# A NEW ALGORITHM FOR COMPUTING VISIBILITY GRAPHS OF POLYGONAL OBSTACLES IN THE PLANE

*Danny Z. Chen*[*] *and Haitao Wang*[†]

Abstract. Given a set of $h$ pairwise disjoint polygonal obstacles with a total of $n$ vertices in the plane, the vertex-vertex visibility graph is an undirected graph whose nodes are vertices of the obstacles and whose edges are pairs of visible vertices. Ghosh and Mount gave a well-known $O(n \log n + k)$ time algorithm for computing the visibility graph, where $k$ is the size of the graph. Later Kapoor and Maheshwari proposed an $O(T + n + h \log h + k)$ time algorithm for the problem, where $T$ is the time for triangulating the free space. In this paper, by incorporating an extended corridor structure into Ghosh and Mount's algorithm, we provide an alternative approach for computing the visibility graph in $O(T + n + h \log h + k)$ time. Like Ghosh and Mount's algorithm, our algorithm can also compute several useful structures such as the funnel structure and the enhanced visibility graph, which have many other applications such as computing the vertex-edge visibility graphs, the edge-edge visibility graphs, the extended vertex-vertex visibility graphs, and the visibility polygons of obstacle vertices, etc.

## 1 Introduction

Given a set of $h$ pairwise disjoint polygonal obstacles, $\mathcal{P} = \{P_1, P_2, \ldots, P_h\}$, with a total of $n$ vertices in the plane, the space minus the interior of all obstacles is called the *free space*. Each obstacle $P_i$ of $\mathcal{P}$ is an arbitrary simple polygon. We say two points in the plane are *visible* to each other if the open line segment between them lies in the free space. Two objects in the plane are *visible* to each other if a point of one object is visible to a point of the other object (this is usually called *weakly visible* in the literature; we use *visible* here for the sake of simplicity).

The *vertex-vertex visibility graph* of $\mathcal{P}$ is defined to be an undirected graph whose nodes are the vertices of the obstacles and whose edges are pairs of visible vertices. Similarly, the *vertex-edge* visibility graph of $\mathcal{P}$ is an undirected graph whose nodes are the vertices and edges of the obstacles, and a vertex and an obstacle edge are adjacent in the graph if the vertex is visible to the obstacle edge; the *edge-edge* visibility graph of $\mathcal{P}$ is an undirected graph whose nodes are the edges of the obstacles, and two obstacle edges are adjacent in the graph if one edge is visible to the other edge. The sizes of these three graphs are asymptotically equal [10]. In this paper, we use $k$ to denote the size of such a graph.

[*] *Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA*, dchen@nd.edu

[†] Corresponding author. *Department of Computer Science, Utah State University, Logan, UT 84322, USA*, haitao.wang@usu.edu

Computing visibility graphs is a fundamental problem in computational geometry and has been studied extensively. For the vertex-vertex visibility graph, $O(n^2 \log n)$ time algorithms were obtained by Lee [18] and Sharir and Schorr [21]. Asano et al. [1] and Welzl [23] gave $O(n^2)$ time algorithms, which are worst-case optimal since $k = \Theta(n^2)$ in the worst case. Overmars and Welzl [20] developed an $O(k \log n)$ time algorithm. An output-sensitive $O(n \log n + k)$ time algorithm was presented by Ghosh and Mount [10], which we call the GM algorithm. The GM algorithm computes a funnel structure that is quite powerful (it actually computes an "enhanced visibility graph"). For example, it was shown that the algorithm in [10] can be extended to compute the vertex-edge and edge-edge visibility graphs, each in $O(n \log n + k)$ time. Note that another $O(n \log n + k)$ time algorithm for computing the edge-edge visibility graphs is given in [17]. With the linear time triangulation algorithm [3], Hershberger [12] gave an $O(n+K)$ time algorithm for computing the vertex-vertex visibility graph in a simple polygon, where $K$ is the size of the graph.

In this paper, we present an $O(T+n+h \log h+k)$ time algorithm for computing the vertex-vertex visibility graph of $\mathcal{P}$, where $T$ is the time for computing an arbitrary triangulation of the free space. The current best triangulation algorithms run in either $O(n \log n)$ time or $O(n + h \log^{1+\epsilon} h)$ time [2] for an arbitrarily small constant $\epsilon > 0$. Therefore, $T = O(\min\{n \log n, n + h \log^{1+\epsilon} h\})$. Hence, our algorithm reduces the additive $O(n \log n)$ time factor in the GM algorithm to $O(T + n + h \log h)$.

As in [10], our algorithm can compute the powerful funnel structure and enhanced visibility graph, and thus can be extended to compute the vertex-edge and edge-edge visibility graphs, each in the same amount of time as above. Also, using the funnel structure, for any obstacle vertex, our algorithm can compute its visibility polygon in $O(m)$ time, where $m$ is the size of the visibility polygon. Note that a *visibility polygon* of a point $p$ is the set of all points in the free space that are visible to $p$. In addition, our algorithm can further be extended to compute the *extended vertex-vertex visibility graph* of $\mathcal{P}$ in the same amount of time, which contains not only the vertex-vertex visibility graph but also for each edge of the visibility graph its two *extension endpoints* on the obstacles of $\mathcal{P}$ (or in infinity) when extending the edge along both directions until it hits some obstacles (or infinity). The extended vertex-vertex visibility graph is useful in many applications, e.g., computing minimum link paths [19] and the weak visibility polygon of a line segment [22] among polygonal obstacles.

It should be noted that Kapoor and Maheshwari [15] have given an algorithm for computing the vertex-vertex visibility graph in $O(T + n + h \log h + k)$ time. Although its running time is essentially the same as ours, our approach is quite different from theirs and our algorithm deserves attention due to the following reasons. First, since the visibility graph is very fundamental, it may be worthwhile to have an alternative solution. Second, unlike Kapoor and Maheshwari's algorithm [15], which is quite different from the well-known GM algorithm [10], our algorithm basically follows the scheme of the GM algorithm [10]. Thus, readers who are familiar with the GM algorithm will find our algorithm relatively easy to follow. Third, as the GM algorithm, our algorithm can also compute other useful structures efficiently, e.g., the funnel structure, the enhanced visibility graph, the vertex-edge and edge-edge visibility graphs, the extended vertex-vertex visibility graph, and the visibility polygons of obstacle vertices, as discussed above.

Henceforth, unless otherwise indicated, our discussion will focus on the vertex-vertex visibility graph and we simply use the term "visibility graph" for it.

### 1.1 Our Approach

We use the idea of computing the funnel structure as the GM algorithm [10]. The additive $O(n \log n)$ time factor in the GM algorithm is due to its use of a particular plane-sweeping triangulation algorithm [13] on the free space (an arbitrary triangulation does not work for the GM algorithm). We give a new approach that avoids the $O(n \log n)$ time plane-sweeping triangulation but still computes the funnel structure as the GM algorithm.

Denote by $\mathcal{G}$ the sought visibility graph of $\mathcal{P}$, and by $\mathcal{F}$ the free space of $\mathcal{P}$. We first compute an arbitrary triangulation of $\mathcal{F}$ in $O(T)$ time, from which we build an extended corridor structure [4, 5, 7, 6, 8]. The corridor structure was originally used to solve shortest path problems (e.g., [14, 16]), and later new concepts like "ocean", "bays", and "canals" have been introduced [4, 5, 7, 6, 8], which is referred to as the "extended corridor structure". This structure partitions $\mathcal{F}$ into a space $\mathcal{M}$ (called *ocean*) and other regions, called *bays* and *canals*. While the ocean $\mathcal{M}$ may be multiply connected, every bay or canal is a simple polygon. Each bay has a common boundary edge with $\mathcal{M}$, and each canal has two common boundary edges with $\mathcal{M}$. But a bay or canal does not share any edge with another bay or canal. The common edges of bays (resp., canals) and $\mathcal{M}$ are called the *gates*. Thus, each bay has one gate and each canal has two gates. All vertices on the boundaries of $\mathcal{M}$ and the bays/canals are obstacle vertices of $\mathcal{P}$.

An important property is that the boundary of the ocean $\mathcal{M}$ consists of $O(h)$ convex chains with a total of $O(n)$ vertices. This allows us to apply Hertel and Mehlhorn's plane-sweeping triangulation algorithm [13] to triangulate the space $\mathcal{M}$ in $O(n + h \log h)$ time. The resulting triangulation of $\mathcal{M}$ is sufficient for the GM algorithm to compute the visibility graph of the vertices of the boundary of $\mathcal{M}$ with respect to the space $\mathcal{M}$. But, this graph is only a subgraph of $\mathcal{G}$. To construct $\mathcal{G}$ fully, we must also consider the vertices in the space of $\mathcal{F} \setminus \mathcal{M}$, i.e., all bays and canals. To this end, we perform the following preprocessing on all bays and canals. For each bay or canal $P$, since it is a simple polygon, by modifying Hershberger's algorithm [12], we build the funnel structure and the enhanced visibility graph for $P$ in $O(k')$ time, where $k'$ is the size of $P$'s visibility graph. Further, the enhanced visibility graphs we build for the bays and canals have some special properties that are needed by our main algorithm. Processing all bays and canals as above takes $O(n+k)$ time in total.

Our main algorithm is guided by the plane-sweeping triangulation algorithm [13] on the ocean $\mathcal{M}$. During the plane-sweeping, we follow a scheme similar to the GM algorithm, but a key difference is that when a bay/canal gate is a side of the current triangle under consideration, we use a *connection procedure* to handle the interaction between the funnel structure of the bay/canal and that of the subregion of $\mathcal{M}$ that has been triangulated, which occurs through the gate of that bay/canal. In this way, after the triangulation of $\mathcal{M}$ is finished, the funnel structure and the enhanced visibility graph for the entire free space $\mathcal{F}$ will be constructed.

In a nutshell, our approach can be viewed as a novel combination of the funnel structure based GM algorithm [10] and Hershberger's algorithm [12] for simple polygons, based on a corridor partition of the free space $\mathcal{F}$. Comparing with the GM algorithm, our algorithm has four main differences: (1) We partition the free space $\mathcal{F}$ into the ocean $\mathcal{M}$ and bays/canals; (2) our algorithm performs a preprocessing step on all bays and canals; (3) our plane-sweeping triangulation is only on the ocean $\mathcal{M}$ (i.e., a subregion of $\mathcal{F}$); (4) our main algorithm calls a collection of connection procedures to merge the visibility graph of $\mathcal{M}$ with those of the bays and canals through every bay/canal gate. We show that the total time of calling all connection procedures in the entire algorithm is $O(n+k)$.

Note that Kapoor and Maheshwari's algorithm [15] also uses the corridor structure, but our algorithm is quite different from theirs. For example, they do not use the extended corridor structure, i.e., they do not partition the free space $\mathcal{F}$ into $\mathcal{M}$ and the bays and canals, and they do not compute the funnel structure.

For ease of exposition only, as in [10] we make a general position assumption that no three obstacle vertices are collinear and no two vertices lie on the same vertical or horizontal line. To distinguish the edges of the obstacles in $\mathcal{P}$ from the edges of the visibility graph $\mathcal{G}$, we often refer to the former as *obstacle edges* and the latter as *graph edges* or *visible segments*. The vertices of the obstacles in $\mathcal{P}$ are also called *obstacle vertices*.

The rest of the paper is organized as follows. In Section 2, we briefly review the GM algorithm [10], in particular, the funnel structure and the enhanced visibility graph. In Section 3, we discuss the extended corridor structure and give some observations. In Section 4, we do the preprocessing for all bays and canals. The main algorithm is given in Section 5. Section 6 concludes with remarks on the extension of our algorithm to other problems, e.g., the vertex-edge, edge-edge, and extended vertex-vertex visibility graphs.

## 2 Preliminaries

In this section, we briefly review the GM algorithm [10], in particular, the funnel structure and the enhanced visibility graph.

### 2.1 The Funnel Structure

For two points $a$ and $b$, let $\overline{ab}$ denote the line segment connecting them, and depending on the context this segment may have a direction from $a$ to $b$.

We first define funnels. Let $\overline{xy}$ be a directed obstacle edge (i.e., it has a direction from $x$ to $y$) such that the free space is on its left. Without loss of generality, assume $x$ is lower than $y$. Let $v$ be an obstacle vertex to the left of $\overline{xy}$ (see Fig. 1). Suppose $v$ is visible to some point $z$ on $\overline{xy}$. A funnel can be formed by considering the paths $\overline{vz} \cup \overline{zx}$ and $\overline{vz} \cup \overline{zy}$ as two rubber bands and allowing them to snap to their natural shape, e.g., see Fig. 1(a). These bands then form two convex chains, one from $v$ to $x$, called the *lower chain*, and the other from $v$ to $y$, called the *upper chain*. The obstacle-free simple polygon bounded by the lower chain, upper chain, and $\overline{xy}$ is called a *funnel* whose *apex* is $v$ and whose *base* is $\overline{xy}$.

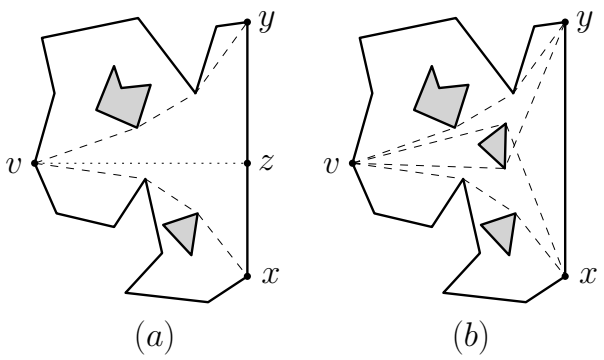


Figure 1: Illustrating some funnels: The apices are $v$ and the bases are $\overline{xy}$.

Note that the same obstacle vertex and edge may define many different funnels (see Fig. 1(b)). A funnel can be uniquely determined by the first segment of its lower chain (or its upper chain). To see this, suppose $u_0, u_1, \ldots, u_m$ are the clockwise sequence of obstacle vertices visible to $v$ such that $\overline{vu_0}$ and $\overline{vu_m}$ are the two obstacle edges incident to $v$. Then for any adjacent pair $u_{i-1}$ and $u_i$, the vertex $v$ can see only one obstacle edge $e$ if looking between $u_{i-1}$ and $u_i$. Thus, there is a unique funnel with apex $v$ and base $e$ whose lower chain (resp., upper chain) begins with $\overline{vu_i}$ (resp., $\overline{vu_{i-1}}$). In other words, given $\overline{vu_i}$ (or $\overline{vu_{i-1}}$), a funnel can be uniquely determined. Due to this property, the total number of funnels for $\mathcal{P}$ is at most $2k$ [10].

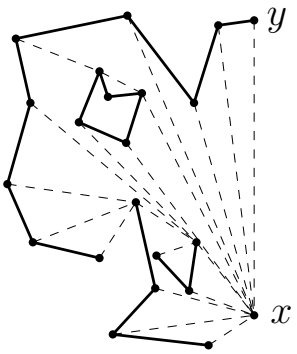


Figure 2: Illustrating the lower tree (with dashed segments) of $\overline{xy}$.

Denote by $F(\overline{xy})$ the set of all funnels whose base is $\overline{xy}$ (directed from $x$ to $y$). Since the free space lies to the left of $\overline{xy}$, all funnels are on the left of $\overline{xy}$. Suppose $v$ is the apex of a funnel $f_v \in F(\overline{xy})$ and $\overline{vu}$ is the first (directed) segment of the lower chain (i.e., from $v$ to $x$) of $f_v$. Then by convexity, $u$ is also the apex of a funnel $f_u$ and $f_u$ is contained in the funnel $f_v$. If we view the apex $u$ as the parent of the apex $v$, then the funnels in $F(\overline{xy})$ form a tree rooted at $x$ (see Fig. 2). We distinguish obstacle vertices from funnel apices because the same obstacle vertex can appear multiple times as apices in $F(\overline{xy})$, whereas each apex can appear only once (e.g., in Fig. 1(b), there are two funnel apices located at the physical obstacle vertex $v$). Each path from the root to a leaf in this tree is a convex chain that turns counterclockwise. We call this tree the *lower tree* for the obstacle edge $\overline{xy}$. Similarly, there

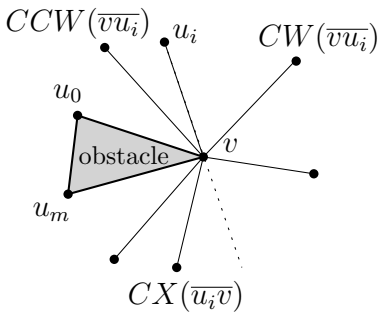


Figure 3: Illustrating the primitive operations. $CCX(\overline{u_iv})$ is undefined in this example.

is an *upper tree* for $\overline{xy}$. There is a natural linear ordering on the funnels of $F(\overline{xy})$ based on the clockwise preorder traversal of the lower tree, which, as shown in [10], is the same as the clockwise postorder traversal of the upper tree. The sequence of funnels in $F(\overline{xy})$ ordered as such is called the *funnel sequence* of $\overline{xy}$. Note that we treat the vertices $x$ and $y$ as two degenerate funnels. Below, we also use $F(\overline{xy})$ to denote the funnel sequence of $\overline{xy}$.

## 2.2 The Enhanced Visibility Graph

We first review some *primitive operations* that are used in the GM algorithm [10].

Consider an obstacle vertex $v$. Suppose $Vis(v) = \{u_0, u_1, \ldots, u_m\}$ is the clockwise sequence of all obstacle vertices of $\mathcal{P}$ visible to $v$ such that $\overline{vu_0}$ and $\overline{vu_m}$ are the two obstacle edges incident to $v$ (see Fig. 3). For any $u_i$, let its *clockwise successor*, denoted by $CW(\overline{vu_i})$, be $\overline{vu_{i+1}}$, and let its *counterclockwise successor*, denoted by $CCW(\overline{vu_i})$), be $\overline{vu_{i-1}}$. Both $CCW(\overline{vu_0})$ and $CW(\overline{vu_m})$ are undefined. We also define the *clockwise extension* $CX(\overline{u_iv})$ as follows. Rotate the ray originating at $v$ and going through $u_i$ clockwise by 180 degrees about $v$. If this rotation encounters neither $\overline{vu_0}$ nor $\overline{vu_m}$, then $CX(\overline{u_iv})$ is the very next visible vertex in $Vis(v)$ encountered after the 180 degree rotation (see Fig. 3); otherwise, $CX(\overline{u_iv})$ is undefined. The *counterclockwise extension* $CCX(\overline{u_iv})$ is defined symmetrically. Finally, define $REV(\overline{vu_i})$ to be the directed reversal of $\overline{vu_i}$, i.e., $REV(\overline{vu_i}) = \overline{u_iv}$ (by viewing $\overline{vu_i}$ and $\overline{u_iv}$ as directed segments).

The above five operations are called the *primitive operations*.

Define the *enhanced visibility graph* of $\mathcal{P}$ to be the visibility graph of $\mathcal{P}$ with the following additional properties: (1) For each graph edge, the five primitive operations $CW, CCW, CX, CCX$, and $REV$ can be performed in constant time each; (2) the funnel sequence $F(\overline{xy})$ of each obstacle edge $\overline{xy}$ is represented as a doubly linked list so that the operations of split, concatenation, predecessor, and successor can be performed in constant time each. Hence, the enhanced visibility graph essentially refers to the same graph as the visibility graph but is associated with more information. Henceforth, we use $\mathcal{G}$ to also denote the enhanced visibility graph of $\mathcal{P}$.

To implement the primitive operations, note that $CW, CCW$, and $REV$ are easy. For the other two, we call them *extension operations*, which are implemented in the following

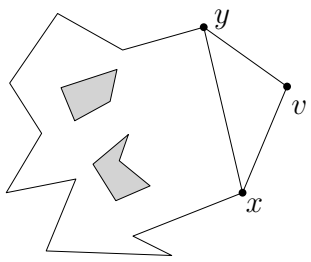


Figure 4: Illustrating an example of the SPLIT procedure performed on $\overline{xy}$ and an obstacle vertex $v$.

way. Let $\mathit{Vis}_l(v)$ be the set of vertices in $\mathit{Vis}(v)$ that are to the left of the vertical line through $v$, and $\mathit{Vis}_r(v) = \mathit{Vis}(v) \setminus \mathit{Vis}_l(v)$. The GM algorithm has two properties that help implement the two extension operations *CX* and *CCX* efficiently: (1) All vertices in $\mathit{Vis}_l(v)$ are identified before any in $\mathit{Vis}_r(v)$; (2) the vertices in $\mathit{Vis}_l(v)$ are cyclically ordered about $v$ before any vertex in $\mathit{Vis}_r(v)$ is identified. With these two properties, the GM algorithm uses the Split-Find (a reversal of the Union-Find) data structure [9] to implement the two extension operations on the incident graph edges of $v$ in $\mathcal{G}$ in constant time each. We call the vertical line through $v$ the *reference line* of $v$. In fact, the reference lines need not be vertical, and to implement the extension operations in constant time each, the same approach as in the GM algorithm still works as long as the above two properties hold. This is summarized in the following lemma.

**Lemma 1.** (Ghosh and Mount [10]) *For any obstacle vertex $v$, suppose there exists a line $l$ through $v$ such that: (1) all obstacle vertices visible to $v$ in one (open) half-plane $H(l)$ of $l$ are identified before any vertices visible to $v$ in the other (closed) half-plane of $l$; (2) the obstacle vertices visible to $v$ in $H(l)$ are ordered cyclically about $v$ before any vertices visible to $v$ in the other (closed) half-plane of $l$ are identified. Then the line $l$ can be used as a reference line of $v$ to implement the two extension operations for the edges of $\mathcal{G}$ incident to $v$ in constant time each.*

Later in our new algorithm, for those obstacle vertices in the bays and canals, we will use some reference lines that are not necessarily vertical.

The following lemma has been proved in [10].

**Lemma 2.** (Ghosh and Mount [10]) *If all primitive operations can be performed in constant time each, then the clockwise and counterclockwise traversals of the lower and upper trees can be carried out in time proportional to the sizes of the trees, and a funnel sequence can be traversed in time linear in its size.*

A fundamental procedure of the GM algorithm is called SPLIT. This procedure takes as input the enhanced visibility graph for a subregion $\mathcal{P}'$ of $\mathcal{P}$, a directed edge $\overline{xy}$ on the external boundary of $\mathcal{P}'$, and a point $v$ lying to the right of $\overline{xy}$ so that the triangle $\triangle xvy$ is external to $\mathcal{P}'$ (see Fig. 4). The SPLIT procedure merges $\triangle xvy$ into $\mathcal{P}'$ by computing the enhanced visibility graph for the enlarged region $\mathcal{P}' \cup \triangle xvy$. This is accomplished by identifying all obstacle vertices in $\mathcal{P}'$ visible to $v$ through $\overline{xy}$, and splitting the funnel sequence $F(\overline{xy})$ into two funnel sequences $F(\overline{xv})$ for $\overline{xv}$ and $F(\overline{vy})$ for $\overline{vy}$. Further, the

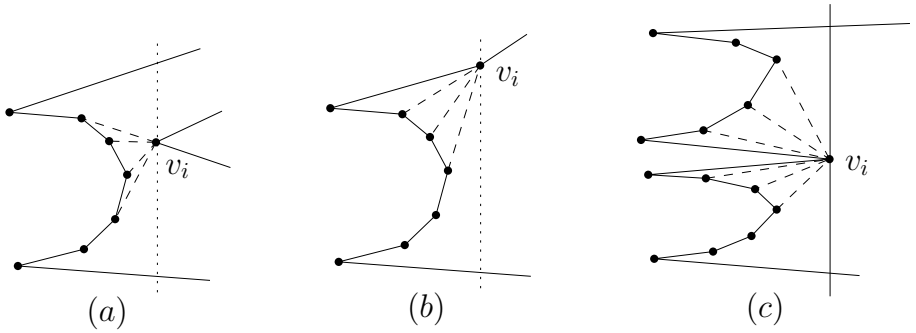


Figure 5: Incorporating an obstacle vertex $v_i$ into the triangulation.

vertices in $\mathcal{P}'$ visible to $v$ are identified in cyclical order about $v$ (each of these visible vertices is connected to $v$ by a visible segment). We say a point $p$ is visible to a point $q$ *through* a line segment $\overline{ab}$ if $p$ is visible to $q$ and $\overline{pq}$ intersects $\overline{ab}$. The following lemma has been shown in [10].

**Lemma 3.** (Ghosh and Mount [10]) *The SPLIT procedure on $\overline{xy}$ and $v$ can be performed in time linear in the number of obstacle vertices visible to $v$ through $\overline{xy}$.*

### 2.3 The Plane-Sweeping Triangulation and the Overall GM Algorithm

The GM algorithm is guided by a particular plane-sweeping triangulation algorithm [13]. The obstacle vertices are first sorted by increasing $x$-coordinates and are included into the triangulation in this order. Let $v_1, v_2, \ldots, v_n$ be the sorted list of the obstacle vertices. In a general step, the first $i-1$ obstacle vertices have been processed, and we have available a triangulation of a subregion $\mathcal{P}_{i-1}$ of $\mathcal{P}$ as well as the enhanced visibility graph of $\mathcal{P}_{i-1}$.

To process the vertex $v_i$, we add visible segments between $v_i$ and all vertices on the boundary of the triangulated region $\mathcal{P}_{i-1}$ that are visible to $v_i$ (see Fig. 5). As shown in [10, 13], these new triangles (with $v_i$ as a common vertex) form either one or two connected sequences around $v_i$ such that the sides opposite to $v_i$ in each such sequence form an inward-convex chain with respect to $v_i$; when there are two such sequences, they are separated by the boundary of $\mathcal{P}$ (see Fig. 5). In this way, we obtain the triangulated subregion $\mathcal{P}_i$.

We then compute the enhanced visibility graph for $\mathcal{P}_i$ by updating that for $\mathcal{P}_{i-1}$, as follows. For each connected sequence of vertices visible to $v_i$, if the sequence contains only one segment $\overline{xy}$, then we call the SPLIT procedure on $\overline{xy}$ and $v_i$. Otherwise, let $u_0, u_1, \ldots, u_m$ be the sequence (see Fig. 6). Our objective is to compute the funnel sequences $F(\overline{u_0v_i})$ for $\overline{u_0v_i}$ and $F(\overline{v_iu_m})$ for $\overline{v_iu_m}$. To this end, for each $1 \le j \le m$, we call the SPLIT procedure on $\overline{u_{j-1}u_j}$ and $v_i$, which splits the funnel sequence $F(\overline{u_{j-1}u_j})$ into two funnel sequences, $L_j$ for $\overline{u_{j-1}v_i}$, and $U_j$ for $\overline{v_iu_j}$. Note that the funnel sequence $F(\overline{u_{j-1}u_j})$ for each $1 \le j \le m$ has already been computed. The following lemma has been proved in [10].

**Lemma 4.** (Ghosh and Mount [10]) *The funnel sequence $F(\overline{u_0v_i})$ (resp., $F(\overline{v_iu_m})$) in $\mathcal{P}_i$ is the concatenation of the $L_j$'s (resp., $U_j$'s) for $j = 1, 2, \ldots, m$, followed (resp., preceded) by the trivial funnel whose apex is $v_i$.*

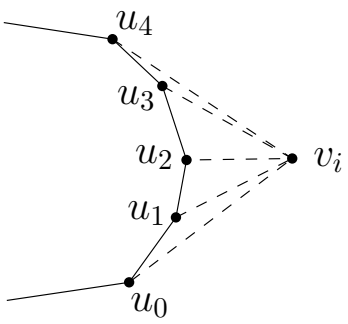


Figure 6: Connecting $v_i$ to a visible inward-convex chain.

By Lemma 3, the time for computing the enhanced visibility graph for $\mathcal{P}_i$ from that for $\mathcal{P}_{i-1}$ is proportional to the number of vertices in $\mathcal{P}_{i-1}$ that are visible to $v_i$.

The total time of the GM algorithm is $O(n \log n + k)$, where the $O(n \log n)$ time is due to the plane-sweeping triangulation and the $O(k)$ time is for all calls to the SPLIT procedure.

## 3 The Extended Corridor Structure and Observations

In this section, we discuss the extended corridor structure, which partitions the free space $\mathcal{F}$ into the ocean $\mathcal{M}$, bays, and canals. We also give some observations, which are used to show the correctness of our algorithm presented later.

The corridor structure was originally used to solve shortest path problems (e.g., [14, 16]), and later new concepts like "ocean", "bays", and "canals" have been introduced [4, 5, 7, 6, 8], which is referred to as the "extended corridor structure". This structure is a subdivision of the free space on which algorithms for specific problems rely. While the extended corridor structure itself is relatively simple, the main difficulty is to discover new observations and techniques that can be used to solve the problem. We briefly review the extended corridor structure below.

### 3.1 The Extended Corridor Structure

For simplicity of discussion, we assume all obstacles are contained in a large rectangle $\mathcal{R}$ (see Fig. 7). In fact, our algorithm works for any arbitrary simple polygon $\mathcal{R}$. Let $\mathcal{F}$ be the free space inside $\mathcal{R}$.

Denote by $Tri(\mathcal{F})$ an arbitrary triangulation of $\mathcal{F}$. The line segments of $Tri(\mathcal{F})$ that are not obstacle edges are referred to as *diagonals*. Let $G(\mathcal{F})$ denote the (planar) dual graph of $Tri(\mathcal{F})$, i.e., each node of $G(\mathcal{F})$ corresponds to a triangle in $Tri(\mathcal{F})$ and each edge connects two nodes of $G(\mathcal{F})$ corresponding to two triangles sharing a diagonal of $Tri(\mathcal{F})$. The degree of each node in $G(\mathcal{F})$ is at most three. Based on $G(\mathcal{F})$, 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 $G(\mathcal{F})$ along with its

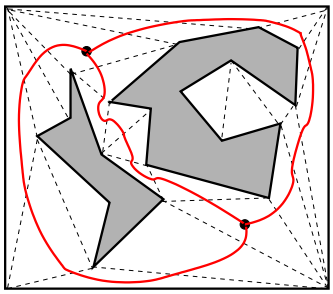

Figure 7: Illustrating a triangulation of the free space among two obstacles and the corridors (with red solid curves). There are two junction triangles indicated by the large dots inside them, connected by three solid (red) curves. Removing the two junction triangles results in three corridors.

incident edge; repeat this process until no degree-one node remains. 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 $G^3$ (e.g., see Fig. 7). By Euler's formula, the graph $G^3$ has $h+1$ faces, $2h-2$ nodes, and $3h-3$ edges [16]. Each node of $G^3$ corresponds to a triangle in $\mathit{Tri}(\mathcal{F})$, which is called a *junction triangle* (e.g., see Fig. 7). The removal of all junction triangles from $G^3$ results in $O(h)$ *corridors*, each of which corresponds to one edge of $G^3$.

The boundary of a corridor $C$ consists of four parts (see Fig. 8): (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 $c$ on an obstacle $P_j \in \mathcal{P}$ ($P_i = P_j$ is possible); (3) a boundary portion of the obstacle $P_j$ from $c$ to a point $d$; (4) a diagonal of a junction triangle from $d$ back to $a$. The two diagonals $\overline{bc}$ and $\overline{ad}$ are called the *doors* of $C$, and the other two parts of the boundary of $C$ are two *sides* of $C$. The corridor $C$ is a simple polygon whose interior does not intersect any obstacle. Let $|C|$ denote the number of obstacle vertices on the boundary of $C$. In $O(|C|)$ time, we can compute the shortest path $\pi(a,b)$ (resp., $\pi(c,d)$) from $a$ to $b$ (resp., $c$ to $d$) inside $C$. The region $H_C$ bounded by $\pi(a,b), \pi(c,d)$, and the two diagonals $\overline{bc}$ and $\overline{da}$ is called an *hourglass*, which is *open* if $\pi(a,b) \cap \pi(c,d) = \emptyset$ and *closed* otherwise (see Fig. 8). If $H_C$ is open, then both $\pi(a,b)$ and $\pi(c,d)$ are convex chains and are called the *sides* of $H_C$; otherwise, $H_C$ consists of two "funnels" and a path $\pi_C = \pi(a,b) \cap \pi(c,d)$ joining the two apices of the two funnels, called the *corridor path* of $C$. To distinguish from the funnels in the GM algorithm, we call the two funnels of $H_C$ *corridor-funnels*, or *c-funnels* for short. Each side of a c-funnel is also convex. We process each corridor as above. After $\mathit{Tri}(\mathcal{F})$ is computed, the time for processing all corridors is $O(n)$.

Define the space $\mathcal{M}$ to be the union of all $O(h)$ junction triangles, open hourglasses, and c-funnels. The space $\mathcal{M}$ is called the *ocean*. The boundary of $\mathcal{M}$ (denoted by $\partial\mathcal{M}$) consists of the sides of all open hourglasses and the sides of all c-funnels, which are all convex. In other words, $\partial\mathcal{M}$ consists of $O(h)$ convex chains with a total of $O(n)$ obstacle vertices. Therefore, the space $\mathcal{M}$ can be triangulated in $O(n + h\log h)$ time by the plane-sweeping algorithm in [13] because we can sort all obstacle vertices on $\partial\mathcal{M}$ in $O(n + h\log h)$ time. Further, this triangulation is sufficient for applying the GM algorithm to $\mathcal{M}$ to compute the

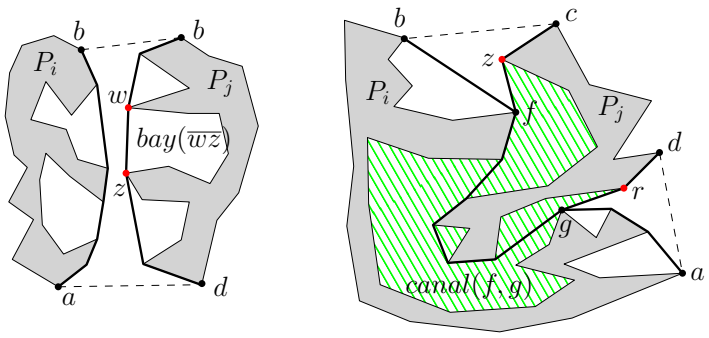


Figure 8: Illustrating an open hourglass (left) and a closed hourglass (right) with a corridor path connecting the apices $f$ and $g$ of the two c-funnels. The dashed segments are diagonals. The paths $\pi(a, b)$ and $\pi(c, d)$ are shown with thick solid curves. A bay $bay(\overline{wz})$ in the open hourglass (left) and the canal $canal(f, g)$ in the closed hourglass (right) are also indicated (with green backslash segments).

visibility graph for the obstacle vertices on $\partial\mathcal{M}$ with respect to the space $\mathcal{M}$. Denote by $\mathcal{G}_{\mathcal{M}}$ this visibility graph on $\mathcal{M}$. Since $\mathcal{M} \subseteq \mathcal{F}$, the graph $\mathcal{G}_{\mathcal{M}}$ is a subgraph of the visibility graph $\mathcal{G}$ for $\mathcal{P}$.

To construct $\mathcal{G}$, we need to know the portions of $\mathcal{G}$ which are not in $\mathcal{G}_{\mathcal{M}}$, i.e., $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$. For this, we first examine the space $\mathcal{F} \setminus \mathcal{M}$. Generally, the space $\mathcal{F} \setminus \mathcal{M}$ consists of two types of regions, called *bays* and *canals*. The details are given below.

Consider the hourglass $H_C$ of a corridor $C$. We first discuss the case when $H_C$ is open (see Fig. 8). $H_C$ 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 $w$ and $z$ be any two adjacent vertices on $S_1(H_C)$ such that the line segment $\overline{wz}$ is not an edge of $P$ (see the left figure in Fig. 8, with $P = P_j$). The free region enclosed by $\overline{wz}$ and a boundary portion of $P$ between $w$ and $z$ is called the *bay* of $\overline{wz}$ and $P$, denoted by $bay(\overline{wz})$, which is a simple polygon. We call $\overline{wz}$ the *bay gate* of $bay(\overline{wz})$.

If the hourglass $H_C$ is closed, then let $f$ and $g$ be the two apices of its two c-funnels. Consider two adjacent vertices $w$ and $z$ on a side of a c-funnel such that the line segment $\overline{wz}$ is not an obstacle edge. If neither $w$ nor $z$ is a c-funnel apex, then $w$ and $z$ must both lie on the same side of the corridor $C$ and thus $\overline{wz}$ defines a bay. However, if one of $w$ and $z$ is a c-funnel apex, say, $f = w$, then $f$ and $z$ may lie on different sides of $C$. If they both lie on the same side of $C$, then they also define a bay; otherwise, we call $\overline{fz}$ the *canal gate* at $f$ (see Fig. 8). Similarly, there is also a canal gate at the c-funnel apex $g$, say $\overline{gr}$. The region of the corridor $C$ bounded by the two canal gates $\overline{fz}$ and $\overline{gr}$ that contains the corridor path of $H_C$ is called the *canal* of $H_C$, denoted by $canal(f, g)$, which is also a simple polygon. The following observation on the canal $canal(f, g)$ has already been discussed in [5].

**Observation 1.** [5] *Given a point $p$ on the gate $\overline{fz}$ and a point $q$ on the gate $\overline{gr}$, the shortest path from $p$ to $q$ in $canal(f, g)$ is the concatenation of $\overline{pf}$, the corridor path of $canal(f, g)$, and $\overline{gq}$. The corridor path of $canal(f, g)$ is the shortest path from $f$ to $g$ in $canal(f, g)$.*

It is easy to see that $\mathcal{F} \setminus \mathcal{M}$ consists of all bays and canals thus defined.

### 3.2 Observations

The fact that each bay has only one gate allows us to handle a bay relatively 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 potentially give us trouble. The next lemma (proved in [5]) shows an important property of the canals, which essentially says that an observer outside a canal cannot see any point outside the canal through the canal (and its two gates).

**Lemma 5.** [5] *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.*

As shown later, Lemma 5 allows us to deal with each canal (almost) as simply as a bay. For example, our algorithm can process each gate of a canal independently.

Next, we consider the (missing) graph portion $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$. For a bay or a canal, if a vertex $v$ of it does not lie on a gate of it, then we call $v$ a *non-gate vertex*; otherwise, $v$ is a *gate vertex*. Note that the vertices of $\mathcal{G}$ that are not in $\mathcal{G}_{\mathcal{M}}$ are non-gate vertices of all bays and canals. For each graph edge $e = \overline{uv}$ in $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$, since $e \in \mathcal{G}$, $u$ is visible to $v$ with respect to $\mathcal{F}$ (i.e., $\overline{uv} \subset \mathcal{F}$). Since $e \notin \mathcal{G}_{\mathcal{M}}$, there are two possible cases: (1) at least one of $u$ and $v$ is not in $\mathcal{G}_{\mathcal{M}}$; (2) both $u$ and $v$ are in $\mathcal{G}_{\mathcal{M}}$ but $\overline{uv} \not\subset \mathcal{M}$ (i.e., $u$ and $v$ are not mutually visible with respect to $\mathcal{M}$).

For case (1), suppose $u$ is a non-gate vertex of a bay/canal and $v$ is not a vertex of the same bay/canal ($v$ may be either a non-gate vertex of another bay/canal or a vertex of $\mathcal{G}_{\mathcal{M}}$); then since $e = \overline{uv} \subset \mathcal{F}$, $\overline{uv}$ must intersect a gate of that bay/canal. If both $u$ and $v$ are vertices of the same bay and $\overline{uv}$ is not a bay gate (i.e., at least one of $u$ and $v$ is a non-gate vertex), then $e$ must be contained in the bay. But, if both $u$ and $v$ are vertices of the same canal and at least one of $u$ and $v$ is a non-gate vertex, then since a canal has two gates, either $e$ is contained in the canal or $e$ intersects both its gates and has a portion outside the canal (i.e., the portion of $e$ between its two intersections with the two canal gates is outside the canal).

For case (2), the next lemma shows that it is actually a nonissue.

**Lemma 6.** *For any two vertices $u$ and $v$ of $\mathcal{G}_{\mathcal{M}}$, if $\overline{uv} \subset \mathcal{F}$ and $\overline{uv} \not\subset \mathcal{M}$, then $u$ and $v$ are the two c-funnel apices of the same canal whose corridor path is the line segment $\overline{uv}$ (see Fig. 9).*

*Proof.* Since $\overline{uv} \not\subset \mathcal{M}$, $\overline{uv}$ must intersect the space $\mathcal{F} \setminus \mathcal{M}$. In other words, $\overline{uv}$ must intersect the inside of a bay or a canal (we say that a point $p$ in a bay/canal is in the *inside* of the bay/canal if and only if $p$ is not on any gate of the bay/canal). Since $\overline{uv} \subset \mathcal{F}$ and both $u$ and $v$ are vertices of $\mathcal{G}_{\mathcal{M}}$, $\overline{uv}$ cannot intersect the inside of any bay because a bay has only one gate. Below, we discuss the case when $\overline{uv}$ intersects the inside of a canal $canal(f, g)$.

Because both $u$ and $v$ are vertices of $\mathcal{G}_{\mathcal{M}}$, neither of them is a non-gate vertex of $canal(f, g)$. Since $\overline{uv}$ intersects the inside of $canal(f, g)$, when moving from $u$ to $v$ along $\overline{uv}$, we must move into the inside of $canal(f, g)$ through one of its gates and move out of

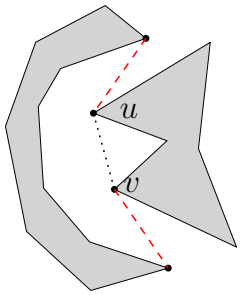


Figure 9: Illustrating a canal: the two gates are shown with (red) dashed segments and the corridor path is $\overline{uv}$.

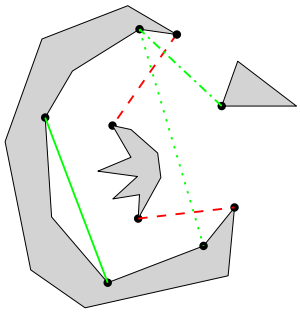

Figure 10: Illustrating the three cases of Lemma 7 with a canal (the two red dashed segments are its gates): an edge of Case (I) is shown with (green) solid segment; an edge of Case (II) is shown with (green) dotted segment; an edge of Case (III) is shown with (green) dashed dotted segment.

its inside through the other gate. Let $\overline{fz}$ and $\overline{gr}$ be these two gates. The above discussion implies that there is a point $u' \in \overline{fz} \cap \overline{uv}$ and a point $v' \in \overline{gr} \cap \overline{uv}$ such that $u' \neq v'$ and the open line segment of $\overline{u'v'}$ is in the inside of $canal(f, g)$. By Observation 1, the shortest path $\pi(u', v')$ from $u'$ to $v'$ in $canal(f, g)$, which is $\overline{u'v'}$, is the concatenation of $\overline{u'f}$, the corridor path of $canal(f, g)$, and $\overline{gv'}$. Thus, the corridor path is $\overline{fg} \subseteq \overline{u'v'}$. Since the open line segment of $\overline{u'v'}$ is in the inside of $canal(f, g)$, $\overline{fg}$ intersects the inside of $canal(f, g)$, and hence $f \neq g$. If $u' \neq f$, then $\overline{u'f}$ is a line segment contained in $\overline{u'v'}$, implying that $\overline{fz}$ is collinear with $g$, a contradiction with the general position assumption that no three obstacle vertices are collinear. Thus, $u' = f$. Similarly, we can show $v' = g$. Hence, both $u'$ and $v'$ are vertices of $\mathcal{G}_{\mathcal{M}}$, and $u = u'$ and $v = v'$ (due to the general position assumption). Therefore, it must be that $\overline{uv} = \overline{fg}$, which is also the corridor path connecting the two c-funnel apices $u = f$ and $v = g$ in $canal(f, g)$. The lemma thus follows. □

If the above case (2) occurs, then by Lemma 6, $u$ and $v$ are the two c-funnel apices of the same canal and its corridor path is $\overline{uv}$ (i.e., $\overline{uv}$ is contained in the canal). Hence, $\overline{uv}$ is an edge of the visibility graph of the canal.

The following lemma summarizes our discussions above.

**Lemma 7.** *Each graph edge $e = \overline{uv} \in \mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$ belongs to one of the following three cases (see Fig. 10).*

**(I)** *Both $u$ and $v$ are vertices of the same bay/canal but $\overline{uv}$ is not a gate, and $\overline{uv}$ is contained in that bay/canal (i.e., $u$ and $v$ are mutually visible within that bay/canal).*

**(II)** *Both $u$ and $v$ are vertices of the same canal and at least one of $u$ and $v$ is a non-gate vertex, and $\overline{uv}$ intersects both gates of that canal and has a portion outside the canal.*

**(III)** *$u$ is a non-gate vertex of a bay/canal, $v$ is not a vertex of the same bay/canal, and $\overline{uv}$ intersects a gate of that bay/canal.*

Henceforth, we call the edges of $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$ the *missing edges*. We need to compute all three cases of missing edges. More specifically, the missing edges of Case (I) are computed

in the preprocessing in Section 4 and those of the other two cases are computed in the main algorithm in Section 5. Actually, our algorithm does not first compute the graph $\mathcal{G}_{\mathcal{M}}$ and then add $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$ to it; instead, it constructs both $\mathcal{G}_{\mathcal{M}}$ and $\mathcal{G} \setminus \mathcal{G}_{\mathcal{M}}$ simultaneously.

## 4 The Preprocessing of the Bays and Canals

In this section, we perform a preprocessing step on all bays and canals. More specifically, for each bay/canal, we compute the enhanced visibility graph (and the funnel structure) for it. Further, the enhanced visibility graph is not built in an arbitrary manner but has some special properties that are needed by the main algorithm.

Our preprocessing algorithm is based on a modification of Hershberger's algorithm [12]. In Section 4.1, we compute an enhanced visibility graph for an arbitrary simple polygon. In Section 4.2, we compute a specific enhanced visibility graph for each bay/canal by modifying the algorithm in Section 4.1, which has some properties useful to our main algorithm.

### 4.1 Computing the Enhanced Visibility Graph of a Simple Polygon

Let $P$ be a simple polygon of $n'$ vertices, and $k'$ be the number of edges in the visibility graph of $P$. Our goal is to compute an enhanced visibility graph (and the funnel structure) of $P$ in $O(n'+k')$ time, by modifying Hershberger's algorithm [12], which computes the visibility graph of $P$ in $O(n'+k')$ time. We first briefly review the algorithm in [12].

Let $v_1, v_2, \ldots, v_{n'}$ be the vertices of $P$ along its boundary counterclockwise. The algorithm first computes the shortest path map in $P$ with respect to the vertex $v_1$, denoted by $SPM(v_1)$, in $O(n')$ time [11]. Using $SPM(v_1)$, we can easily find all vertices of $P$ visible to $v_1$ in $O(1)$ time each. Then, the algorithm computes the shortest path map $SPM(v_2)$ by updating $SPM(v_1)$. From $SPM(v_2)$, we find all vertices of $P$ visible to $v_2$ in constant time each. This process is repeated until $SPM(v_{n'})$ is computed. The visibility graph of $P$ is thus constructed.

To analyze the running time, it was shown in [12] that computing $SPM(v_2)$ from $SPM(v_1)$ can be done in time proportional to the size of differences between $SPM(v_1)$ and $SPM(v_2)$. More specifically, denote by $diff(v_1, v_2)$ the set of interior edges each of which is in one of $SPM(v_1)$ and $SPM(v_2)$ but not in both. In other words, $diff(v_1, v_2)$ is the symmetric difference between the two sets of interior edges of $SPM(v_1)$ and $SPM(v_2)$. Denote by $P_{vis}(\overline{v_1v_2})$ the *edge-visibility polygon* of the edge $\overline{v_1v_2}$, i.e., the subpolygon of $P$ each of whose points is visible to some point on $\overline{v_1v_2}$. It was proved in [12] that the edges in $diff(v_1, v_2)$ are all in $P_{vis}(\overline{v_1v_2})$. Given $SPM(v_1)$, $SPM(v_2)$ can be computed in $O(|diff(v_1, v_2)|)$ time [12]. It was also proved [12] that the total number of differences between the shortest path maps of all adjacent vertices of $P$ is $O(k')$. Hence, the time of the entire algorithm is $O(k'+n')$.

Next, we describe our algorithm for computing an enhanced visibility graph of $P$, denoted by $EG_P$. Let $G_P$ be the (ordinary) visibility graph of $P$. Recall that $EG_P$ is $G_P$ which has the following two additional properties: (1) For each edge of $EG_P$, the primitive

operations $CW, CCW, CX, CCX$, and $REV$ can be performed in constant time each; (2) the funnel sequence $F(\overline{xy})$ of each edge $\overline{xy}$ of $P$ is represented as a doubly linked list.

To compute $EG_P$, we first apply the algorithm in [12] to obtain $G_P$. Then, we use exactly the same approach as the GM algorithm to implement the primitive operations in constant time each. (In fact, since the graph $G_P$ is available, this can be done much more easily.) Specifically, for each vertex $v$, we use a doubly linked list to represent all its incident edges in $G_P$ in a cyclical order to implement the operations $CW$ and $CCW$. The operation $REV$ is trivial. For the two extension operations $CX$ and $CCX$, recall that for each vertex $v$, the GM algorithm uses a vertical line through $v$ as a reference line to help implement them. Here, since $G_P$ is already available, we know all edges incident to $v$, and thus can use any line passing through $v$ as a reference line to implement $CX$ and $CCX$.

Consider the problem of constructing the funnel sequence $F(\overline{xy})$ for each edge $\overline{xy}$ of $P$. Since we can implement all primitive operations in constant time each, by Lemma 2, for any edge $\overline{xy}$ of $P$, we can perform a clockwise preorder traversal of the lower tree of $\overline{xy}$ in time proportional to the size of its funnel sequence. Recall that the obstacle vertices visited by the clockwise preorder traversal of the lower tree of $\overline{xy}$ are exactly the ordered apices of the funnel sequence of $\overline{xy}$. Thus, during the traversal, we can easily build a doubly linked list for $F(\overline{xy})$. Therefore, we construct $F(\overline{xy})$ in time proportional to the size of its funnel sequence. We compute this for each edge of $P$, after which all funnel sequences are obtained.

**Lemma 8.** *The total time for constructing the funnel sequences for all edges of $P$ is $O(k')$.*

*Proof.* For any edge $\overline{xy}$ of $P$, we have shown that constructing its funnel sequence can be done in linear time in the size of the funnel sequence. According to the result in [10], the total size of the funnel sequences of all edges of $P$ is at most $2k'$. Hence, the lemma follows. □

We thus have the following result.

**Theorem 1.** *For any simple polygon $P$ of $n'$ vertices, we can build the enhanced visibility graph of $P$ in $O(n' + k')$ time, where $k'$ is the number of edges in the graph.*

### 4.2 Computing the Enhanced Visibility Graphs of Bays and Canals

We discuss a specific implementation of the enhanced visibility graphs for bays and canals. The reason for this is that in the main algorithm we will need to combine the funnel sequences of a bay/canal with those outside the bay/canal through its gates and this specific implementation of the enhanced visibility graph for the bay/canal will be helpful for the main algorithm. More precisely, for each bay/canal, we choose specific reference lines for implementing the two extension operations $CX$ and $CCX$, based on some observations on the structure of the bay/canal. The details are given below. Recall that $\mathcal{G}$ is the visibility graph of $\mathcal{P}$.

We first discuss the case of a bay $bay(\overline{wz})$. Later in the main algorithm we will identify the vertices outside $bay(\overline{wz})$ that are visible to the non-gate vertices of the bay.

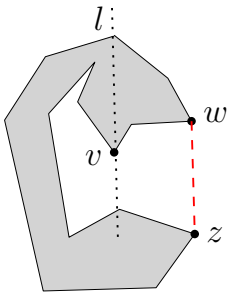


Figure 11: Illustrating the reference line $l$ of a non-gate vertex $v$ in a bay.

Each visible segment between such a vertex pair must intersect the gate $\overline{wz}$ of $bay(\overline{wz})$. The next lemma determines a reference line for each non-gate vertex of $bay(\overline{wz})$ to be used in the main algorithm.

**Lemma 9.** *For each non-gate vertex $v$ of $bay(\overline{wz})$, the line through $v$ and parallel to the gate $\overline{wz}$ can be used as a reference line for implementing the two extension operations for the edges of $\mathcal{G}$ incident to $v$ in constant time each in the main algorithm (see Fig. 11).*

*Proof.* If the vertex $v$ is not visible to any point outside the bay, then $v$ cannot be visible to any obstacle vertex outside the bay, and thus we can use any line through $v$ to be its reference line as in the case of an arbitrary simple polygon in Section 4.1. In the following, we assume $v$ is visible to at least one point outside the bay. Denote the line specified in the lemma by $l$ (see Fig. 11). We need to show that $l$ has the two properties required in Lemma 1.

The line $l$ divides the plane into two half-planes. Let $L$ be the open half-plane of $l$ that does not contain $\overline{wz}$. Denote by $R$ the other (closed) half-plane of $l$. Let $V_L$ (resp., $V_R$) be the set of vertices of $bay(\overline{wz})$ visible to $v$ in $L$ (resp., $R$). Let $V'_R$ be the set of obstacle vertices visible to $v$ that will be identified in the main algorithm. In other words, $V'_R$ is the set of obstacle vertices visible to $v$ that are outside the bay. Consider any vertex $u \in V'_R$. As discussed earlier, $\overline{uv}$ must intersect $\overline{wz}$. Since $l$ is parallel to $\overline{wz}$ and $v$ is a non-gate vertex of the bay, $u$ must be in $R$. Therefore, all vertices of $V'_R$ are in $R$.

Since the visibility graph of $bay(\overline{wz})$ is computed in the preprocessing, we can process the vertices of $V_R$ later than those in $V_L$ when implementing the extension operations (i.e., we may view $V_R$ as being identified later than $V_L$ in the main algorithm). Since the vertices in $V'_R$ will be found in the main algorithm, they are identified later than those in $V_L$ (which are found in the preprocessing). Since all vertices in $V_R$ and $V'_R$ are in $R$, property (1) of Lemma 1 is satisfied. Since the visibility graph of $bay(\overline{wz})$ is available, the vertices of $V_L$ are cyclically ordered before the main algorithm; thus property (2) of Lemma 1 is satisfied too. Hence, the lemma follows. □

We then consider the case of a canal. Our task is to determine a reference line for each canal non-gate vertex to implement the two extension operations in constant time each in the main algorithm. To this end, as in the bay case, the key is to determine, for each

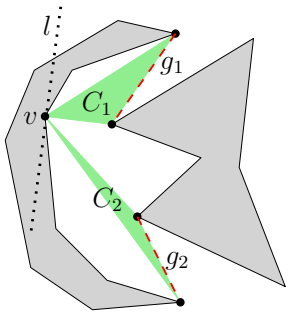


Figure 12: Illustrating a canal with two gates $g_1$ and $g_2$. The two cones $C_1$ and $C_2$ of a non-gate vertex $v$ are shown.

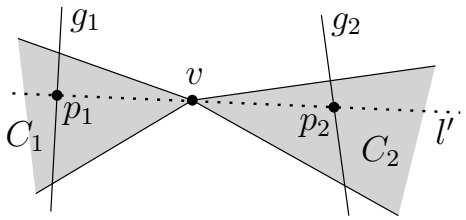


Figure 13: The line $l'$ is in the union of the cones $C_1$ and $C_2$; $g_1$ and $g_2$ are the two canal gates.

canal non-gate vertex $v$, a reference line through $v$ such that all obstacle vertices visible to $v$ to be found in the main algorithm lie in the same half-plane of the line. In the case of a bay, this is easy since it has only one gate. In Lemma 10 below, we show that such a reference line for each vertex in a canal still exists. Later in Lemma 11, we give an algorithm based on the proof of Lemma 10 that computes the reference lines for all vertices of a canal in linear time.

**Lemma 10.** *For each non-gate vertex $v$ in a canal, there exists a reference line through $v$ for implementing the two extension operations in constant time each in the main algorithm.*

*Proof.* Consider a non-gate vertex $v$ in a canal $\mathcal{C}$. Our goal is to prove that there exists a reference line $l$ through $v$ such that all obstacle vertices visible to $v$ to be identified in the main algorithm lie on the same (closed) side of $l$. Note that for any vertex $u$ visible to $v$ identified in the main algorithm, the line segment $\overline{uv}$ must intersect a gate of the canal, say, at a point $p$, such that $p$ is visible to $v$ with respect to $\mathcal{C}$ (i.e., $\overline{pv}$ is contained in $\mathcal{C}$). (Note that $\overline{uv}$ may be a Case (II) or Case (III) type missing edge in Lemma 7.) Thus, it suffices to show the following claim: There exists a reference line $l$ through $v$ such that all points on the two canal gates of $\mathcal{C}$ visible to $v$ with respect to $\mathcal{C}$ lie on the same (closed) side of $l$. In the proof of this claim below, "visible" always means "visible with respect to $\mathcal{C}$".

If $v$ is not visible to any point on the two canal gates, then the claim trivially holds.

If $v$ is visible to only one gate of $\mathcal{C}$, then as in the case of a bay, the claim follows.

If $v$ is visible to both canal gates, then for each gate, we can determine a 2-D cone that has $v$ as the apex and is bounded by two half-lines such that the portion of the gate visible to $v$ is the intersection of the cone and the gate [11] (to be exact, the two half-lines are those containing the first segments on the shortest paths in $\mathcal{C}$ from $v$ to the two endpoints of the gate, and such a half-line may degenerate into a single point if $v$ is an endpoint of the gate). See Fig. 12. Let $g_1$ and $g_2$ be the two canal gates. Denote by $C_1$ and $C_2$ the two cones intersecting the two gates $g_1$ and $g_2$, respectively. To prove the claim, it suffices to show that there exists a line $l$ through $v$ such that these two cones both lie on the same (closed) side of $l$, as follows.

Assume to the contrary that there exists no such line. Then we can find a line $l'$

contained in $C_1 \cup C_2$ such that $l'$ does not contain any bounding half-line of a cone, say, $C_1$ (see Fig. 13). Let $g_1'$ be the intersection of $g_1$ and $C_1$. Then, $l'$ intersects $g_1'$ at an interior point of $g_1'$, say $p_1$; thus $p_1$ is not an endpoint of the gate $g_1$. Note that $v$ is also not an endpoint of $g_1$ (otherwise, the cone $C_1$ consists of the single half-line containing the gate $g_1$, and the line $l'$ must contain the single bounding half-line of $C_1$, which contradicts with the assumption that $l'$ does not contain any bounding half-line of $C_1$). Let $p_2$ be the intersection of $l'$ and $g_2$ (see Fig. 13). Thus, $\overline{vp_1}$ (resp., $\overline{vp_2}$) is contained in the cone $C_1$ (resp., $C_2$). Since $v \in \overline{p_1p_2} \subset l'$, the line segment $\overline{p_1p_2}$ is contained in the canal $\mathcal{C}$, and thus the shortest path from $p_1$ to $p_2$ in $\mathcal{C}$ is the line segment $\overline{p_1p_2}$. Let $g_1 = \overline{gr}$ such that $g$ is the apex of a c-funnel of $\mathcal{C}$. By Observation 1, the shortest path from $p_1$ to $p_2$ in the canal, which is $\overline{p_1p_2}$, contains $\overline{p_1g}$. Since $p_1$ is an interior point of $\overline{gr}$, we obtain that $\overline{p_1p_2}$ is collinear with $\overline{gr}$. Since $v$ is not an endpoint of $g_1 = \overline{gr}$ and $v \in \overline{p_1p_2}$, the three obstacle vertices $g$, $r$, and $v$ are collinear, which contradicts with our general position assumption.

Therefore, there exists a line $l$ through $v$ such that the two cones $C_1$ and $C_2$ are both on the same side of $l$. Consequently, for the case that $v$ is visible to both canal gates of $\mathcal{C}$, a reference line through $v$ also exists. The lemma thus follows. □

**Lemma 11.** *For any canal of $n'$ vertices, the reference lines for all non-gate vertices of the canal can be computed in $O(n')$ time.*

*Proof.* Let $g_1$ and $g_2$ be the two gates of a canal $\mathcal{C}$. We compute the edge-visibility polygon $\mathcal{C}(g_1)$ for $g_1$ in $O(n')$ time [11]. For each vertex $v$ in $\mathcal{C}(g_1)$, we can determine two line segments such that all points on $g_1$ visible to $v$ are exactly those seen by an observer located at $v$ looking between these two line segments (i.e., the two line segments are contained in the half-lines bounding the cone $C_1$ defined in the proof of Lemma 10). In other words, one (resp., the other) line segment is the first line segment in the shortest path in $\mathcal{C}$ from $v$ to one (resp., the other) endpoint of $g_1$. Note that these two line segments are readily available after we compute $\mathcal{C}(g_1)$ [11]. Similarly, we compute the edge-visibility polygon $\mathcal{C}(g_2)$ for the gate $g_2$, and for each vertex in $\mathcal{C}(g_2)$, determine two such line segments as in the case for $\mathcal{C}(g_1)$.

Then, for each vertex $v$ in $\mathcal{C}$, there are at most four line segments thus determined. By Lemma 10, there exists a line such that these four line segments all lie on the same (closed) side of the line. We then find an arbitrary such line in constant time as a reference line for $v$.

Therefore, we can compute the reference lines for all vertices of $\mathcal{C}$ in $O(n')$ time. □

Since the total number of vertices in all bays and canals is $O(n)$, the total time for processing all bays and canals is $O(n + k'')$, where $k''$ is the total number of edges in the visibility graphs within all bays and canals (note that $k'' \leq k$).

**Lemma 12.** *The running time of the preprocessing on all bays and canals is $O(n + k)$.*

In the main algorithm given next, we assume the preprocessing has been done and thus the specific enhanced visibility graphs for all bays and canals have been computed. Note

that all Case (I) missing edges in Lemma 7 have also been computed in the preprocessing. The Cases (II) and (III) edges will be computed in the main algorithm.

## 5 The Main Algorithm

As the GM algorithm, our main algorithm is guided by Hertel and Mehlhorn's plane-sweeping triangulation algorithm [13]. The difference is that the triangulation in the GM algorithm is on the space $\mathcal{F}$ whereas ours is on the space $\mathcal{M}$. As discussed before, since the boundary of $\mathcal{M}$ consists of $O(h)$ convex chains with a total of $O(n)$ vertices, applying Hertel and Mehlhorn's plane-sweeping triangulation algorithm on $\mathcal{M}$ takes $O(n + h \log h)$ time [13] (instead of $O(n \log n)$ time on $\mathcal{F}$; e.g., sorting all vertices $\mathcal{M}$ can be done in $O(n + h \log h)$ time). As in the GM algorithm, the funnel sequences are built during the triangulation. Further, during the triangulation, we also take the space $\mathcal{F} \setminus \mathcal{M}$, i.e., all bays and canals, into consideration, by combining the funnel sequences in all bays and canals with those in $\mathcal{M}$ through the gates of the bays and canals (this is partially done by so-called *connection procedures*).

In the general step of the triangulation algorithm, we consider a vertex, say $v$. The vertex $v$ will be connected to either one or two connected vertex sequences about $v$ such that the sides opposite to $v$ form an inward-convex chain with respect to $v$ (e.g., $v = v_i$ in Fig. 5).

### 5.1 The Single Triangle Case

We first consider the case of adding only a single triangle $\triangle vxy$ to the triangulation involving $v$. That is, $v$ is to the right of the directed edge $\overline{xy}$ with $x$ lower than $y$. Denote by $\mathcal{M}(v)$ the triangulated subregion of $\mathcal{M}$ right after the triangle $\triangle vxy$ is incorporated into the triangulation. Denote by $BC(v)$ the union of all bays and canals each of which has at least one gate on the boundary of $\mathcal{M}(v)$. Note that in the GM algorithm, $\overline{xv}$ (resp., $\overline{vy}$) is either a triangulation diagonal or an obstacle edge, whereas in our algorithm, it may also be a bay/canal gate.

Depending on whether $\overline{xv}$ is a bay/canal gate, we define a funnel sequence $F(\overline{xv})$ as follows. If $\overline{xv}$ is a triangulation diagonal or an obstacle edge, then we let $F(\overline{xv})$ be the funnel sequence of $\overline{xv}$ with respect to the region $\mathcal{M}(v) \cup BC(v)$; in other words, $F(\overline{xv})$ consists of all funnels whose bases are $\overline{xv}$ and whose apices are either vertices of $\mathcal{M}(v)$ or non-gate vertices of the bays and canals in $BC(v)$. But if $\overline{xv}$ is a bay/canal gate, then we let $F(\overline{xv})$ be the funnel sequence of $\overline{xv}$ with respect to that bay/canal, which has been computed in the preprocessing.

Define the funnel sequence $F(\overline{vy})$ for $\overline{vy}$ similarly. Our algorithm maintains the invariant that after the triangle $\triangle vxy$ is processed, the two funnel sequences $F(\overline{xv})$ and $F(\overline{vy})$ are computed.

We assume the corresponding funnel sequence $F(\overline{xy})$ for $\overline{xy}$ has been computed. Initially, if $\overline{xy}$ is an obstacle edge or a triangulation diagonal, then $F(\overline{xy}) = \emptyset$; otherwise, $\overline{xy}$ is a bay/canal gate, and $F(\overline{xy})$ has been computed in the preprocessing. Our task is to

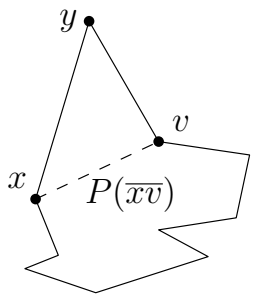


Figure 14: Illustrating an example that $\overline{xv}$ is a gate of a bay/canal $P(\overline{xv})$.

compute the two funnel sequences $F(\overline{xv})$ and $F(\overline{vy})$. Depending on whether $\overline{xv}$ and $\overline{vy}$ are bay/canal gates, there are three cases.

### 5.1.1 The first case

The first case is when neither $\overline{xv}$ nor $\overline{vy}$ is a bay/canal gate. The algorithm for this case is the same as the GM algorithm. That is, we call the SPLIT procedure on $\overline{xy}$ and $v$ to split $F(\overline{xy})$ and obtain two funnel sequences $F(\overline{xv})$ and $F(\overline{vy})$. By Lemma 3, the time of this computation is proportional to the number of vertices visible to $v$ through $\overline{xy}$.

### 5.1.2 The second case

The second case is when exactly one of $\overline{xv}$ and $\overline{vy}$ is a bay/canal gate. Without loss of generality, assume $\overline{xv}$ is a bay/canal gate but $\overline{vy}$ is not. Denote by $P(\overline{xv})$ the bay/canal bounded by $\overline{xv}$ (see Fig. 14).

We first call the SPLIT procedure on $\overline{xy}$ and $v$ to split the funnel sequence $F(\overline{xy})$ and obtain two funnel sequences: $F_{\overline{xy}}(\overline{xv})$ for $\overline{xv}$ and $F_{\overline{xy}}(\overline{vy})$ for $\overline{vy}$. Again by Lemma 3, the time for this is proportional to the number of vertices visible to $v$ through $\overline{xy}$. Since $\overline{xv}$ is a bay/canal gate, $F(\overline{xv})$ has been computed in the preprocessing. Further, there are two other key differences from the first case, as follows.

- First, $F(\overline{vy})$ may not be $F_{\overline{xy}}(\overline{vy})$ any more since some non-gate vertices in the bay/canal $P(\overline{xv})$ may be visible to $\overline{vy}$ through the gate $\overline{xv}$. Note that such non-gate vertices of $P(\overline{xv})$ may possibly define some missing edges of Cases (II) and (III) in Lemma 7. In other words, it is possible that there is a funnel whose apex is at a non-gate vertex of $P(\overline{xv})$ and whose base is $\overline{vy}$, and clearly this funnel should be in $F(\overline{vy})$.

- Second, it is possible that the apex of a funnel in $F_{\overline{xy}}(\overline{xv})$ is visible to a non-gate vertex of $P(\overline{xv})$ through $\overline{xv}$, and each such visible vertex pair defines a missing edge of Case (II) or Case (III) in Lemma 7. Since our triangulation of $\mathcal{M}$ will not be performed on $P(\overline{xv})$, we compute such missing edges right in this step. Below, we show how to handle these two differences.

To avoid confusion, we let $F_{P(\overline{xv})}(\overline{xv})$ denote $F(\overline{xv})$, i.e., the funnel sequence of $\overline{xv}$ with respect to $P(\overline{xv})$, which has been computed in the preprocessing.

To deal with the first difference, we call the SPLIT procedure on $\overline{xv}$ and $y$ to split $F_{P(\overline{xv})}(\overline{xv})$ and obtain two funnel sequences: $F_{P(\overline{xv})}(\overline{vy})$ for $\overline{vy}$ and $F_{P(\overline{xv})}(\overline{xy})$ for $\overline{xy}$. By Lemma 3, the time for this is proportional to the number of vertices of $P(\overline{xv})$ visible to $y$. Actually, we only need $F_{P(\overline{xv})}(\overline{vy})$. We have the following observation.

**Observation 2.** *The funnel sequence $F(\overline{vy})$ is the concatenation of $F_{P(\overline{xv})}(\overline{vy})$ and $F_{\overline{xy}}(\overline{vy})$.*

*Proof.* This is a special case of Lemma 4. □

Since both $F_{P(\overline{xv})}(\overline{vy})$ and $F_{\overline{xy}}(\overline{vy})$ have been computed, $F(\overline{vy})$ can thus be obtained.

To handle the second difference, we use a *connection procedure*, as follows.

First, we connect the apices of the funnels in $F_{\overline{xy}}(\overline{xv})$ by line segments following their funnel order in $F_{\overline{xy}}(\overline{xv})$ from $x$ to $v$. Since the funnel order in $F_{\overline{xy}}(\overline{xv})$ follows the clockwise preorder of the lower tree of $\overline{xv}$, the line segment connecting any two adjacent funnel apices is in the free space $\mathcal{F}$ (Lemma 5.1 in [10] gives similar results). Thus, the region bounded by these line segments and $\overline{xv}$ is a simple polygon, denoted by $P''$. Note that since an obstacle vertex may serve as the apices of multiple funnels, a same physical point location may serve as multiple vertices of the polygon $P''$, but $P''$ can still be viewed as a (degenerate) simple polygon. The number of vertices of $P''$ is exactly the number of funnels in $F_{\overline{xy}}(\overline{xv})$ (recall that we treat the two vertices $v$ and $x$ as two degenerate funnels); let $m'$ be this number. Then, we merge $P''$ with the bay/canal $P(\overline{xv})$ by removing $\overline{xv}$ to form a new simple polygon, denoted by $P'$. Let $n'$ be the number of vertices of $P(\overline{xv})$. Thus, $P'$ has $O(m'+n')$ vertices.

Let $V(\overline{xv})$ be the set of all vertices of the bay/canal $P(\overline{xv})$. Let $G(P')$ be the visibility graph of $P'$, and let $E$ be the set of the edges of $G(P')$ incident to the vertices of $V(\overline{xv})$. In the following, we will compute the set $E$ (note that we do not compute all edges of $G(P')$ but only the edges of $E$). To this end, since $P'$ is a (degenerate) simple polygon, we apply Hershberger's algorithm [12] on $P'$ but only on the vertices of $V(\overline{xv})$, as follows.

Suppose the obstacle vertices of $V(\overline{xv})$ ordered cyclically along the boundary of $P(\overline{xv})$ are $x = v_1, v_2, \ldots, v_{n'} = v$. We apply Hershberger's algorithm [12] on the vertices of $V(\overline{xv})$ in their index order. Specifically, we first compute the shortest path map $SPM(v_1)$ of $v_1$ in $P'$. Then, the edges of $G(P')$ incident to $v_1$ are known. Next, we compute $SPM(v_2)$ by updating $SPM(v_1)$ and using Hershberger's algorithm [12], after which the edges of $G(P')$ incident to $v_2$ are known. We continue this until $SPM(v_{n'})$ is computed, after which all edges of $E$ are computed.

The above algorithm is our *connection procedure*, which can be viewed as a process of "propagating" the funnels in $F_{\overline{xy}}(\overline{xv})$ into $P(\overline{xv})$ through the gate $\overline{xv}$. We have the following lemma.

**Lemma 13.** *The connection procedure runs in $O(n'+m'+k')$ time, where $k'$ is the sum of the sizes of the funnel sequences of the obstacle edges $\overline{v_{i-1}v_i}$ for $i = 2, 3, \ldots, n'$ with respect to $P'$. The total time of all connection procedures in the entire algorithm is $O(n+k)$.*

*Proof.* Recall that $P'$ is the union of $P''$ and $P(\overline{xv})$. As already discussed above, computing $P''$ can be done in $O(m')$ time by traversing the funnel sequence $F_{\overline{xy}}(\overline{xv})$. Hence, we can obtain $P'$ in $O(n'+m')$ time.

Consider our algorithm for computing the edges of $E$ by using Hershberger's algorithm [12]. The first step computes the shortest path map $SPM(v_1)$, which takes $O(n'+m')$ time. As discussed in Section 4.1, computing $SPM(v_2)$ is done by updating $SPM(v_1)$, which takes $O(|diff(v_1,v_2)|)$ time, where $diff(v_1,v_2)$ is the symmetric difference between the sets of edges in $SPM(v_1)$ and $SPM(v_2)$.

Let $F(\overline{v_1v_2})$ denote the funnel sequence of $\overline{v_1v_2}$ in $P'$. We claim that $|diff(v_1,v_2)| = O(|F(\overline{v_1v_2})|)$, where $|F(\overline{v_1v_2})|$ is the size of $F(\overline{v_1v_2})$. Indeed, since $P'$ is a simple polygon, for any vertex $u$ of $P'$, $F(\overline{v_1v_2})$ has at most one funnel whose apex is $u$. Thus, each vertex of $P'$ can serve as the apex of at most one funnel in $F(\overline{v_1v_2})$. Hence, the size of $F(\overline{v_1v_2})$ is exactly the number of vertices of $P'$ that are visible to $\overline{v_1v_2}$. Let $P'_{vis}(\overline{v_1v_2})$ denote the edge-visibility polygon of $\overline{v_1v_2}$ in $P'$. The above discussions show that $|P'_{vis}(\overline{v_1v_2})|$, i.e., the size of $P'_{vis}(\overline{v_1v_2})$, is bounded by $O(|F(\overline{v_1v_2})|)$. On the other hand, the results in [12] imply that the $|diff(v_1,v_2)| = O(|P'_{vis}(\overline{v_1v_2})|)$ because the edges of $diff(v_1,v_2)$ are all in $P'_{vis}(\overline{v_1v_2})$ [12]. Hence, we obtain that $|diff(v_1,v_2)| = O(|F(\overline{v_1v_2})|)$.

In general, for any $2 \leq i \leq n'$, computing $SPM(v_i)$ by updating $SPM(v_{i-1})$ runs in $O(|diff(v_{i-1},v_i)|)$ time, which is $O(|F(\overline{v_{i-1}v_i})|)$ time. Recall that $k' = \sum_{i=2}^{n'} |F(\overline{v_{i-1}v_i})|$. According to the above discussion, after $SPM(v_1)$ is computed, the total time for computing $SPM(v_i)$ for $2 \leq i \leq n'$ is $O(k')$. Hence, the total time of the algorithm for computing $E$ is $O(n'+m'+k')$, and total time of the connection procedure is also $O(n'+m'+k')$.

To prove the lemma, in the sequel we show that the sums of all such $n'$, $m'$, and $k'$ in all connection procedures in the entire algorithm are $O(n)$, $O(k)$, and $O(k)$, respectively.

For the case of $n'$, this is easy. Indeed, if $P(\overline{xv})$ is a bay, then $P(\overline{xv})$ is only involved in at most one connection procedure since it has only one gate; if $P(\overline{xv})$ is a canal, then $P(\overline{xv})$ is involved in at most two connection procedures since it has two gates. In either case, the sum of all such $n'$ in the entire algorithm is $O(n)$.

To prove the other two cases for $m'$ and $k'$, we first argue that the funnels in $F_{\overline{xy}}(\overline{xv})$ will not be involved in any other connection procedure later in the algorithm. If $P(\overline{xv})$ is a bay, this is obviously true since $P(\overline{xv})$ has only one gate (intuitively, the funnels in $F_{\overline{xy}}(\overline{xv})$ cannot be propagated out of $P(\overline{xv})$ and they all "terminate" inside $P(\overline{xv})$).

If $P(\overline{xv})$ is a canal, one may wonder that some funnels in $F_{\overline{xy}}(\overline{xv})$ could be involved in other connection procedures since $P(\overline{xv})$ has another gate. This actually cannot happen due to Lemma 5. Indeed, consider the apex $u$ of a funnel $f_u$ in $F_{\overline{xy}}(\overline{xv})$. If $u$ is outside the canal (and thus $u$ may form some Case III missing edges), then in light of Lemma 5, $u$ cannot be visible to a vertex outside the canal through the canal (and its two gates). In other words, after the funnel $f_u$ is propagated into the canal through the gate $\overline{xv}$, it cannot be propagated out of the canal through the other gate. If $u$ is a gate vertex of the gate $\overline{xv}$, then the funnel $f_u$ is the only vertex $u$, which is a degenerate funnel and can be ignored. The remaining case is that $u$ is a vertex of $P(\overline{xv})$ and $u$ is neither $x$ nor $v$, which implies that the funnel $f_u$ had been propagated out of $P(\overline{xv})$ through the other gate, say $g_1$. Thus,

the other gate $g_1$ had already been incorporated into the triangulated region $\mathcal{M}(v)$ before the vertex $v$ is considered; further, if $u'$ is a non-gate vertex of $P(\overline{xv})$ visible to $u$ through the gate $\overline{xv}$ identified in the above connection procedure, then $\overline{u'u}$ also intersects $g_1$ and $\overline{u'u}$ is a Case II missing edge of Lemma 7. (Thus the Case II missing edges are computed in the connection procedures.) After the funnel $f_u$ is propagated into $P(\overline{xv})$ through $\overline{xv}$, since $f_u$ intersects both gates of $P(\overline{xv})$, it cannot be propagated out of $P(\overline{xv})$ through the other gate $g_1$ again. Hence, in any case, no funnels of $F_{\overline{xy}}(\overline{xv})$ will be propagated outside of $P(\overline{xv})$ through the other gate. In other words, no funnels in $F_{\overline{xy}}(\overline{xv})$ will be involved in another connection procedure later in the algorithm.

The above shows the observation that the funnels in $F_{\overline{xy}}(\overline{xv})$ will not be involved in any other connection procedure later in the algorithm.

Recall that $m'$ is the number of funnels in $F_{\overline{xy}}(\overline{xv})$. We proceed to show that the sum of all such $m'$ in all connection procedures in the entire algorithm is $O(k)$. To this end, due to the above observation, it is sufficient to show that the total number of such funnels of $F_{\overline{xy}}(\overline{xv})$ in all connection procedures of the entire algorithm is $O(k)$. Indeed, when the funnels of $F_{\overline{xy}}(\overline{xv})$ are propagated into $P(\overline{xv})$, some funnels may be split and eventually become funnels of the obstacle edges of $P(\overline{xv})$. This implies that that the number of funnels of $F_{\overline{xy}}(\overline{xv})$ is no more than the number of funnels of the obstacle edges of $P(\overline{xv})$. Since the total number of funnels of all obstacle edges of all bays/canals of $\mathcal{P}$ is $O(k)$, our claim follows.

Recall that $k' = \sum_{i=2}^{n'} |F(\overline{v_{i-1}v_i})|$. To prove the lemma, it remains to show that the sum of all such $k'$ in all connection procedures of the entire algorithm is $O(k)$.

Recall that a funnel can be uniquely determined by the first visible segment (i.e., the segment incident to the funnel apex) of its lower chain. Let $S$ denote the set of the first visible segments of the lower chains of all funnels in $F(\overline{v_{i-1}v_i})$ for all $i = 2, 3, \ldots, n'$. Hence, $|S| = k'$. We partition the segments of $S$ into three types, as follows.

Consider any segment $\overline{ua}$ of $S$ such that $u$ is a funnel apex. According to the definition of funnels, if we extend $\overline{ua}$ along the direction from $u$ to $a$, then the extension will first hit an obstacle edge $\overline{v_{i-1}v_i}$ of $P(\overline{xv})$, and the funnel corresponding to $\overline{ua}$ belongs to $F(\overline{v_{i-1}v_i})$.

- If $\overline{ua}$ intersects the gate $\overline{vx}$ and $\{u, a\} \cap \{v, x\} = \emptyset$, we call $\overline{ua}$ a $k'_1$-*type* segment. Let $k'_1$ denote the total number of all such segments in $S$.
- If $\overline{ua}$ is contained in $P(\overline{xv})$, we call $\overline{ua}$ a $k'_2$-*type* segment. Let $k'_2$ denote the total number of all such segments in $S$.
- The rest of segments of $S$ are called $k'_3$-*type* segments. Let $k'_3$ denote the total number of all such segments in $S$. Note that for each $k'_3$-type segment $\overline{ua}$, if we extend $\overline{ua}$ along the direction from $u$ to $a$, then the extension will first hit the gate $\overline{xv}$ before it hits an obstacle edge of $P(\overline{xv})$; this implies that $\overline{ua}$ corresponds to a funnel of $F_{\overline{xy}}(\overline{xv})$.

Clearly, $k' = k'_1 + k'_2 + k'_3$. In the following, we show that for each $i = 1, 2, 3$, the sum of all such $k'_i$ in the entire algorithm is $O(k)$.

- If $\overline{ua}$ is a $k'_1$-type segment, then it is computed only in the current connection procedure because no other connection procedures will be called on the gate $\overline{xv}$. Hence, $\overline{ua}$ is computed only once in all connection procedures of the entire algorithm. Thus, the sum of all such $k'_1$ in the entire algorithm is $O(k)$.
- If $\overline{ua}$ is a $k'_2$-type segment, then it can be computed at most twice in all connection procedures of the entire algorithm: once in the current connection procedure and once in the connection procedure for the other gate if $P(\overline{xv})$ is a canal. Hence, the sum of all such $k'_2$ in the entire algorithm is $O(k)$.
- If $\overline{ua}$ is a $k'_3$-type segment, then $\overline{ua}$ corresponds to a funnel of $F_{\overline{xy}}(\overline{xv})$. Since the funnels of $F_{\overline{xy}}(\overline{xv})$ will never be involved in any connection procedure later in the algorithm, the sum of all such $k'_3$ in the entire algorithm is $O(k)$.

Therefore, the sum of all $k'$ in all connection procedures in the entire algorithm is $O(k)$.

This proves the lemma. □

We give some comments on a few technical details for implementing the primitive operations. We have computed the funnel sequence $F(\overline{vy})$ as a concatenation of $F_{P(\overline{xv})}(\overline{vy})$ and $F_{\overline{xy}}(\overline{vy})$. Later in the algorithm, we may identify a vertex visible to a non-gate vertex in $P(\overline{xv})$ through $\overline{xv}$ and $\overline{vy}$. For each non-gate vertex of $P(\overline{xv})$, to implement the primitive operations in constant time each, as in the GM algorithm, the operations $CW, CCW$, and $REV$ are quite straightforward. For the two extension operations $CX$ and $CCX$, recall that our preprocessing for computing the enhanced visibility graphs for all bays and canals has found a particular reference line for each non-gate vertex of $P(\overline{xv})$ that can make sure that the two extension operations are implemented in constant time each (see Lemmas 9 and 10).

In addition, for each of the two gate vertices $x$ and $v$, again the operations $CW, CCW$, and $REV$ are trivial. For the two extension operations $CX$ and $CCX$, consider the gate vertex $v$. As in the GM algorithm, we can still use the vertical lines through $v$ as the reference line. However, since the connection procedure may have found some vertices in $P(\overline{xv})$ visible to $v$ that are to the right of $v$, in order to fulfill the two conditions of the reference line in Lemma 1, we can process those vertices (i.e., insert them to the Split-Find data structure) after we process the rest of the vertices visible to $v$ (i.e., those visible vertices that are to the left of $v$). This is in fact consistent with the GM algorithm. The other gate vertex $x$ can be handled in the same way.

This finishes the discussion for the second case where $\overline{xv}$ is a bay/canal gate but $\overline{vy}$ is not.

#### 5.1.3 The third case

The third case is when both $\overline{xv}$ and $\overline{vy}$ are bay/canal gates (see Fig. 15). The process for this case is similar to the second case except that there is one more connection procedure (because there is one more bay/canal gate involved). The details are given below.

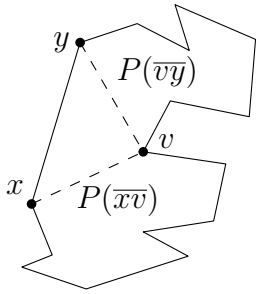


Figure 15: Illustrating an example that both $\overline{xv}$ and $\overline{vy}$ are bay/canal gates.

Denote by $P(\overline{xv})$ (resp., $P(\overline{vy})$) the bay/canal bounded by $\overline{xv}$ (resp., $\overline{vy}$). In this case, the two funnel sequences $F(\overline{xv})$ and $F(\overline{vy})$ have been computed in the preprocessing, and again we re-denote them by $F_{P(\overline{xv})}(\overline{xv})$ and $F_{P(\overline{vy})}(\overline{vy})$, respectively. The remaining task is to compute the Cases (II) and (III) missing edges, as in the second case.

We first do the same things as in the second case, i.e., compute the funnel sequences $F_{\overline{xy}}(\overline{xv})$ and $F_{\overline{xy}}(\overline{vy})$, compute the funnel sequence $F'(\overline{vy})$ (which is $F(\overline{vy})$ in the second case) that is the concatenation of $F_{P(\overline{xv})}(\overline{vy})$ and $F_{\overline{xy}}(\overline{vy})$, and apply the connection procedure to $F_{\overline{xy}}(\overline{xv})$ and $P(\overline{xv})$ through the gate $\overline{xv}$.

Since now $\overline{vy}$ is also a bay/canal gate, the new funnel sequence $F'(\overline{vy})$ needs to be propagated into $P(\overline{vy})$ through $\overline{vy}$. In other words, we need to identify all visible vertex pairs between the apices of the funnels in $F'(\overline{vy})$ and the vertices in $P(\overline{vy})$. To do so, we use another connection procedure on $F'(\overline{vy})$ and $P(\overline{vy})$ similarly to that on $F_{\overline{xy}}(\overline{xv})$ and $P(\overline{xv})$. Specifically, we connect the apices of the funnels in $F'(\overline{vy})$ to form a (degenerate) simple polygon and merge it with $P(\overline{vy})$ to obtain another simple polygon, denoted by $P'$. Then, we compute the edges of the visibility graph of $P'$ incident to the vertices of $P(\overline{vy})$ in the same way as before.

### 5.2 The General Case

In the general case, $v$ may be connected to a convex chain of vertices, as shown in Fig. 6. As in the GM algorithm [10], this case can be handled by an easy generalization of the algorithm for the single triangle case. Comparing with the GM algorithm, as in the single triangle case, our algorithm makes one or two connection procedure calls. The details are given below.

Let the involved sequence of vertices be $u_0, u_1, \ldots, u_m$ (see Fig. 6, with $v = v_i$). Note that the funnel sequences $F(\overline{u_{i-1}u_i})$ for $1 \leq i \leq m$ have all been computed. The segments $\overline{vu_i}$, $i = 1, 2, \ldots, m-1$, are all triangulation diagonals. But $\overline{vu_0}$ and $\overline{vu_m}$ can be triangulation diagonals, obstacle edges, or bay/canal gates. As the single triangle case, depending on whether $\overline{vu_0}$ and $\overline{vu_m}$ are bay/canal gates, there are three cases.

In the first case, if neither $\overline{vu_0}$ nor $\overline{vu_m}$ is a bay/canal gate, then the algorithm is the same as the GM algorithm (as discussed in Lemma 4).

The second case is when exactly one of $\overline{vu_0}$ and $\overline{vu_m}$ is a bay/canal gate. Without

loss of generality, we assume $\overline{vu_0}$ is a bay/canal gate but $\overline{vu_m}$ is not. Denote by $P(\overline{vu_0})$ the bay/canal bounded by $\overline{vu_0}$. We first apply the same approach as in the GM algorithm to obtain two funnel sequences, $F'(\overline{vu_0})$ for $\overline{vu_0}$ and $F'(\overline{vu_m})$ for $\overline{vu_m}$, by splitting the funnel sequences $F(\overline{u_{i-1}u_i})$ for all $1 \leq i \leq m$ and connecting $v$ to all visible vertices thus found. Since $\overline{vu_0}$ is a bay/canal gate, we have a funnel sequence $F_{P(\overline{vu_0})}(\overline{vu_0})$ with respect to $P(\overline{vu_0})$, which has been computed in the preprocessing. We use the SPLIT procedure to split $F_{P(\overline{vu_0})}(\overline{vu_0})$ into two funnel sequences: $F_{P(\overline{vu_0})}(\overline{vu_m})$ for $\overline{vu_m}$ and another one for the convex chain from $u_0$ to $u_m$ (the latter one is actually not needed), in the following way.

Note that since $u_0$ is connected to $u_m$ by a convex chain instead of a line segment, we cannot directly apply the SPLIT procedure on $\overline{vu_0}$ and $u_m$ to compute the funnel sequence $F_{P(\overline{vu_0})}(\overline{vu_m})$. Instead, we use the following approach to compute $F_{P(\overline{vu_0})}(\overline{vu_m})$ without affecting the overall $O(n+k)$ time complexity, and the approach is a modification of the SPLIT procedure. We apply the SPLIT procedure on $\overline{vu_0}$ and $u_m$ to split the funnel sequence $F_{P(\overline{vu_0})}(\overline{vu_0})$ by assuming $u_0$ is connected to $u_m$ by a line segment and by scanning the funnel sequence $F_{P(\overline{vu_0})}(\overline{vu_0})$ from $v$. However, the modification is that during the SPLIT procedure whenever we find a visible segment added to $u_m$ (i.e., a vertex of $P(\overline{vu_0})$ visible to $u_m$ is identified) that is actually "blocked" by the line segment $\overline{u_m u_{m-1}}$, we stop the procedure and $F_{P(\overline{vu_0})}(\overline{vu_m})$ is thus obtained. Recall that the visible segments added to $u_m$ in the SPLIT procedure are found in the cyclical order about $u_m$ from $v$ [10]. Thus, the time for computing $F_{P(\overline{vu_0})}(\overline{vu_m})$ above is linear in the number of visible segments added to $u_m$, i.e., the number of vertices of $P(\overline{vu_0})$ visible to $u_m$ through the gate $\overline{vu_0}$.

As in the single triangle case, the funnel sequence $F(\overline{vu_m})$ for $\overline{vu_m}$ is the concatenation of $F_{P(\overline{u_0v})}(\overline{vu_m})$ and $F'(\overline{vu_m})$.

Next, as in the single triangle case, the apices of the funnels in $F'(\overline{u_0v})$ may be visible to some non-gate vertices of $P(\overline{u_0v})$. To identify these visible vertex pairs, we also apply a connection procedure in the same way as the single triangle case. Although in this case we have a convex chain from $u_0$ to $u_m$ instead of a single line segment, if we connect the apices of all funnels in $F'(\overline{u_0v})$ by line segments, we still have a (degenerate) simple polygon (similar results have been shown in [10]), and we merge it with $P(\overline{u_0v})$ to form a new simple polygon, say $P'$. Again, we can use the same algorithm as before to compute the edges of the visibility graph of $P'$ incident to the vertices of $P(\overline{u_0v})$.

The third case is when both $\overline{vu_0}$ and $\overline{vu_m}$ are bay/canal gates. Denote by $P(\overline{vu_0})$ (resp., $P(\overline{vu_m})$) the bay/canal bounded by $\overline{vu_0}$ (resp., $\overline{vu_m}$). As in the second case, we compute $F'(\overline{u_0v})$ and $F''(\overline{vu_m})$ that is the concatenation of $F_{P(\overline{u_0v})}(\overline{vu_m})$ and $F'(\overline{vu_m})$. We also call the connection procedure on $F'(\overline{u_0v})$ and $P(\overline{vu_0})$ through the gate $\overline{vu_0}$. Then, as in the single triangle case, we make another connection procedure call on the funnel sequence $F''(\overline{vu_m})$ and $P(\overline{vu_m})$ through the gate $\overline{vu_m}$.

We have finished the discussions of all cases. All connection procedures in the entire algorithm take $O(n+k)$ time. The proof of the next theorem summarizes the time analysis of the overall algorithm.

**Theorem 2.** *The enhanced visibility graph for $\mathcal{P}$ can be computed in $O(n + h\log h + k)$ time, after an arbitrary triangulation of the free space is computed.*

*Proof.* Given any triangulation of the free space, we can compute the extended data structure, i.e., identifying the ocean $\mathcal{M}$ as well as all bays and canals, in $O(n)$ time [16]. Then, we do the preprocessing for all bays and canals, which takes $O(n+k)$ time. We perform the plane-sweeping triangulation on $\mathcal{M}$ in $O(n+h\log h)$ time [13]. Following the triangulation of $\mathcal{M}$, we compute the enhanced visibility graph in the same way as the GM algorithm, applying the connection procedures if needed. The total time of all SPLIT procedures and connection procedures is $O(n+k)$. □

## 6 Concluding Remarks

We have presented a new algorithm for computing the vertex-vertex visibility graph of $\mathcal{P}$. Our algorithm follows a similar scheme to the GM algorithm but avoids the $O(n\log n)$ time triangulation used in the GM algorithm. This is done with the help of the extended corridor structure. More specifically, instead of performing triangulation in the entire free space, we only do so in the ocean, and then use connection procedures to connect the visibility graph in the ocean with that in all bays and canals. This reduces the $O(n\log n)$ time factor in the GM algorithm to $O(T+n+h\log h)$, and thus provides an alternative $O(T+n+h\log h+k)$ time algorithm for the problem to the one that was already given by Kapoor and Maheshwari [15].

The enhanced visibility graph and funnel structures of $\mathcal{P}$ are quite useful and have many other applications. For example, we can extend our algorithm to compute the vertex-edge, edge-edge, and extended vertex-vertex visibility graphs in an output-sensitive manner. Some of these have been discussed in [10] and we give some details below for the completeness of this paper.

Note that not all obstacle edges of $\mathcal{P}$ have its funnel sequences computed in the algorithm, e.g., those on the left of the free space. But after the enhanced visibility graph is computed, we can easily construct the funnel sequences for all obstacle edges in additional $O(n+k)$ time by using the primitive operations to traverse either the lower trees or upper trees, as discussed in Lemma 2.

**The vertex-edge visibility graph.** To compute the vertex-edge visibility graph, as discussed in [10], we can label each funnel apex by the unique obstacle edge that can be seen by the apex through its funnel. Specifically, for each obstacle edge $e$, by Lemma 2, we can traverse the lower tree of $e$ in time proportional to the size of the tree, which is the number of apices visible to $e$. Hence, we can label each funnel apex of the tree by $e$. Since the total number of funnels is $O(k)$, we can finish this labeling for all obstacle edges in totally $O(n+k)$ time and construct the vertex-edge graph. Thus, the vertex-edge visibility graph can be constructed in $O(n+h\log h+k)$ time, given any triangulation of the free space.

**The visibility polygons of obstacle vertices.** For any obstacle vertex $v$, let $u_0, u_1, \ldots, u_m$ be the clockwise sequence of obstacle vertices visible to $v$ such that $\overline{vu_0}$ and $\overline{vu_m}$ are the two obstacle edges incident to $v$. Then for any adjacent pair of $u_{i-1}$ and $u_i$, we have a funnel whose apex is $v$ and whose lower chain (resp., upper chain) begins with $\overline{vu_i}$ (resp., $\overline{vu_{i-1}}$). Since we have already associated every funnel with its base, we know the obstacle

edge that $v$ can see between $u_{i-1}$ and $u_i$. Thus, we can simply scan $u_0, u_1, \ldots, u_m$ clockwise to construct the visibility polygon of $v$ in $O(m)$ time, where $m$ is the size of the visibility polygon.

**The extended vertex-vertex visibility graph.** To build the extended vertex-vertex visibility graph (i.e., for every edge of the vertex-vertex visibility graph, extend it along both directions until hitting an obstacle or infinity and record the two extension endpoints), consider any obstacle edge $e$. For any edge of the lower tree of $e$, one of its two extension endpoints is on $e$. Thus, after we traverse the lower trees of all obstacle edges, we can identify the extension endpoints of all edges of the vertex-vertex visibility graph. Hence, the extended vertex-vertex visibility graph can be built in $O(n+h\log h+k)$ time, given any triangulation of the free space.

**The edge-edge visibility graph.** For the edge-edge visibility graph, as discussed in [10], two obstacle edges $e_1$ and $e_2$ are mutually visible if and only if there exist vertices $u$ and $v$ such that the funnel whose lower chain begins with $\overline{uv}$ sees the edge $e_1$ and the funnel whose lower chain begins with $\overline{vu}$ sees the edge $e_2$. Hence, in addition to the vertex-vertex and vertex-edge relations (on the corresponding obstacle edges), for each edge of the vertex-vertex visibility graph of $\mathcal{P}$, if its two extension endpoints are on two obstacle edges $e_1$ and $e_2$, then $e_1$ is visible to $e_2$. Therefore, the edge-edge visibility graph can also be built in $O(n + h\log h + k)$ time, given any triangulation of the free space.

## Acknowledgments

The authors would like to thank the anonymous reviewers for their comments that help improve the presentation of the paper. We are particularly grateful to one reviewer for pointing out several mistakes in our original manuscript. D.Z. Chen's research was supported in part by NSF under Grant CCF-1217906. H. Wang's research was supported in part by NSF under Grant CCF-1317143.

## References

[1] T. Asano, T. Asano, L. Guibas, J. Hershberger, and H. Imai. Visibility of disjoint polygons. *Algorithmica*, 1(1):49–63, 1986.

[2] R. Bar-Yehuda and B. Chazelle. Triangulating disjoint Jordan chains. *International Journal of Computational Geometry and Applications*, 4(4):475–481, 1994.

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

[4] D.Z. Chen and H. Wang. A nearly optimal algorithm for finding $L_1$ shortest paths among polygonal obstacles in the plane. In *Proc. of the 19th European Symposium on Algorithms (ESA)*, pages 481–492, 2011.

[5] D.Z. Chen and H. Wang. Computing the visibility polygon of an island in a polygonal domain. In *Proc. of the 39th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 218–229, 2012. Journal version published online in *Algorithmica*, 2015.

[6] D.Z. Chen and H. Wang. $L_1$ shortest path queries among polygonal obstacles in the plane. In *Proc. of 30th Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 293–304, 2013.

[7] D.Z. Chen and H. Wang. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms*, 11, 2015. Article No. 26.

[8] D.Z. Chen and H. Wang. Visibility and ray shooting queries in polygonal domains. *Computational Geometry: Theory and Applications*, 48:31–41, 2015.

[9] H. Gabow and R.E. Tarjan. A linear-time algorithm for a special case of disjoint set union. *Journal of Computer and System Sciences*, 30:209–221, 1985.

[10] S.K. Ghosh and D.M. Mount. An output-sensitive algorithm for computing visibility graphs. *SIAM Journal on Computing*, 20(5):888–910, 1991.

[11] L.J. Guibas, J. Hershberger, D. Leven, M. Sharir, and R.E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

[12] J. Hershberger. An optimal visibility graph algorithm for triangulated simple polygons. *Algorithmica*, 4:141–155, 1989.

[13] S. Hertel and K. Mehlhorn. Fast triangulation of the plane with respect to simple polygons. *Information and Control*, 64:52–76, 1985.

[14] S. Kapoor and S.N. Maheshwari. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proc. of 4th Annual ACM Symposium on Computational Geometry (SoCG)*, pages 172–182, 1988.

[15] S. Kapoor and S.N. Maheshwari. Efficiently constructing the visibility graph of a simple polygon with obstacles. *SIAM Journal on Computing*, 30(3):847–871, 2000.

[16] S. Kapoor, S.N. Maheshwari, and J.S.B. Mitchell. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete and Computational Geometry*, 18(4):377–383, 1997.

[17] M. Keil, D.M. Mount, and S. K. Wismath. Visibility stabs and depth-first spiralling on line segments in output sensitive time. *International Journal of Computational Geometry and Applications*, 10(5):535–552, 2000.

[18] D.T. Lee. Proximity and reachability in the plane. Ph.D thesis and Technical Report ACT-12, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, IL, 1978.

[19] J.S.B. Mitchell, G. Rote, and G. Woeginger. Minimum-link paths among obstacles in the plane. *Algorithmica*, 8:431–459, 1992.

[20] M.H. Overmars and E. Welzl. New methods for computing visibility graphs. In *Proc. of the 4th Annual Symposium on Computational Geometry (SoCG)*, pages 164–171, 1988.

[21] M. Sharir and A. Schorr. On shortest paths in polyhedral spaces. *SIAM Journal on Computing*, 15(1):193–215, 1986.

[22] S. Suri and J. O'Rourke. Worst-case optimal algorithms for constructing visibility polygons with holes. In *Proc. of the 2nd Annual Symposium on Computational Geometry (SoCG)*, pages 14–23, 1986.

[23] E. Welzl. Constructing the visibility graph for $n$ line segments in $O(n^2)$ time. *Information Processing Letters*, 20:167–171, 1985.