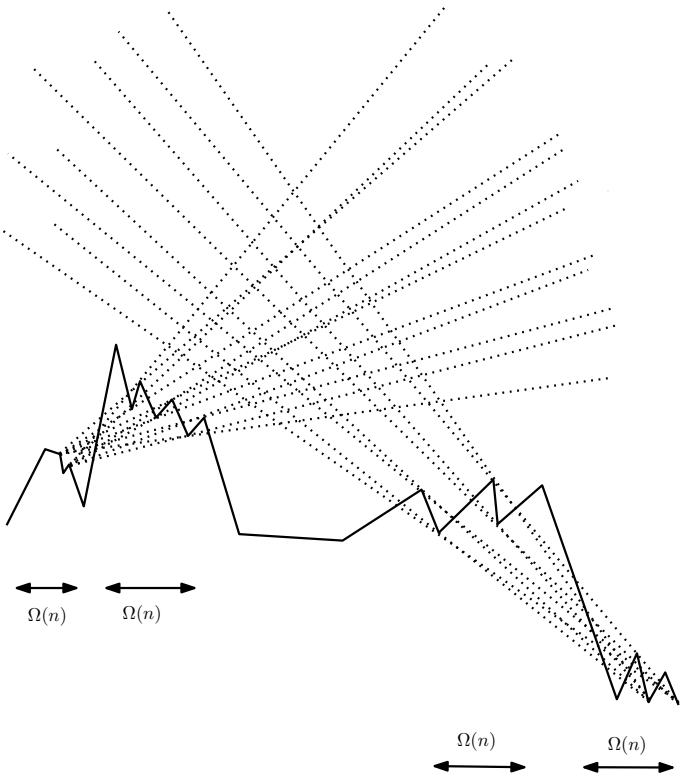


Figure 7.5: The 4-kernel of a monotone chain has $O(n^4)$ vertices. There are $O(n^2)$ cells in the arrangement of dotted lines that form the v-regions of the vertices on the terrain. These lines have $O(n^2)$ points of intersection.

if any edge of the k-kernel crosses T, then this point of intersection corresponds to the location of a watchtower of height zero: this is the solution, and the algorithm terminates. Consequently, the number of intersection events processed is at most 1. Since the number of edges in the k-kernel is $m \in O(n^4)$ and the number of edges in T is n, the total running time of the algorithm is $O((n^2 + m) \log n)$. This is due to the time complexity needed for making the k-kernel first and the sweep line algorithm on the terrain.

Although we seek the k-kernel in a restricted type of polygon, i.e., a monotone

polygon, the k-kernel for a monotone polygon has $\Theta(n^4)$ complexity in the worst case when $k \geq 4$; see Figure 7.5. The complexity of the k-kernel when $k = 3$ is unknown [56]. When $k = 2$ its complexity is $O(n^2)$.

When $k = 0$, the 0-kernel corresponds to the kernel of the polygon T'. This kernel is a convex polygon with $O(n)$ vertices from which the entire polygon is 0-crossing visible. Additionally, the kernel is the feasible region for the watchtower, and can be determined in $O(n)$ time [78; 79]; see Figure 7.6. As mentioned above, to find the solution for the continuous 1-watchtower problem, it is sufficient to examine the vertical line segments from the vertices of the kernel to T, and the vertical line segments from the vertices of T to the kernel. The boundary of the 0-kernel is an x-monotone chain consisting of $O(n)$ vertices given in order. The terrain T is an x-monotone chain of n vertices given in order. By merging the two sets of sorted vertices of T and of the kernel in $O(n)$ time, for each vertex in the merged sorted list the corresponding edge intersected by the vertical line segment can be found in $O(1)$ time by comparing the current vertex against the previous vertex in the list. If the previous vertex is on the same chain, then the current vertex intersects the same edge as the previous vertex. Otherwise, if the previous vertex is not on the same chain, then the edge that starts from the previous vertex is the intersected edge. At each step, the minimum vertical line segment encountered is maintained. Thus, the minimum length segment can be found in $O(n)$ time.

When $k = 2$, the boundary of the 2-kernel has $O(n^2)$ vertices [56]. Consequently,

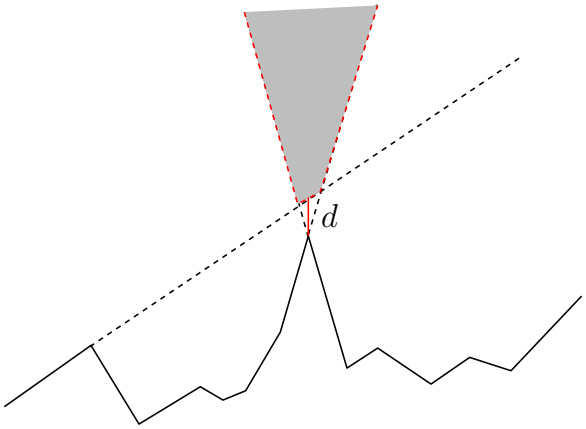


Figure 7.6: The shaded region is the intersection of the visible part of the plane for each vertex; dotted lines show the boundaries of some of these regions.

we can find the minimum length vertical line segment between the 2-kernel and the terrain T in $O(n^2 \log n)$ time, so the continuous 1-watchtower problem for 2-visibility can be solved in $O(n^2 \log n)$ time using our algorithm.

Theorem 5. *The continuous 1-watchtower problem can be solved in* $O((n^2+m)\log n)$ *time under k-crossing visibility, where* $m \in O(n^4)$ *is the size of the k-kernel. For* $k = 0$ *and* $k = 2$*, the continuous 1-watchtower problem can be solved in* $O(n)$ *and* $O(n^2 \log n)$ *time, respectively.*

7.3 Discrete 1-Watchtower Problem

In this section, first, we explain the main lemmas which hold for the discrete 1-watchtower problem. Then, we propose an $O(n^4)$-time algorithm for the discrete k-crossing visible watchtower problem on a terrain T. We propose how to reduce the time complexity to $O(n^3 \log n)$. Finally, we express an algorithm with $O(n^3)$ time complexity.

Evans and Sember [56] defined the *v-region* of a vertex v as the region of all points q that are k-crossing visible from every point on the ray from q through v, with respect to the polygon P. The boundary of each v-region is a simple polygon with $O(n)$ vertices [56]. Computing the v-region of each vertex of the polygon takes $O(n \log n)$ time. We compute the v-region for each vertex of T' in $O(n \log n)$ time per vertex using the algorithm of Evans and Sember [56], using $O(n^2 \log n)$ total time. The intersection of v-regions of the polygon P is the k-kernel of the polygon P [56]. In other words, the intersection of v-regions of the vertices of P is the locus for the top of the watchtower. So, the intersection of the v-regions is the k-kernel of T' [56], which is the region where the entire region T' (including T) is k-crossing visible from. So, T is k-crossing visible from any watchtower located in this region.

Observation 2. *The intersection of the v-regions of the vertices of T corresponds to the set of feasible locations for the top of the watchtower.*

In the discrete problem, the watchtower must be located on a vertical line emanating from a vertex of the terrain. Consider a vertical line passing through a vertex of the terrain. We find the intersection of the v-regions of the vertices of T with this vertical line.

Lemma 13. *Any vertical line crosses the boundaries of the v-regions of the vertices of T, and the number of such crossings is $O(n^2)$.*

Proof. The number of vertices on the boundary of each v-region is $O(n)$. So each v-region may intersect a vertical line $O(n)$ times. As there exist n v-regions, the number of intersections between v-regions and any given vertical line is $O(n^2)$. □

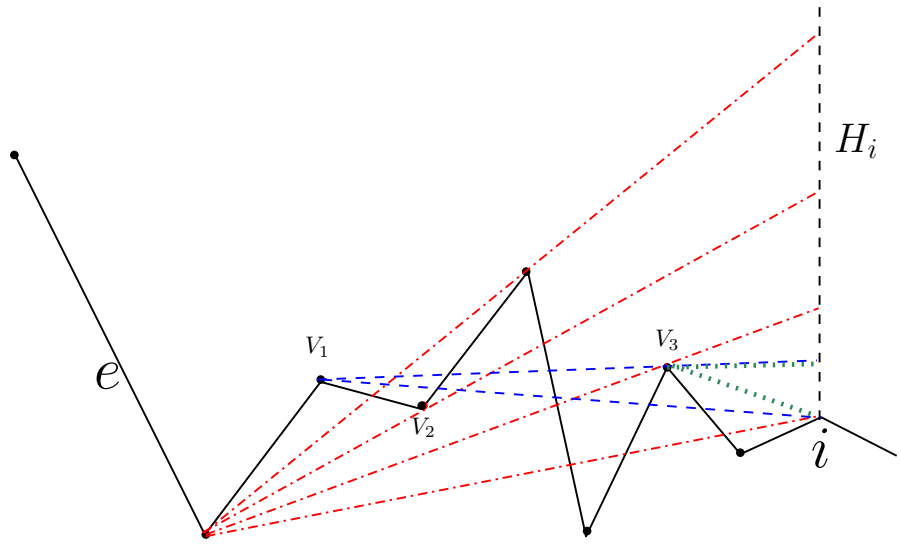


Figure 7.7: The v-regions and their intersection with H_i for three vertices V_1, V_2 and V_3 are shown in dashed, dotted, dashed and dotted respectively.

Let V_i denote the v-region of vertex v_i in T. We have the following lemma:

Lemma 14. *The intersection of any v-region with any vertical line is a set of at most n disjoint intervals on the line, where the topmost interval is open.*

Proof. Consider a bounding box around T'. The v-region of a vertex v_i is a closed Jordan curve with $O(n)$ complexity. The intersection between the vertical line and the inside of this closed Jordan curve is a set of $O(n)$ intervals. The last interval is open as after moving sufficiently high above the terrain T all of T will be visible while looking toward the vertex v_i. □

Consider a vertical line ℓ_i passing through a given vertex v_i of T, and the intersections with the v-regions $V_1, \ldots, V_n$ for the vertices $v_1, \ldots, v_n$ of T. Let each v-region be determined by a specific colour. As a result, we have n different colours of intervals on the line ℓ_i. Each colour is a set of $O(n)$ intervals, and intervals with

the same colour do not intersect. If the optimal watchtower lies on this vertical line, it is in the interval with the lowest y-coordinate which intersects all n v-regions. We define depth-n intervals as the intervals on ℓ_i on which all n v-regions intersect.

Lemma 15. *The minimum height of a watchtower located above the vertex v_i is the closest depth-n interval.*

Proof. Intervals with the same colour do not intersect each other. So, the maximum number of intersection is n where n v-regions intersect. So, a depth-n interval is in the k-kernel and T is k-crossing visible from such intervals. Among all such depth-n intervals we look for the one which has the smaller distance with the terrain. □

As a result of Lemma 15, we can remove the colour on the intervals. This transforms the problem to that of finding the depth-n intervals among $O(n^2)$ intervals.

7.3.1 $O(n^4)$-Time Algorithm

In the following, we propose the $O(n^4)$-time algorithm for the discrete 1-watchtower problem.

Fix a vertex v_i and consider the vertical line ℓ_i passing through v_i. We find the sorted list (by y-coordinate) of the intersections of each v-region V_j with the line ℓ_i in $O(n^2 \log n)$ time. So we have n sorted lists each containing $O(n)$ intervals. Let's label these lists as $L_1, L_2, ..., L_n$. For these n sorted lists, build a fractional cascading data structure in $O(n^2)$ time[38; 39]. Given a point p on ℓ_i, this data structure allows in $O(n)$ time to find, for each list, the interval where p lies (if any).

Lemma 16. *The deepest interval with the minimum y-coordinate for a given line ℓ_i can be found in $O(n^3)$ time.*

Proof. For each L_i, take the bottom point of the interval with smallest y-coordinate, and let x be the maximum of all such points. This can be done in $O(n)$ time since each list of intervals is sorted by y-coordinate. The number of $v-regions$ intersecting x can be computed in $O(n)$ time, denoted as the *depth* of x, using fractional cascading while searching for x. If the depth of x is n, then we are done. Otherwise, there are some lists L_j that do not contain x in any interval. For each of these lists, take the point above x that is closest to x. Finding all of these points can be done in $O(n)$ time since fractional cascading gave us a pointer to each of these when we computed the depth of x. We compute the highest of these points and repeat the process to find the depth of this new interval.

Among all the points for which this algorithm computed the depth, consider the ones with depth n. At least one such point will be found since the topmost interval in each list is open. Among those, the lowest one is the deepest lowest point. Each step takes $O(n)$ time and the algorithm repeats $O(n^2)$ times. So in total, the algorithm takes $O(n^3)$ time. □

Theorem 6. *The discrete 1-watchtower problem can be solved in $O(n^4)$ time under k-crossing visibility.*

Proof. There are n vertices in T corresponding to n candidate vertical lines for the watchtower. By Lemmas 15 and 19, finding the minimum height of a watchtower

located at the vertex v_i takes $O(n^3)$ time. So, the total required time is $O(n^4)$.

□

7.3.2 $O(n^3 \log n)$-Time Algorithm

The approach proposed in the section 7.3.1 can be improved. This new approach takes $O(n^3 \log n)$ time to solve the discrete 1-watchtower problem.

Lemma 17. *The deepest interval with the minimum height for a set of $O(n^2)$ intervals on a given line ℓ_i can be found in $O(n^2 \log n)$ time.*

Proof. First, we suppose $O(n^2)$ intervals' endpoints are sorted (by y-coordinate). Let x be a counter variable initialized as 0. We start from the vertex v_i and move above the terrain T along the line ℓ_i. Parameter x increases by 1 at the beginning of an interval, and decreases by 1 when the end of an interval is reached. The first place which x equals n, is the placement of the watchtower with the minimum height located at the vertex v_i. As there exist $O(n^2)$ intervals, there are $O(n^2)$ such events, each takes constant time to process. So, this process takes $O(n^2)$ time complexity. As sorting $O(n^2)$ intervals takes $O(n^2 \log n)$ time, the overall time complexity is $O(n^2 \log n)$. □

Theorem 7. *The discrete 1-watchtower problem can be solved in $O(n^3 \log n)$ time under k-crossing visibility.*

Proof. There are n vertices in T corresponding to n candidate vertical lines for the watchtower. By Lemmas 15 and 17, finding the minimum height of a watchtower located at the vertex v_i takes $O(n^2 \log n)$ time. So, the total required time is $O(n^3 \log n)$.

□

7.3.3 $O(n^3)$-Time Algorithm

An improvement can be applied to the above algorithm so that it takes $O(n^3)$ time to solve the discrete 1-watchtower problem for k-crossing visibility.

Lemma 18. *Given a v-region of a vertex of the terrain T, finding and sorting the intersections of this v-region with a given vertical line takes $O(n)$ time.*

Proof. We can find the intersection of a v-region with the vertical line ℓ_i in $O(n)$ time as the number of edges of each v-region is $O(n)$. This gives a set of $O(n)$ intervals on ℓ_i. We can sort these intervals in $O(n)$ time as the v-region is a Jordan arc [59]. □

Fix a vertex v_i and consider the vertical line ℓ_i passing through v_i. We find the sorted list (by y-coordinate) of the intersections of each v-region V_j with the line ℓ_i in $O(n^2)$ time, using Lemma 18. So we have n sorted lists each containing $O(n)$ intervals. Let these lists be labeled as $L_1, L_2, ..., L_n$. We have the following lemma:

Lemma 19. *The deepest interval with the minimum y-coordinate for a set of $O(n^2)$ intervals on a given line ℓ_i can be found in $O(n^2)$ time.*

Proof. As mentioned in Lemma 18, each set of n intervals in the list L_i can be sorted in linear time. There exist n lists, so it takes $O(n^2)$ time to sort all $L_1, \ldots, L_n$. Consider two lists L_1 and L_2. First, we find the intersections between L_1 and L_2. Given two sets of sorted intervals X and Y, their intersection can be found in $O(|X| + |Y| + m)$ time, where m denotes the number of output intervals [97]. As X and Y are of size $O(n)$ for the lists L_1 and L_2, m is also of size $O(n)$. This is because if an interval in L_1

intersects m intervals of L_2, remaining intervals in L_1 can intersect at most $n-m+2$ intervals in L_2. As a result, finding the intersection between L_1 and L_2 takes $O(n)$ time; let the output list be called L'_1. Notice that the intersection of two intervals of size n includes at most n intervals. Next, we find the intersection of L'_1 and L_3 (called L'_2) in $O(n)$ time. Repeating this process, the intersection between L'_{n-1} and L_n results in the intersections of $L_1, L_2, ..., L_n$. There are n steps, each taking $O(n)$ time. The algorithm takes $O(n^2)$ total time. □

Theorem 8. *The discrete 1-watchtower problem can be solved in $O(n^3)$ time under k-crossing visibility.*

Proof. There are n vertices in T corresponding to n vertical lines as the candidates for the location of the watchtower. By Lemmas 15 and 19, finding the minimum height of a watchtower located at the vertex v_i takes $O(n^2)$ time. So, the total required time is $O(n^3)$.

□

Considering 0-crossing visibility, the kernel is the potential location of the top of the watchtower as described for the continuous version. The difference between the discrete and continuous versions is that in the discrete version, the algorithm restricts the possible watchtowers to those whose x-coordinates coincide with a vertex of T. As a result, the discrete 1-watchtower problem under 0-crossing visibility can also be solved in $O(n)$ time.

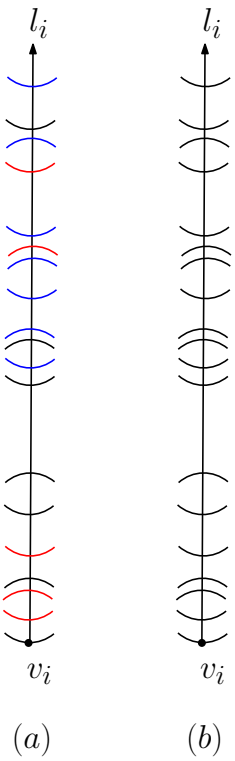


Figure 7.8: a. Colored intervals on a vertical line ℓ_i. b. Intervals can be considered as a set of $O(n^2)$ intervals without color.

In the case of 2-crossing visibility, we apply the same approach as for the continuous version. The key difference is that only the vertical line segments emanating from vertices of the terrain are of interest as the possible locations for the watchtower. As a result, the discrete version of the 2-watchtower problem can also be solved in $O(n^2 \log n)$ time.

7.3.4 Comparison Between k-Visibility and 0-Visibility

As mentioned, both the discrete and continuous versions of the 1-watchtower problem for 0-crossing visibility can be solved in $O(n)$ time, while for k-crossing visibility the time complexity increases significantly when $k > 0$. The main reason

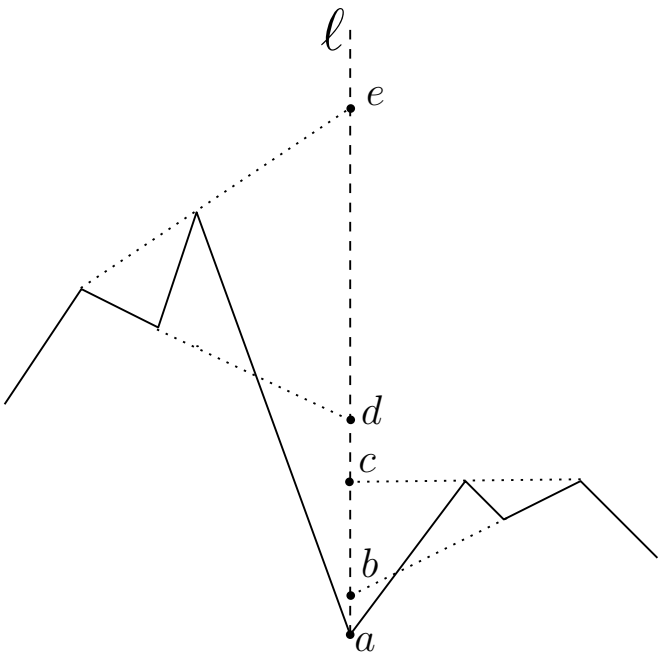


Figure 7.9: Going up and losing visibility: On point a, the entire terrain T is 2-crossing visible. At point b, a part on the right sight of the horizontal line is not 2-crossing visible anymore. At point c, the entire terrain T becomes 2-crossing visible, while on d apart on the left side of the horizontal line is not 2-crossing visible. At point e, T is 2-crossing visible again.

is the fact that when $k \neq 0$, the k-kernel can be disconnected. Under 0-visibility, increasing the height of a watchtower always increases its visibility; that is, if p and q are two points on a vertical line above T, where p lies above q, then the region of T visible to q is contained in the region of T visible to p. This property does not hold when $k > 0$; q could see all of T (i.e., q is in the k-kernel), whereas p does not see all of T, even though p lies above q. See Figure 7.9.

Chapter 8

Conclusion

While in this thesis several fundamental questions related to k-visibility are answered, many questions remain open.

Chapter 4 presents the first algorithm parameterized in terms of k for computing the k-visible region for a given point q in a given polygon P, resulting in asymptotically faster worst-case running time relative to previous algorithms when k is $o(\log n)$, and bridging the gap between the $O(n)$-time algorithm for computing the 0-visibility region of q in P [53; 75; 69], and the $O(n \log n)$-time algorithm for computing the k-visibility region of q in P [10]. It remains open whether the problem can be solved faster. In particular, an $O(n \log k)$-time algorithm would provide a natural parameterization for all k. Alternatively, can a lower bound of $\Omega(n \log n)$ be shown in the worst-case time when k is $\omega(\log n)$?

Chapter 5 proposes data structures to report the k-visibility region of a query

point more efficiently. The remaining open question is whether the size of this structure can be reduced. This may be possible with the price of increasing the query time. Another interesting question is whether we can design a data structure where the query time depends on not only n, but also on k.

In Chapter 6, we proposed algorithms for reporting the k-visibility polygon in the limited workspace model, and we provided time-space trade-offs for this problem. We leave it as an open problem whether there exists an output-sensitive algorithm whose running time depends on the number of windows in the k-visibility region, instead of the critical vertices in the input polygon.

In Chapter 7 the watchtower problem was discussed. The 1-watchtower problem generalizes to the M-watchtower problem, where instead of positioning a single watchtower to guard the terrain T, an algorithm must select positions for M watchtowers. The goal is to minimize the maximum height of any watchtower, while ensuring that each point on T is k-crossing visible from at least one watchtower. To solve the continuous 1-watchtower problem, it suffices to consider candidate locations for the watchtower whose x-coordinate coincides with that of a vertex of T or a vertex of the k-kernel of T. This property is not true in general for the continuous M-watchtower problem, even when $M = 2$; see Figures 8.1 and 8.2. It remains open to find an efficient algorithm to solve the (discrete or continuous) M-watchtower problem under k-crossing visibility, even for $M = 2$.

A polygon P is said to be *weakly visible* from a region s inside P if and only if

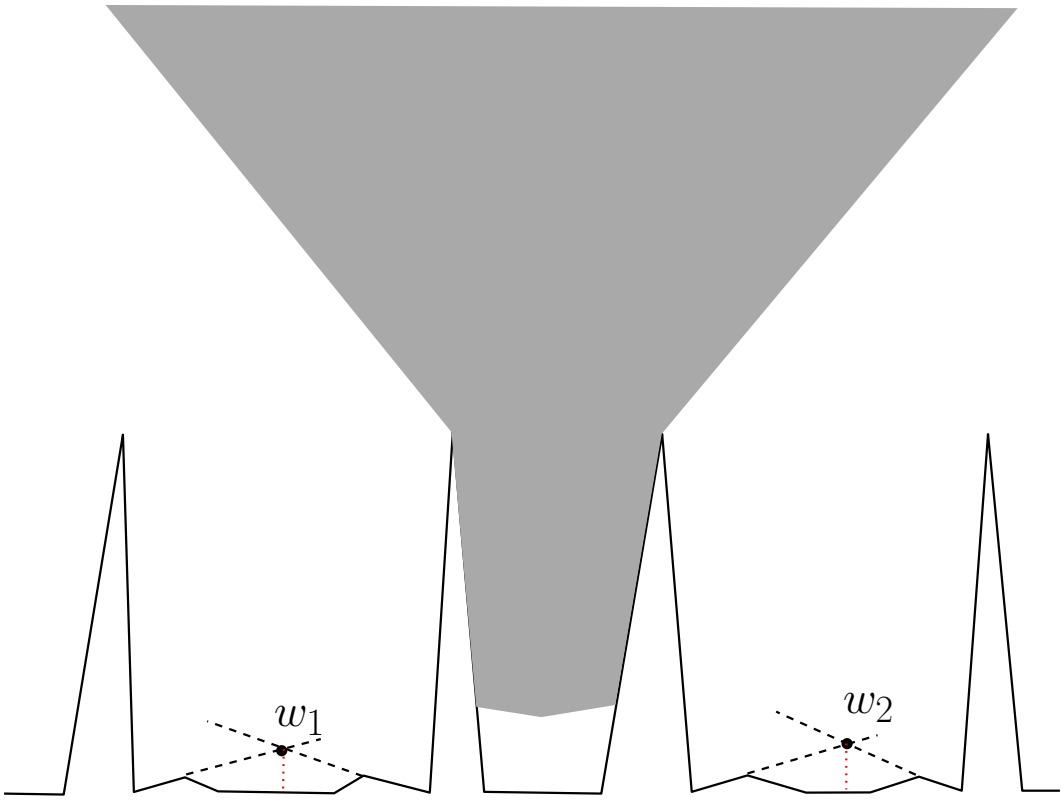


Figure 8.1: The x-coordinates of the watchtowers w_1 and w_2 do not coincide with that of a vertex of T or a vertex of the 2-kernel of T, and each point on T is 2-crossing visible from either w_1 or w_2.

each point in P is visible from at least one point in s. If each point in P is visible from all points in s, then P is said to be *strongly visible* from s. Besides the problem discussed in this thesis, k-visibility can be studied under different settings, likewise weak visibility and strong visibility. Approximation algorithms may be of interest in attempting to solve these problems.

For wireless communication, in addition to considering the number of obstacles between two devices, some models also consider the distance between obstacles. For example, we can consider a model where two points p and q are mutually visible when

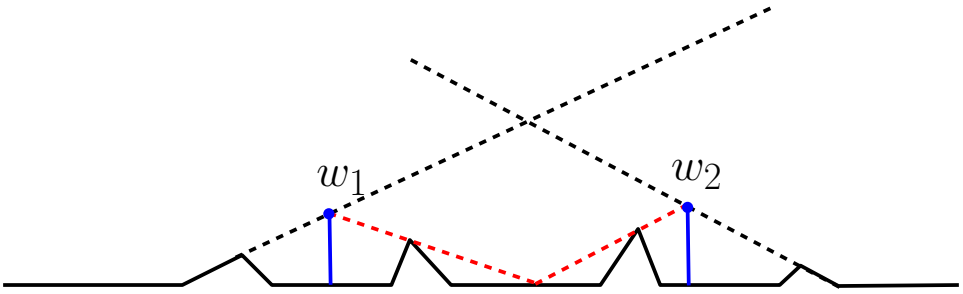


Figure 8.2: Even when $k = 0$ for 2-watchtower problem, the x-coordinates of the watchtowers does not coincide with those of vertices of the terrain, vertices of the k-kernel, nor of the intersections of the n^2 lines determined by pairs of vertices of the terrain.

the line segment between pq intersects at most k times with the obstacles in the plane and the distance between p and q is at most d, for some given d. This may be a more realistic model of wireless communication.

Recently, the Pursuit-Evasion problem, an extension of the Art Gallery problem, has gained significant attention. The most basic form of the Pursuit-Evasion problem is as follows: given a simple polygon P, there exists a set of mobile agents, called the intruders, inside P whose positions move along continuous unknown trajectories. A *trajectory* is a continuous function $f : \mathbb{R} \to \mathbb{R}^2$, i.e., a mapping of time to position in the plane; specific problems may impose additional constraints, such as bounding the maximum speed $||f(t_1) - f(t_2)||/|t_1 - t_2|$. The goal of the problem is to assign a set of trajectories inside P to another set of mobile agents, called the pursuers, such that the pursuers are able to detect every intruder by moving on these defined trajectories. By *detection* we mean either to touch or see the intruder. When the goal of detection is to see the intruder, we can distinguish a variety of types of visibility for the pursuer; such as the case of pursuers with k-crossing visibility.

The Pursuit-Evasion problem is widely studied in robotics, graph theory, and computational geometry [5; 41]. The problem was first considered inside a polygonal environment by Suzuki and Yamashita [95]. Different versions of the Pursuit-Evasion problem can be defined based on four parameters [68]:

- the environment in which the pursuers attempt to find the intruders (e.g. plane or polygon)
- the manner of pursuer and intruder movement (e.g. bounded or unbounded speed)
- the definition of detection (seeing or touching)
- the information the pursuers and intruders have about each other

Changing any of the above parameters can create a completely new problem. For instance, the visibility-based Pursuit-Evasion problem for a single omnidirectional pursuer has been widely studied [95; 46; 74; 73]. This problem was considered for a pursuer who can see along a line whose direction can be modified by the pursuer [94]. Icking and Klein [66], and also Hefferman [64] designed algorithms for cases with two mutually visible pursuers capable of movement along the boundary in a simple polygon P to detect the intruders. In another recent study, the pursuers and intruders move inside a room which is a polygon with one point that has to be seen all the time by a pursuer [80].

For all the varieties of visibility based Pursuit-Evasion problem, k-crossing visibility can be considered for the pursuer. Such a setting was only considered in [11]. This leaves great potential to work on k-visibility in this field.

Bibliography

[1] M. Abrahamsen. An optimal algorithm computing edge-to-edge visibility in a simple polygon. In *Proc. 25th Canad. Conf. Comput. Geom. (CCCG)*, 2013.

[2] P. Agarwal, S. Bereg, O. Daescu, H. Kaplan, S. Ntafos, and B. Zhu. Guarding a terrain by two watchtowers. In *Proceedings of the twenty-first annual symposium on Computational geometry*, pages 346–355. ACM, 2005.

[3] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. In *Proc. 25th European Workshop on Computational Geometry (EWCG 2009)*, pages 167–170, 2009.

[4] O. Aichholzer, R. Fabila-Monroy, D. Flores-Peñaloza, T. Hackl, C. Huemer, J. Urrutia, and B. Vogtenhuber. Modem illumination of monotone polygons. *Computational Geometry*, 68:101–118, 2018.

[5] B. Alspach. Searching and sweeping graphs: a brief survey. *Le matematiche*, 59(1, 2):5–37, 2006.

[6] B. Aronov, L. J. Guibas, M. Teichmann, and L. Zhang. Visibility queries

and maintenance in simple polygons. *Discrete & Computational Geometry*, 27(4):461–483, 2002.

[7] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[8] T. Asano, K. Buchin, M. Buchin, M. Korman, W. Mulzer, G. Rote, and A. Schulz. Memory-constrained algorithms for simple polygons. *Comput. Geom. Theory Appl.*, 46(8):959–969, 2013.

[9] Y. Bahoo, B. Banyassady, P. Bose, S. Durocher, and W. Mulzer. Time-space trade-off for finding the k-visibility region of a point in a polygon. In *WALCOM: Algorithms and Computation: 11th International Conference and Workshops, WALCOM 2017, Hsinchu, Taiwan, March 29–31, 2017, Proceedings*, volume 10167, page 308. Springer, 2017.

[10] Y. Bahoo, B. Banyassady, P. Bose, S. Durocher, and W. Mulzer. A time-space trade-off for computing the k-visibility region of a point in a polygon. *Theoretical Computer Science*, 2018.

[11] Y. Bahoo, A. Mohades, M. Eskandari, and M. Sorouri. 2-modem pursuit-evasion problem. In *29th European Workshop on Computational Geometry*, pages 201–204. TU Braunschweig, 2013.

[12] A. Bajuelos, S. Canales, G. Hernández, and M. Martins. A hybrid metaheuristic strategy for covering with wireless devices. *Journal of Universal Computer Science*, 18(14):1906–1932, 2012.

[13] B. Ballinger, N. Benbernou, P. Bose, M. Damian, E. Demaine, V. Dujmović, R. Flatland, F. Hurtado, J. Iacono, A. Lubiw, et al. Coverage with k-transmitters in the presence of obstacles. *Journal of Combinatorial Optimization*, 25(2):208–233, 2013.

[14] B. Banyassady, M. Korman, and W. Mulzer. Geometric algorithms with limited workspace: A survey. *arXiv preprint arXiv:1806.05868*, 2018.

[15] L. Barba, M. Korman, S. Langerman, K. Sadakane, and R. I. Silveira. Space-time trade-offs for stack-based algorithms. *Algorithmica*, 72(4):1097–1129, 2015.

[16] L. Barba, M. Korman, S. Langerman, and R. I. Silveira. Computing a visibility polygon using few variables. *Comput. Geom. Theory Appl.*, 47(9):918–926, 2014.

[17] P. Belleville, P. Bose, J. Czyzowicz, J. Urrutia, and J. Zaks. K-guarding polygons on the plane. In *CCCG*, pages 381–386, 1994.

[18] B. Ben-Moshe, P. Carmi, and M. Katz. Computing all large sums-of-pairs in rn and the discrete planar two-watchtower problem. *Information processing letters*, 89(3):137–139, 2004.

[19] B. Ben-Moshe, M. Katz, and J. Mitchell. A constant-factor approximation algorithm for optimal 1.5 d terrain guarding. *SIAM Journal on Computing*, 36(6):1631–1647, 2007.

[20] J. L. Bentley and T. A. Ottmann. Algorithms for reporting and counting geometric intersections. *IEEE Transactions on computers*, (9):643–647, 1979.

[21] S. Bereg. On k-vertex guarding simple polygons. *Computational Geometry and Discrete Mathematics*, 1641:106–113, 2009.

[22] S. Bespamyatnikh, Z. Chen, K. Wang, and B. Zhu. On the planar two-watchtower problem. *Computing and Combinatorics*, pages 121–130, 2001.

[23] S. Birchfield. An introduction to projective geometry (for computer vision). *Unpublished note, Stanford university*, 1998.

[24] P. Bose, A. Lubiw, and J. Munro. Efficient visibility queries in simple polygons. *Computational Geometry*, 23(3):313–335, 2002.

[25] W. Brink. Lecture notes in computer vision, Fall 2017.

[26] H. Brönnimann, T. Chan, and E. Chen. Towards in-place geometric algorithms and data structures. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 239–246. ACM, 2004.

[27] H. Brönnimann, J. Iacono, J. Katajainen, P. Morin, J. Morrison, and G. Toussaint. In-place planar convex hull algorithms. In *Latin American Symposium on Theoretical Informatics*, pages 494–507. Springer, 2002.

[28] D. Busto, W. Evans, and D. Kirkpatrick. On k-guarding polygons. In *CCCG*, 2013.

[29] S. Cannon, T. Fai, J. Iwerks, U. Leopold, and C. Schmidt. Combinatorics and complexity of guarding polygons with edge and point 2-transmitters. *arXiv preprint arXiv:1503.05681*, 2015.

[30] T. Chan. Faster core-set constructions and data-stream algorithms in fixed dimensions. *Computational Geometry*, 35(1-2):20–35, 2006.

[31] T. Chan. Comparison-based time-space lower bounds for selection. *ACM Transactions on Algorithms*, 6(2):26, 2010.

[32] T. Chan and E. Chen. Multi-pass geometric algorithms. *Discrete Comput. Geom.*, 37(1):79–102, 2007.

[33] T. Chan and E. Chen. In-place 2-d nearest neighbor search. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 904–911. Society for Industrial and Applied Mathematics, 2008.

[34] T. Chan and E. Chen. Optimal in-place and cache-oblivious algorithms for 3-d convex hulls and 2-d segment intersection. *Computational Geometry*, 43(8):636–646, 2010.

[35] T. Chan, J. I. Munro, and V. Raman. Selection and sorting in the restore model. In *Proc. 25th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 995–1004, 2014.

[36] H. Chang, J. Erickson, and C. Xu. Detecting weakly simple polygons. In *Proceedings of the SIAM Symposium on Discrete Algorithms (SODA)*, pages 1655–1670. ACM-SIAM, 2015.

[37] B. Chazelle. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry*, 6(3):485–524, 1991.

[38] B. Chazelle and L. Guibas. Fractional cascading: I. a data structuring technique. *Algorithmica*, 1(1-4):133–162, 1986.

[39] B. Chazelle and L. Guibas. Fractional cascading: II. applications. *Algorithmica*, 1(1-4):163–191, 1986.

[40] D. Chen, V. Estivill-Castro, and J. Urrutia. Optimal guarding of polygons and monotone chains. In *CCCG*, pages 133–138, 1995.

[41] T. Chung, G. Hollinger, and V. Isler. Search and pursuit-evasion in mobile robotics. *Autonomous robots*, 31(4):299–316, 2011.

[42] V. Chvatal. A combinatorial theorem in plane geometry. *Journal of Combinatorial Theory, Series B*, 18(1):39–41, 1975.

[43] K. Clarkson and K. Varadarajan. Improved approximation algorithms for geometric set cover. *Discrete & Computational Geometry*, 37(1):43–58, 2007.

[44] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[45] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to algorithms, third edition*. MIT press, 2009.

[46] D. Crass, I. Suzuki, and M. Yamashita. Searching for a mobile intruder in a corridor—the open edge variant of the polygon search problem. *International Journal of Computational Geometry & Applications*, 5(04):397–412, 1995.

[47] L. Davis and M. Benedikt. *Computational models of space: Isovists and isovist fields*. Citeseer, 1979.

[48] M. De Berg, M. Van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry. In *Computational geometry*, pages 1–17. Springer, 1997.

[49] J. Dean, A. Lingas, and J. Sack. Recognizing polygons, or how to spy. *The Visual Computer*, 3(6):344–355, 1988.

[50] F. Duque and C. Hidalgo-Toscano. An upper bound on the k-modem illumination problem. *arXiv preprint arXiv:1410.4099*, 2014.

[51] A. Efrat, S. Har-Peled, and J. Mitchell. Approximation algorithms for two optimal location problems in sensor networks. In *2nd International Conference on Broadband Networks, 2005.*, pages 714–723. IEEE, 2005.

[52] S. Eidenbenz, C. Stamm, and P. Widmayer. Inapproximability results for guarding polygons and terrains. *Algorithmica*, 31(1):79–113, 2001.

[53] H. El Gindy and D. Avis. A linear algorithm for computing the visibility polygon from a point. *Journal of Algorithms*, 2(2):186–197, 1981.

[54] K. Elbassioni, E. Krohn, D. Matijević, J. Mestre, and D. Ševerdija. Improved approximations for guarding 1.5-dimensional terrains. *Algorithmica*, 60(2):451–463, 2011.

[55] W. Evans and J. Sember. k-star-shaped polygons. In *the 22nd Canadian Conference on Computational Geometry (CCCG2010)*, pages 215–218, 2010.

[56] W. Evans and J. Sember. *k*-star-shaped polygons. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 215–218, 2010.

[57] R. Fabila-Monroy, A. Vargas, and J. Urrutia. On modem illumination problems. *XIII encuentros de geometria computacional, Zaragoza, Spain*, 2009.

[58] S. Fisk. A short proof of chvátal's watchman theorem. *Journal of Combinatorial Theory, Series B*, 24(3):374, 1978.

[59] K. Fung, T. Nicholl, R. Tarjan, and C. J. Van W. Simplified linear-time Jordan sorting and polygon clipping. *Information Processing Letters*, 35(2):85–92, 1990.

[60] S. Ghosh. *Visibility algorithms in the plane*. Cambridge University Press, 2007.

[61] M. Gibson, G. Kanade, E. Krohn, and K. Varadarajan. An approximation scheme for terrain guarding. *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 140–148, 2009.

[62] H. González-Baños. A randomized art-gallery algorithm for sensor placement. In *In Proceedings of the seventeenth annual symposium on Computational geometry*, pages 232–240. ACM, 2001.

[63] M. He. Succinct and implicit data structures for computational geometry. In *Space-Efficient Data Structures, Streams, and Algorithms*, pages 216–235. Springer, 2013.

[64] P. Heffernan. An optimal algorithm for the two-guard problem. *International Journal of Computational Geometry & Applications*, 6(01):15–44, 1996.

[65] K. Hoffmann, K. Mehlhorn, P. Rosenstiehl, and R. Tarjan. Sorting Jordan sequences in linear time using level-linked search trees. *Information and Control*, 68(1-3):170–184, 1986.

[66] C. Icking and R. Klein. The two guards problem. *International Journal of Computational Geometry & Applications*, 2(03):257–285, 1992.

[67] IEEE. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. *IEEE Std 802.11-1997*, pages 1–445, Nov 1997.

[68] V. Isler. *Algorithms for distributed and mobile sensing.* PhD thesis, University of Pennsylvania, 2004.

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

[70] J. King. A 4-approximation algorithm for guarding 1.5-dimensional terrains. In *LATIN*, volume 3887, pages 629–640. Springer, 2006.

[71] J. King and E. Krohn. Terrain guarding is np-hard. *SIAM Journal on Computing*, 40(5):1316–1339, 2011.

[72] D. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal on Computing*, 12(1):28–35, 1983.

[73] S. LaValle and J. Hinrichsen. Visibility-based pursuit-evasion: The case of curved environments. *IEEE Transactions on Robotics and Automation*, 17(2):196–202, 2001.

[74] S. LaValle, D. Lin, L. Guibas, J. Latombe, and R. Motwani. Finding an unpredictable target in a workspace with obstacles. In *In Proceedings of 1997 IEEE International Conference on Robotics and Automation*, volume 1, pages 737–742. IEEE, 1997.

[75] D. Lee. Visibility of a simple polygon. *Computer Vision, Graphics, and Image Processing*, 22(2):207–221, 1983.

[76] D. Lee and A. Lin. Computational complexity of art gallery problems. *IEEE Transactions on Information Theory*, 32(2):276–282, 1986.

[77] D. Lee and F. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal on computing*, 6(3):594–606, 1977.

[78] D. Lee and F. Preparata. An optimal algorithm for finding the kernel of a polygon. Technical report, University of Illinois at Urbana–Champaign, Coordinated Science lab, 1977.

[79] D. Lee and F. Preparata. An optimal algorithm for finding the kernel of a polygon. *Journal of the ACM (JACM)*, 26(3):415–421, 1979.

[80] J. Lee, S. Park, and K. Chwa. Searching a polygonal room with one door by a 1-searcher. *International Journal of Computational Geometry & Applications*, 10(02):201–220, 2000.

[81] N. Mouawad and T. Shermer. The superman problem. *The Visual Computer*, 10(8):459–473, 1994.

[82] J. Munro and M. Paterson. Selection and sorting with limited storage. In *Foundations of Computer Science, 1978., 19th Annual Symposium on*, pages 253–258. IEEE, 1978.

[83] J. Munro and V. Raman. Selection from read-only memory and sorting with minimum data movement. *Theoret. Comput. Sci.*, 165(2):311–323, 1996.

[84] S. Muthukrishnan. Data streams: Algorithms and applications (foundations and trends in theoretical computer science). *Hanover, MA: Now Publishers Inc*, 2005.

[85] G. Navarro. *Compact data structures: A practical approach*. Cambridge University Press, 2016.

[86] J. O'Rourke. *Art gallery theorems and algorithms*, volume 57. Oxford University Press Oxford, 1987.

[87] F. Preparata. A new approach to planar point location. *SIAM Journal on Computing*, 10(3):473–482, 1981.

[88] P. Rosenstiehl. Planar permutations defined by two intersecting Jordan curves. *Graph Theory and Combinatorics*, pages 259–271, 1984.

[89] I. Salleh. K-vertex guarding simple polygons. *Computational Geometry*, 42(4):352–361, 2009.

[90] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the ACM*, 29(7):669–679, 1986.

[91] M. Shamos and D. Hoey. Geometric intersection problems. In *In Proceedings of 17th IEEE Annual Symposium on Foundation of Computer Science*, pages 208–215. IEEE, 1976.

[92] M. Sharir. The shortest watchtower and related problems for polyhedral terrains. *Information Processing Letters*, 29(5):265–270, 1988.

[93] T. Shermer. Recent results in art galleries. *Proceedings of the IEEE*, 80(9):1384–1399, 1992.

[94] B. Simov, G. Slutzki, and S. LaValle. Pursuit-evasion using beam detection. In *In Proceedings of IEEE International Conference on Robotics and Automation*, volume 2, pages 1657–1662. IEEE, 2000.

[95] I. Suzuki and M. Yamashita. Searching for a mobile intruder in a polygonal region. *SIAM Journal on computing*, 21(5):863–888, 1992.

[96] B. Zhu. Computing the shortest watchtower of a polyhedral terrain in $O(n \log n)$ time. *Computational Geometry*, 8(4):181–193, 1997.

[97] A. Zomorodian and H. Edelsbrunner. Fast software for box intersections. *International Journal of Computational Geometry & Applications*, 12:143–172, 2002.