Visibility and Ray Shooting Queries in Polygonal Domains

Danny Z. Chen[1,⋆] and Haitao Wang[2,⋆⋆]

[1] Department of Computer Science and Engineering
University of Notre Dame, Notre Dame, IN 46556, USA
dchen@cse.nd.edu
[2] Department of Computer Science
Utah State University, Logan, UT 84322, USA
haitao.wang@usu.edu

Abstract. Given a polygonal domain (or polygon with holes) in the plane, we study the problem of computing the visibility polygon of any query point. As a special case of visibility problems, we also study the ray-shooting problem of finding the first point on the polygon boundaries that is hit by any query ray. These are fundamental problems in computational geometry and have been studied extensively. We present new algorithms and data structures that improve the previous results.

1 Introduction

Given a set $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$ of h pairwise-disjoint polygonal obstacles of totally n vertices in the plane, the space minus the interior of all obstacles is called the *free space*. Two points are *visible* to each other if the open line segment connecting them lies entirely in the free space. For any point q in the free space, the *visibility polygon* of q, denoted by $Vis(q)$, is the set of points in the plane visible to q. The *visibility query* problem seeks an efficient data structure that allows fast computation of $Vis(q)$ for any query point q. Let $|Vis(q)|$ denote the number of vertices of $Vis(q)$. We present two new visibility query data structures. The first one uses $O(n^2)$ space and is constructed in $O(n^2 \log n)$ time; for any query point q, $Vis(q)$ can be computed in $O(\log^2 n + \min\{h, |Vis(q)|\} \log n + |Vis(q)|)$ time. Our second data structure is of size $O(n + h^2)$, and its preprocessing time and query time are $O(n + h^2 \log h)$ and $O(|Vis(q)| \log n)$, respectively. Note that in some cases the value h can be substantially smaller than n.

We also study the *ray-shooting query*, a special case of visibility problems: Given any query ray $\sigma(q)$ with its origin point q in the free space, find the first point on the obstacle boundaries or in infinity that is hit by $\sigma(q)$. We construct a data structure of size $O(n + h^2)$ in $O(n + h^2 \cdot \text{poly}(\log \text{h}))$ time that answers any query in $O(\log n)$ time, where poly(log h) is a polynomial function of $\log h$.

⋆ Chen's research was supported in part by NSF under Grants CCF-0916606 and CCF-1217906.

⋆⋆ Corresponding author.

F. Dehne, R. Solis-Oba, and J.-R. Sack (Eds.): WADS 2013, LNCS 8037, pp. 244–255, 2013.

Table 1. Summary of ray-shooting data structures in polygonal domains

Data Structure	Preprocessing Time	Size	Query Time
[5,10]	$O(n\sqrt{h} + n\log n + h^{3/2}\log h)$	$O(n)$	$O(\sqrt{h}\log n)$
[17]	$O(n^2)$	$O(n^2)$	$O(\log n)$
[1]	$O((n\log n + h^2)\log h)$	$O((n+h^2)\log h)$	$O(\log^2 n \log^2 h)$
Our Result	$O(n + h^2 \cdot \mathrm{poly}(\log \mathrm{h}))$	$O(n+h^2)$	$O(\log n)$

Throughout this paper, we always let k denote $|Vis(q)|$ for any query point q. We say the *complexity* of a data structure is $O(f_1(\cdot), f_2(\cdot), f_3(\cdot))$ if its preprocessing time, size, and query time are $O(f_1(\cdot)), O(f_2(\cdot))$, and $O(f_3(\cdot))$, respectively.

Previous Work. For the ray-shooting query problem, Table 1 gives a summary. Our new data structure improves the previous work for small h. For simple polygons, ray-shooting data structures of $O(n, n, \log n)$ complexity have been proposed [5,6,9,10].

For the visibility query problem, previous work has been done on both the single simple polygon case and the polygonal domain case. For a single simple polygon, Bose *et al.* [4] proposed a data structure of complexity $O(n^3\log n, n^3, k + \log n)$. Aronov *et al.* [2] gave a smaller-size data structure with a little larger query time, with complexity $O(n^2\log n, n^2, k + \log^2 n)$. As indicated in [2], by using a ray-shooting data structure [5,10], a visibility query data structure of complexity $O(n, n, k\log n)$ is possible. For the polygonal domain case, Zarei and Ghodsi [21] gave a data structure of complexity $O(n^3\log n, n^3, k + \min\{h, k\}\log n)$, and Inkulu and Kapoor [12] obtained a data structure of complexity $O(n^2\log n, n^2,\ k + h + \min\{h, k\}\log^2 n)$. Another data structure in [12] has complexity related to the size of the visibility graph of the polygonal domain, which is $O(n^2)$; in the worst case, its complexity is $O(n^2h^3, n^2h^2, k\log n)$. Nouri and Ghodsi [16] gave a data structure of complexity $O(n^4\log n, n^4, k + \log n)$, and Lu *et al.* [15] presented a data structure of complexity $O(n^2\log n, n^2, k + \log^2 n + h\log(n/h))$. Table 2 summarizes these results for the polygonal domain case.

Comparing with the result in [21], our first data structure is $O(n)$ smaller in space and preprocessing time, but with an additive $O(\log^2 n)$ query time, which seems difficult to improve unless the query time of the data structure for the simple polygon case [2] can be reduced (because it has the same preprocessing time and space as our data structure). Comparing with the results in [12,15], our first data structure has the same processing time and space but with smaller query time. Our second data structure, comparing with the second one in [12], has the same query time but uses much less preprocessing time and space.

In addition, our results for visibility queries can be extended to *cone visibility queries* where, in addition to a query point q, a query also includes a cone with q as the apex that delimits the visibility of q. Our first visibility query data structure can be extended to this case with the same performances; for our

Table 2. Summary of visibility query data structures in polygonal domains. The value k is the output size of the visibility polygon of the query point.

Data Structure	Preprocessing Time	Size	Query Time
[21]	$O(n^3 \log n)$	$O(n^3)$	$O(k + \min\{h, k\} \log n)$
[12]	$O(n^2 \log n)$	$O(n^2)$	$O(k + h + \min\{h, k\} \log^2 n)$
[12]	$O(n^2 h^3)$	$O(n^2 h^2)$	$O(k \log n)$
[15]	$O(n^2 \log n)$	$O(n^2)$	$O(k + \log^2 n + h \log(n/h))$
Our Result 1	$O(n^2 \log n)$	$O(n^2)$	$O(k + \log^2 n + \min\{h, k\} \log n)$
Our Result 2	$O(n + h^2 \log h)$	$O(n + h^2)$	$O(k \log n)$

second one, the extended version has the same performances as before except that the preprocessing time becomes $O(n + h^2 \text{poly}(\log h))$.

Our Approaches. A corridor structure of polygonal domains has been used for solving shortest path problems [7,11,13], and later some new concepts like "bays", "canals", and "ocean" were introduced [8], which we refer to as the "extended corridor structure". In this paper, we also use the extended corridor structure [8], which partitions the free space into an ocean $\mathcal{M}$, bays, and canals. Each bay or canal is a simple polygon. The ocean $\mathcal{M}$ is multiply connected and its boundary consists of $O(h)$ convex chains. The extended data structure was used in [8] for computing the visibility polygon from a single line segment in polygonal domains. Unfortunately, the algorithm in [8] does not work for visibility queries. The techniques given in this paper focus on visibility queries. We process each bay/canal using data structures for simple polygons, and process $\mathcal{M}$ using data structures for convex obstacles. For example, for the visibility query problem, we build the data structure [2] for each bay/canal; for $\mathcal{M}$, we utilize the visibility complex [19,20]. For any query point q, $\mathit{Vis}(q)$ is obtained by consulting the data structures for $\mathcal{M}$ and for bays/canals.

Note that the corridor structure [13] was also used by the visibility query data structures in [12,15]; but, their approaches are quite different from ours. For example, they do not use the extended corridor structure (i.e., they do not use the ocean, bays, and canals). As shown later, our techniques not only yields better results but also makes the solutions quite simple.

In Section 2, we review the geometric structures of $\mathcal{P}$. We present our ray-shooting data structure in Section 3. In Section 4, we give our data structures for the visibility query problems. Due to the space limit, some proofs are omitted and can be found in the full version of this paper. For ease of exposition, we assume that no three obstacle vertices of $\mathcal{P}$ are collinear.

2 Preliminaries

For completeness of this paper, we briefly review the extended corridor structure [8]. Further, the rest of this paper relies heavily on the notation related to the

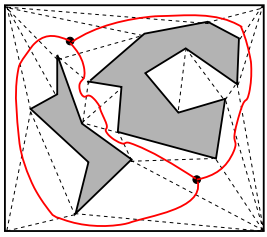

Fig. 1. Illustrating a triangulation of the free space among two obstacles and the corridors (with red solid curves). There are two junction triangles indicated by the large dots inside them, connected by three solid (red) curves. Removing the two junction triangles results in three corridors.

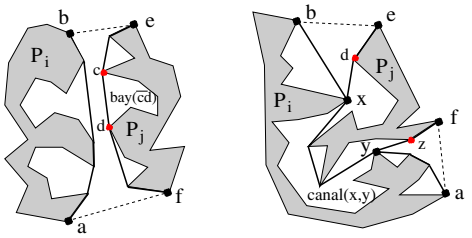

Fig. 2. Illustrating an open hourglass (left) and a closed hourglass (right) with a corridor path connecting the apices x and y of the two funnels. The dashed segments are diagonals. The paths $\pi(a, b)$ and $\pi(e, f)$ are marked by thick solid curves. A bay $bay(\overline{cd})$ with gate $\overline{cd}$ (left) and a canal $canal(x, y)$ with gates $\overline{xd}$ and $\overline{yz}$ (right) are also shown.

structure. For simplicity, we assume all obstacles in $\mathcal{P}$ are contained in a rectangle $\mathcal{R}$ (see Fig. 1), and we also view $\mathcal{R}$ as an obstacle in $\mathcal{P}$.

Let $\mathcal{F}$ denote the free space in $\mathcal{R}$, and $Tri(\mathcal{F})$ denote a triangulation of $\mathcal{F}$. Let $G(\mathcal{F})$ be the (planar) dual graph of $Tri(\mathcal{F})$. The degree of each node in $G(\mathcal{F})$ is at most three. Using $G(\mathcal{F})$, we compute a planar 3-regular graph, denoted by G^3 (the degree of each node in G^3 is three), possibly with loops and multi-edges, as follows. First, we remove every degree-one node from $G(\mathcal{F})$ together with its incident edge; repeat this process until no degree-one node remains in the graph. Second, remove every degree-two node from $G(\mathcal{F})$ and replace its two incident edges by a single edge; repeat this process until no degree-two node remains. The resulting graph is G^3 (see Fig. 1), which has $O(h)$ faces, nodes, and edges [13]. Each node of G^3 corresponds to a triangle in $Tri(\mathcal{F})$, which is called a *junction triangle* (see Fig. 1). The removal of all junction triangles results in $O(h)$ *corridors* (defined below), each of which corresponds to one edge of G^3.

The boundary of a corridor C consists of four parts (see Fig. 2): (1) A boundary portion of an obstacle $P_i \in \mathcal{P}$, from a point a to a point b; (2) a diagonal of a junction triangle from b to a boundary point e on an obstacle $P_j \in \mathcal{P}$ ($P_i = P_j$ is possible); (3) a boundary portion of the obstacle P_j from e to a point f; (4) a diagonal of a junction triangle from f to a. The corridor C is a simple polygon. Let $\pi(a, b)$ (resp., $\pi(e, f)$) be the shortest path from a to b (resp., e to f) inside C. The region H_C bounded by $\pi(a, b), \pi(e, f)$, and the two diagonals $\overline{be}$ and $\overline{fa}$ is called an *hourglass*, which is *open* if $\pi(a, b) \cap \pi(e, f) = \emptyset$ and *closed* otherwise (see Fig. 2). If H_C is open, then both $\pi(a, b)$ and $\pi(e, f)$ are convex chains and are called the *sides* of H_C; otherwise, H_C consists of two "funnels" [14] and a path $\pi_C = \pi(a, b) \cap \pi(e, f)$ joining the two apices of the two funnels, called the *corridor path* of C. Each funnel side is also a convex chain. We compute the hourglass of each corridor. The triangulation $Tri(\mathcal{F})$ can be computed

in $O(n \log n)$ time or $O(n + h \log^{1+\epsilon} h)$ time for any constant $\epsilon > 0$ [3]. After $Tr(\mathcal{F})$ is produced, computing all hourglasses takes $O(n)$ time.

Let $\mathcal{M}$ be the union of all $O(h)$ junction triangles, open hourglasses, and funnels. We call the space $\mathcal{M}$ the *ocean*. Note that $\mathcal{M} \subseteq \mathcal{F}$. Since the sides of open hourglasses and funnels are all convex, the boundary $\partial\mathcal{M}$ of $\mathcal{M}$ consists of $O(h)$ convex chains with totally $O(n)$ vertices; further, $\partial\mathcal{M}$ has $O(h)$ reflex vertices (with respect to $\mathcal{R} \setminus \mathcal{M}$). Thus, $\mathcal{R} \setminus \mathcal{M}$ can be partitioned into a set $\mathcal{P}'$ of $O(h)$ pairwise interior-disjoint convex polygons of totally $O(n)$ vertices [13] (e.g., by extending an angle-bisecting segment inward from each reflex vertex). If we view the convex polygons in $\mathcal{P}'$ as obstacles, then the ocean $\mathcal{M}$ is the free space with respect to $\mathcal{P}'$. The set $\mathcal{P}'$ can be obtained easily in $O(n + h \log h)$ time. It should be pointed out that our algorithms given later can be applied to $\mathcal{M}$ directly without explicitly computing the convex polygons in $\mathcal{P}'$. But for ease of exposition, we always discuss our algorithms on $\mathcal{P}'$ instead of on $\mathcal{M}$.

2.1 Bays and Canals

Recall that $\mathcal{M} \subseteq \mathcal{F}$. We examine the free space of $\mathcal{F}$ not in $\mathcal{M}$, i.e., $\mathcal{F} \setminus \mathcal{M}$, which consists of two types of regions: *bays* and *canals*, as defined below.

Consider the hourglass H_C of a corridor C. We first discuss the case when H_C is open (see Fig. 2). H_C has two sides. Let $S_1(H_C)$ be an arbitrary side of H_C. The obstacle vertices on $S_1(H_C)$ all lie on the same obstacle, say $P \in \mathcal{P}$. Let c and d be any two consecutive vertices on $S_1(H_C)$ such that the line segment $\overline{cd}$ is not an edge of P (see the left figure in Fig. 2, with $P = P_j$). The free region enclosed by $\overline{cd}$ and a boundary portion of P between c and d is called the *bay* of $\overline{cd}$ and P, denoted by $bay(\overline{cd})$, which is a simple polygon. We call $\overline{cd}$ the *bay gate* of $bay(\overline{cd})$, which is a common edge of $bay(\overline{cd})$ and $\mathcal{M}$.

If the hourglass H_C is closed, then let x and y be the two apices of its two funnels. Consider two consecutive vertices c and d on a side of a funnel such that $\overline{cd}$ is not an obstacle edge. If neither c nor d is a funnel apex, then c and d must lie on the same obstacle and the segment $\overline{cd}$ also defines a bay with that obstacle. However, if c or d is a funnel apex, say, $c = x$, then c and d may lie on different obstacles. If they lie on the same obstacle, then they also define a bay; otherwise, we call $\overline{xd}$ the *canal gate* at $x = c$ (see Fig. 2). Similarly, there is also a canal gate at the other funnel apex y, say $\overline{yz}$. Let P_i and P_j be the two obstacles bounding the hourglass H_C. The free region enclosed by P_i, P_j, and the two canal gates $\overline{xd}$ and $\overline{yz}$ that contains the corridor path of H_C is the *canal* of H_C, denoted by $canal(x, y)$, which is also a simple polygon.

Clearly, all bays and canals together constitute the space $\mathcal{F} \setminus \mathcal{M}$.

The fact that each bay has only one gate allows us to process a bay easily. Intuitively, an observer outside a bay cannot see any point outside the bay "through" its gate. But, each canal has two gates, which could cause trouble. The next lemma, proved in [8], gives an important property that an observer outside a canal cannot see any point outside the canal through the canal (and its two gates); we call it *the opaque property* of canals.

Lemma 1. [8] (The Opaque Property) *For any canal, suppose a line segment $\overline{pq}$ is in $\mathcal{F}$ (i.e., p is visible to q) such that neither p nor q is in the canal. Then $\overline{pq}$ cannot contain any point of the canal that is not on its two gates.*

3 The Ray-shooting Queries

We present our ray-shooting data structure in this section. We assume that we have already computed the ocean $\mathcal{M}$, and all bays and canals. We also assume the convex obstacle set $\mathcal{P}'$ is given. Recall that $\mathcal{M}$ is the free space among $\mathcal{P}'$. The preprocessing for these takes $O(n + h \log^{1+\epsilon} h)$ time.

Consider a ray $\sigma(q)$ with its origin $q \in \mathcal{F}$. Let q^* be the outcome of the ray-shooting query of $\sigma(q)$, i.e., q^* is the point on the input obstacles of $\mathcal{P}$ or on the boundary of $\mathcal{R}$ (denoted by $\partial\mathcal{R}$) that is hit first by $\sigma(q)$. We first show how to find q^*, and then discuss the preprocessing of our data structure. For simplicity of discussion, we assume the line containing the ray $\sigma(q)$ does not contain any obstacle vertex. Note that the origin q can be in $\mathcal{M}$, a bay, or a canal.

We first consider the case of $q \in \mathcal{M}$. If $\sigma(q)$ does not hit any obstacle of $\mathcal{P}'$ before it hits $\partial\mathcal{R}$, then the portion of $\sigma(q)$ inside $\mathcal{R}$ lies entirely in $\mathcal{M}$ and thus q^* is on $\partial\mathcal{R}$. Below, we assume $\sigma(q)$ hits an obstacle of $\mathcal{P}'$. Let p be the first point on the obstacles of $\mathcal{P}'$ hit by $\sigma(q)$. Based on our discussion in Section 2, each edge of any obstacle of $\mathcal{P}'$ is either an edge of an input obstacle of $\mathcal{P}$ or a bay/canal gate. If p is not on a gate of any bay/canal, then p is on an input obstacle of $\mathcal{P}$, and hence $q^* = p$. Otherwise, p is on a gate of a bay or a canal. If p is on the gate of a bay B, then since B has only one gate, q^* must be on the boundary of B (and thus on the boundary of an input obstacle of $\mathcal{P}$). If p is on a gate of a canal C, then although C has two gates, due to the opaque property of Lemma 1, q^* must be on the boundary of C that lies on an input obstacle.

If the origin q is in a bay/canal, then we find the first point p on the boundary of the bay/canal hit by $\sigma(q)$. If p is not on a gate, then $q^* = p$; otherwise, the ray $\sigma(q)$ goes out of the bay/canal and enters $\mathcal{M}$ through that gate, and we use a procedure as for the case of $q \in \mathcal{M}$ to compute q^*.

The discussion above shows that to compute q^*, we only need to conduct at most three ray-shooting queries each of which is either on a bay/canal or on the convex obstacle set $\mathcal{P}'$. We perform the preprocessing accordingly. For a bay/canal, because it is a simple polygon, we build a data structure for simple polygons [5,10] for it. Since the total number of vertices of all bays and canals is $O(n)$, preprocessing all bays and canals takes $O(n)$ time and space, and each query inside a bay/canal takes $O(\log n)$ time.

For the convex obstacle set $\mathcal{P}'$, Pocchiola and Vegter [18] showed that by using the visibility complex, a data structure of $O(n + k')$ size can be built in $O(n + k' \cdot \text{poly}(\log h))$ time that allows to answer each ray-shooting query in $O(\log n)$ time, where $k' = O(h^2)$ is the number of common tangents of the convex obstacles in $\mathcal{P}'$ that lie in the free space of $\mathcal{P}'$ (i.e., $\mathcal{M}$).

In summary, we have the following result.

Theorem 1. *For an input polygonal domain $\mathcal{P}$, we can build a data structure of size $O(n + h^2)$ in $O(n + h^2 \cdot \text{poly}(\log h))$ preprocessing time that allows to answer each ray-shooting query in $O(\log n)$ time.*

4 The Visibility Queries

In this section, we present our two visibility query data structures. We assume that the ocean $\mathcal{M}$, and all bays and canals have been computed, and the convex obstacle set $\mathcal{P}'$ is given. The needed preprocessing takes $O(n + h \log^{1+\epsilon} h)$ time. We begin with the first data structure, described in Sections 4.1, 4.2, and 4.3. The second data structure is shown in Section 4.4, which uses some ingredients of the first data structure.

For a query point q, we seek to compute the visibility polygon $Vis(q)$. For simplicity of discussion, assume q is not collinear with any two obstacle vertices.

To provide some intuition, in Section 4.1, we sketch an algorithmic procedure for computing $Vis(q)$ without any preprocessing, and argue its correctness. Our query algorithm (with preprocessing) given later will follow this procedure. In Section 4.2, we present the preprocessing of our first data structure. Its query algorithm and time analysis are shown in Section 4.3.

4.1 The Algorithm for Computing $Vis(q)$

The query point q may be in $\mathcal{M}$, a bay, or a canal. We start with the case of $q \in \mathcal{M}$. For any subset S of the free space $\mathcal{F}$, let $Vis(q, S)$ denote the intersection of $Vis(q)$ and S. For example, $Vis(q, \mathcal{M})$ is the subpolygon of $Vis(q)$ in the ocean $\mathcal{M}$, and $Vis(q, \mathcal{F})$ is $Vis(q)$.

We first compute $Vis(q, \mathcal{M})$. Because the space $\mathcal{F} \setminus \mathcal{M}$ consists of all bays and canals, the region $Vis(q) \setminus Vis(q, \mathcal{M})$ is the union of the visibility subpolygons of $Vis(q)$ in all bays and canals. Next, we show how to compute $Vis(q) \setminus Vis(q, \mathcal{M})$.

Observation 1. *For $q \in \mathcal{M}$, if a bay/canal does not have any gate that intersects with the boundary of $Vis(q, \mathcal{M})$, then no point in that bay/canal is visible to q.*

Proof. Consider any point p in a bay $bay(\overline{cd})$. Suppose p is visible to q. Since $q \in \mathcal{M}$, $\overline{pq}$ must intersect the gate $\overline{cd}$, say at a point p'. Hence, p' is visible to q. Because $\overline{cd}$ is on $\partial\mathcal{M}$, p' is on the boundary of $Vis(q, \mathcal{M})$. Thus $\overline{cd}$ intersects the boundary of $Vis(q, \mathcal{M})$. The case for canals can be proved similarly.

Suppose for a bay $bay(\overline{cd})$ with gate $\overline{cd}$, we want to compute $Vis(q, bay(\overline{cd}))$. If its gate $\overline{cd}$ does not intersect the boundary $\partial Vis(q, \mathcal{M})$ of $Vis(q, \mathcal{M})$, then by Observation 1, $Vis(q, bay(\overline{cd})) = \emptyset$. If $\overline{cd}$ has a single sub-segment on $\partial Vis(q, \mathcal{M})$, then q can see part of $bay(\overline{cd})$ through the cone delimited by this sub-segment and with q as the apex, and we compute $Vis(q, bay(\overline{cd}))$ "seeing through" this cone. The general case is when multiple disjoint sub-segments of $\overline{cd}$ are on $\partial Vis(q, \mathcal{M})$ (e.g., see Fig. 3). In this case, some interior points of $bay(\overline{cd})$ are visible to q

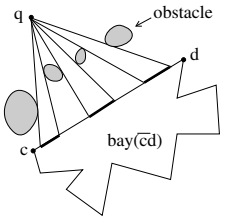

Fig. 3. Three sub-segments (the thick ones) of $\overline{cd}$ are visible to the point q

through multiple cones. We compute the visible region of q in $bay(\overline{cd})$ for each such cone. It is easy to see that the visible regions for these cones are mutually disjoint. Therefore, $Vis(q, bay(\overline{cd}))$ is the union of them.

Next, suppose for a canal $canal(x, y)$ with two gates $\overline{xd}$ and $\overline{yz}$ (as in Fig. 2), we want to compute $Vis(q, canal(x, y))$. Similarly, if its two gates do not intersect $\partial Vis(q, \mathcal{M})$, then $Vis(q, canal(x, y)) = \emptyset$. Otherwise, let $Vis(\overline{xd})$ denote the region of $canal(x, y)$ visible to q through the gate $\overline{xd}$ and $Vis(\overline{yz})$ denote the region of $canal(x, y)$ visible to q through the gate $\overline{yz}$. Clearly, $Vis(q, canal(x, y)) = Vis(\overline{xd}) \cup Vis(\overline{yz})$. We compute $Vis(\overline{xd})$ and $Vis(\overline{yz})$ separately using our above approach for the bay case. Note that $x = y$ is possible, in which case the two gates share a common vertex x but we view x as belonging only to $\overline{xd}$ (i.e., $\overline{yz}$ is viewed as a half-open segment). In this way, the two gates never intersect. Lemma 2 below shows that $Vis(\overline{xd})$ and $Vis(\overline{yz})$ are mutually disjoint. Thus, once $Vis(\overline{xd})$ and $Vis(\overline{yz})$ are available, computing $Vis(\overline{xd}) \cup Vis(\overline{yz})$ is trivial. The proof of Lemma 2 is omitted.

Lemma 2. *For $q \in \mathcal{M}$, the visibility polygons $Vis(\overline{xd})$ and $Vis(\overline{yz})$ in $canal(x, y)$ do not intersect with each other.*

Based on the above, after we obtain $Vis(q, \mathcal{M})$, to compute $Vis(q) \setminus Vis(q, \mathcal{M})$, we can simply check the boundary $\partial Vis(q, \mathcal{M})$. For each sub-segment of a bay/canal gate on $\partial Vis(q, \mathcal{M})$, we compute the region in the bay/canal visible to q through that sub-segment. All these regions are pairwise disjoint and $Vis(q) \setminus Vis(q, \mathcal{M})$ is a trivial union of them. We hence finish the discussion of our procedure for computing $Vis(q)$ in the case of $q \in \mathcal{M}$.

Next, we consider the case when the query point q is in a bay, say $bay(\overline{cd})$. In this case, we first compute the visibility polygon of q in $bay(\overline{cd})$, i.e., $Vis(q, bay(\overline{cd}))$. If the gate $\overline{cd}$ does not intersect the boundary of $Vis(q, bay(\overline{cd}))$, then $Vis(q) = Vis(q, bay(\overline{cd}))$ because q is not visible to any point outside $bay(\overline{cd})$. Otherwise, there must be a single (maximal) sub-segment of $\overline{cd}$ on the boundary of $Vis(q, bay(\overline{cd}))$ through which q can see the outside of $bay(\overline{cd})$ (in the cone delimited by that sub-segment). In other words, $Vis(q) \setminus Vis(q, bay(\overline{cd}))$ is the visible region in the space $\mathcal{F} \setminus bay(\overline{cd})$ visible to q through the cone. To compute $Vis(q) \setminus Vis(q, bay(\overline{cd}))$, we use a procedure similar to that for the case of $q \in \mathcal{M}$. The difference is that here the visibility is through a cone.

The remaining case is when the query point q is in a canal. This case is very similar to the bay case above. The difference is that we consider the two canal gates separately, using the procedure for the bay case. We omit the details.

4.2 The Preprocessing

We discuss the preprocessing for our algorithm in Section 4.1, in which we need to compute the visibility polygons of q in $\mathcal{M}$ or in a bay/canal.

We first discuss the preprocessing for computing $Vis(q, \mathcal{M})$ when $q \in \mathcal{M}$. Recall that we have a set $\mathcal{P}'$ of $O(h)$ convex obstacles of totally $O(n)$ vertices and its free space is the ocean $\mathcal{M}$. By using the visibility complex [19,20], we have the following lemma with proof omitted.

Lemma 3. *We can build a data structure of size $O(n + h^2)$ in $O(n + h^2 \log h)$ time that allows to compute $Vis(q, \mathcal{M})$ in $O(|Vis(q, \mathcal{M})| + h' \log n)$ time for any query point $q \in \mathcal{M}$, where h' is the number of obstacles in $\mathcal{P}'$ visible to q.*

Further, recall that in our algorithm discussed in Section 4.1, when the query point q is in a bay (or canal), $Vis(q, \mathcal{M})$ is the visible region of q in $\mathcal{M}$ through a cone (or a sub-segment of the bay gate). Therefore, we need to deal with the cone visibility in $\mathcal{M}$. For this, we extend the result in Lemma 3.

Corollary 1. *We can build a data structure of size $O(n+h^2)$ in $O(n+h^2 \log h)$ time that allows to compute $Vis(q, \mathcal{M})$ in $O(|Vis(q, \mathcal{M})| + h' \log n)$ time for any query point q in a bay or canal within its visibility cone, where h' is the number of obstacles in $\mathcal{P}'$ visible to q.*

Next, we discuss the preprocessing for bays and canals. Recall that there are two types of query situations on a bay/canal. The first type is that the query point q is inside a bay/canal and we need to compute the visibility polygon of q in that bay/canal. The second type is that q is outside a bay/canal along with a sub-segment of a gate of that bay/canal and we need to compute the visibility polygon of q in the bay/canal through that sub-segment.

For the first type, we simply use the data structure by Aronov *et al.* [2] for simple polygons. Since all bays and canals have totally at most n vertices, the preprocessing time and space for all bays and canals are $O(n^2 \log n)$ and $O(n^2)$, respectively. After that, for any query point q in a bay/canal, the visibility polygon P of q in the bay/canal can be computed in $O(\log^2 n + |P|)$ time.

For the second type, we do the following preprocessing. Consider a convex obstacle $P \in \mathcal{P}'$. Let $BayCanal(P)$ (or $BC(P)$) denote the set of bays and canals each of which has a gate lying on the boundary of P. For any query point $q \notin P$, let $\mathcal{C}_q$ be a cone with apex q. Denote by $Vis(q, BC(P))$ the union of the visibility polygons of q in all bays and canals of $BC(P)$, and here all other obstacles in $\mathcal{P}'$ are ignored (i.e., we assume they are transparent and do not block the view of q). Let $Vis(\mathcal{C}_q, BC(P)) = \mathcal{C}_q \cap Vis(q, BC(P))$, i.e., $Vis(\mathcal{C}_q, BC(P))$ is the union of the visibility polygons of q in all bays and canals of $BC(P)$ through the cone $\mathcal{C}_q$. Using the techniques in [2], we have Lemma 4, with proof omitted.

Lemma 4. *For a convex obstacle P, suppose the total number of vertices in all bays and canals of $BC(P)$ is m. We can build a data structure of size $O(m^2)$ in $O(m^2 \log m)$ time such that for any query point $q \notin P$, in $O(\log m)$ time, we can obtain (a pointer to) a data structure storing $Vis(q, BC(P))$, and if needed, report $Vis(q, BC(P))$ explicitly in additional $O(|Vis(q, BC(P))|)$ time. Further, given any cone $\mathcal{C}_q$ with apex q, from the above data structure, we can obtain $Vis(\mathcal{C}_q, BC(P))$ in additional $O(\log m + |Vis(\mathcal{C}_q, BC(P))|)$ time.*

We compute the data structure for Lemma 4 for each convex obstacle in $\mathcal{P}'$. Since all bays and canals have $O(n)$ vertices, the total preprocessing time is $O(n^2 \log n)$ and the space is $O(n^2)$.

In summary, our preprocessing includes: (1) preprocessing $\mathcal{M}$ (or $\mathcal{P}'$) using Lemma 3 and Corollary 1, (2) preprocessing all bays and canals for the first type query situation using the data structure in [2], and (3) preprocessing all bays and canals for the second type query situation using Lemma 4. The overall preprocessing time is $O(n^2 \log n)$ and the space is $O(n^2)$.

4.3 The Query Algorithm

Consider a query point q. Our query algorithm for computing $Vis(q)$ follows the same procedure as given in Section 4.1. We first discuss the case of $q \in \mathcal{M}$.

In Step (1), we compute $Vis(q, \mathcal{M})$ using the data structure for Lemma 3, which takes $O(|Vis(q, \mathcal{M})| + h' \log n)$ time. In Step (2), for each obstacle $P \in \mathcal{P}'$ visible to q, by Lemma 4, we obtain the data structure for storing $Vis(q, BC(P))$, in $O(\log n)$ time. In Step (3), we check the boundary of $Vis(q, \mathcal{M})$; for every obstacle P visible to q, if q's view of P is blocked partially by some other obstacles of $\mathcal{P}'$, i.e., there are some cones through which q is visible to one or more portions of P, then for each such cone $\mathcal{C}_q$, by Lemma 4, we compute $Vis(\mathcal{C}_q, BC(P))$ in additional $O(\log n + |Vis(\mathcal{C}_q, BC(P))|)$ time. Then, $Vis(q)$ is obtained and is represented as a cyclically ordered list of visible edges and vertices. The correctness of the algorithm follows from our discussion in Section 4.1.

To analyze the query time, let $k = |Vis(q)|$. First, $|Vis(q, \mathcal{M})|$ plus the sum of all $|Vis(\mathcal{C}_q, BC(P))|$'s is $O(k)$. Second, the number of cones in Step (3) is $O(h')$ because only h' obstacles of $\mathcal{P}'$ are visible to q. Therefore, the overall time of the query algorithm is $O(k + h' \log n)$. Clearly, $h' \leq h$ and $h' \leq k$.

Next, we discuss the case when q is in a bay, say $bay(\overline{cd})$. In Step (1), we compute the visibility polygon $Vis(q, bay(\overline{cd}))$ in $bay(\overline{cd})$, in $O(\log^2 n + |Vis(q, bay(\overline{cd}))|)$ time using the data structure in [2]. If $\overline{cd}$ has a sub-segment $\overline{c'd'}$ on the boundary of $Vis(q, bay(\overline{cd}))$, then in Step (2), we compute the visibility polygon of q outside $bay(\overline{cd})$ seeing through the cone with apex q and delimited by $\overline{c'd'}$. This step and the rest of the algorithm are basically the same as the former case of $q \in \mathcal{M}$. One difference is that we use Corollary 1 instead of Lemma 3 to compute $Vis(q, \mathcal{M})$. Similarly to the analysis above, the overall query time is $O(\log^2 n + k + \min\{k, h\} \log n)$. Note that we have an additive $O(\log^2 n)$ time due to using the data structure for simple polygons [2].

The remaining case when q is in a canal is the same as the bay case except that we process the two canal gates separately. The time of each query is also $O(\log^2 n + k + \min\{k, h\} \log n)$. In summary, we have the following result.

Theorem 2. *For a polygonal domain $\mathcal{P}$, we can build a data structure of size $O(n^2)$ in $O(n^2 \log n)$ preprocessing time that can answer each visibility query in $O(\log^2 n + k + \min\{k, h\} \log n)$ time.*

4.4 The Second Data Structure

The main difference between our second data structure and the first one is that for bays and canals, we do not preprocess them using the data structures in [2] and Lemma 4. Instead, we build a ray-shooting data structure in simple polygons [5,10] for each bay and canal, which takes totally $O(n)$ preprocessing time and space. But, we still keep the data structures for Lemma 3 and Corollary 1. The overall preprocessing time and space then become $O(n+h^2 \log h)$ and $O(n+h^2)$, respectively. Below, we discuss the query algorithm.

Consider a query point q. We first discuss the case of $q \in \mathcal{M}$. In the first step, we still compute $\mathit{Vis}(q, \mathcal{M})$ by Lemma 3. In the second step, we check the boundary of $\mathit{Vis}(q, \mathcal{M})$. If a sub-segment of a bay/canal gate appears on $\partial \mathit{Vis}(q, \mathcal{M})$, then we use the ray-shooting approach [2] to compute the visibility polygon of q in the bay/canal through that sub-segment, which takes $O(k' \log n)$ time, where k' is the output size of this visibility polygon. For the query time, the first step takes $O(|\mathit{Vis}(q, \mathcal{M})| + h' \log n)$ time. Again, $h' = O(k)$. For the second step, clearly, the sum of all such k' terms is $O(k)$. Therefore, the query time is $O(k \log n)$. The other cases when q is in a bay or canal are very similar and we omit the discussions of them. In summary, we have the following result.

Theorem 3. *For a polygonal domain $\mathcal{P}$, we can build a data structure of size $O(n + h^2)$ in $O(n + h^2 \log h)$ preprocessing time that can answer each visibility query in $O(k \log n)$ time.*

5 Conclusions

In this paper we propose new data structures for ray-shooting queries and computing visibility polygons for query points in polygonal domains, which benefit in a large part from the extended corridor structure [8]. It would be interesting to see whether further improvements are possible. In addition, the current best visibility query data structures on simply polygons have complexities $O(n^3 \log n, n^3, k + \log n)$ and $O(n^2 \log n, n^2, k + \log^2 n)$, respectively; improving these results would also be interesting, and in particular, an open question is whether $O(n^2 \log n, n^2, k + \log n)$ complexity data structures exist.

Acknowledgments. The authors would like to thank Tiancong Chen for helpful discussions in early phases of this work.

References

1. Agarwal, P., Sharir, M.: Ray shooting amidst convex polygons in 2D. Journal of Algorithms 21(3), 508–519 (1996)
2. Aronov, B., Guibas, L., Teichmann, M., Zhang, L.: Visibility queries and maintenance in simple polygons. Discrete and Computational Geometry 27(4), 461–483 (2002)
3. Bar-Yehuda, R., Chazelle, B.: Triangulating disjoint Jordan chains. International Journal of Computational Geometry and Applications 4(4), 475–481 (1994)
4. Bose, P., Lubiw, A., Munro, J.: Efficient visibility queries in simple polygons. Computational Geometry: Theory and Applications 23(3), 313–335 (2002)
5. Chazelle, B., Edelsbrunner, H., Grigni, M., Gribas, L., Hershberger, J., Sharir, M., Snoeyink, J.: Ray shooting in polygons using geodesic triangulations. Algorithmica 12(1), 54–68 (1994)
6. Chazelle, B., Guibas, L.: Visibility and intersection problems in plane geometry. Discrete and Computational Geometry 4, 551–589 (1989)
7. Chen, D.Z., Wang, H.: A nearly optimal algorithm for finding L_1 shortest paths among polygonal obstacles in the plane. In: Demetrescu, C., Halldórsson, M.M. (eds.) ESA 2011. LNCS, vol. 6942, pp. 481–492. Springer, Heidelberg (2011)
8. Chen, D.Z., Wang, H.: Computing the visibility polygon of an island in a polygonal domain. In: Czumaj, A., Mehlhorn, K., Pitts, A., Wattenhofer, R. (eds.) ICALP 2012, Part I. LNCS, vol. 7391, pp. 218–229. Springer, Heidelberg (2012)
9. Guibas, L., Hershberger, J., Leven, D., Sharir, M., Tarjan, R.: Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. Algorithmica 2(1-4), 209–233 (1987)
10. Hershberger, J., Suri, S.: A pedestrian approach to ray shooting: Shoot a ray, take a walk. Journal of Algorithms 18(3), 403–431 (1995)
11. Inkulu, R., Kapoor, S.: Planar rectilinear shortest path computation using corridors. Computational Geometry: Theory and Applications 42(9), 873–884 (2009)
12. Inkulu, R., Kapoor, S.: Visibility queries in a polygonal region. Computational Geometry: Theory and Applications 42(9), 852–864 (2009)
13. Kapoor, S., Maheshwari, S., Mitchell, J.: An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. Discrete and Computational Geometry 18(4), 377–383 (1997)
14. Lee, D., Preparata, F.: Euclidean shortest paths in the presence of rectilinear barriers. Networks 14(3), 393–410 (1984)
15. Lu, L., Yang, C., Wang, J.: Point visibility computing in polygons with holes. Journal of Information and Computational Science 8(16), 4165–4173 (2011)
16. Nouri, M., Ghodsi, M.: Space–query-time tradeoff for computing the visibility polygon. In: Deng, X., Hopcroft, J.E., Xue, J. (eds.) FAW 2009. LNCS, vol. 5598, pp. 120–131. Springer, Heidelberg (2009)
17. Pocchiola, M.: Graphics in flatland revisited. In: Gilbert, J.R., Karlsson, R. (eds.) SWAT 1990. LNCS, vol. 447, pp. 85–96. Springer, Heidelberg (1990)
18. Pocchiola, M., Vegter, G.: Pseudo-triangulations: Theory and applications. In: Proc. of the 12th Annual Symposium on Computational Geometry, pp. 291–300 (1996)
19. Pocchiola, M., Vegter, G.: Topologically sweeping visibility complexes via pseudo-triangulations. Discrete and Computational Geometry 16(4), 419–453 (1996)
20. Pocchiola, M., Vegter, G.: The visibility complex. International Journal of Computational Geometry and Applications 6(3), 279–308 (1996)
21. Zarei, A., Ghodsi, M.: Query point visibility computation in polygons with holes. Computational Geometry: Theory and Applications 39(2), 78–90 (2008)