A New Algorithm for Euclidean Shortest Paths in the Plane

HAITAO WANG, University of Utah

Given a set of pairwise disjoint polygonal obstacles in the plane, finding an obstacle-avoiding Euclidean shortest path between two points is a classical problem in computational geometry and has been studied extensively. Previously, Hershberger and Suri (in *SIAM Journal on Computing*, 1999) gave an algorithm of $O(n \log n)$ time and $O(n \log n)$ space, where n is the total number of vertices of all obstacles. Recently, by modifying Hershberger and Suri's algorithm, Wang (in SODA'21) reduced the space to $O(n)$ while the runtime of the algorithm is still $O(n \log n)$. In this article, we present a new algorithm of $O(n + h \log h)$ time and $O(n)$ space, provided that a triangulation of the free space is given, where h is the number of obstacles. The algorithm is better than the previous work when h is relatively small. Our algorithm builds a shortest path map for a source point s so that given any query point t, the shortest path length from s to t can be computed in $O(\log n)$ time and a shortest s-t path can be produced in additional time linear in the number of edges of the path.

CCS Concepts: • **Theory of computation** → **Computational geometry**; **Design and analysis of algorithms;**

Additional Key Words and Phrases: Shortest path, shortest path map, shortest path query, Euclidean distance, obstacle avoidance, polygonal domain

ACM Reference format:
Haitao Wang. 2023. A New Algorithm for Euclidean Shortest Paths in the Plane. *J. ACM* 70, 2, Article 11 (March 2023), 62 pages.
https://doi.org/10.1145/3580475

11

1 INTRODUCTION

Let $\mathcal{P}$ be a set of h pairwise disjoint polygonal obstacles with a total of n vertices in the plane. Let $\mathcal{F}$ denote the *free space*—that is, the plane minus the interior of the obstacles. Given two points s and t in $\mathcal{F}$, we consider the problem of finding a Euclidean shortest path from s to t in $\mathcal{F}$. This is a classical problem in computational geometry and has been studied extensively (e.g., [3, 18–20, 22, 24, 25, 34, 36, 38, 42, 44, 45, 48]).

To solve the problem, there are two commonly used methods: the visibility graph and the continuous Dijkstra. The visibility graph method is to first construct the visibility graph of the vertices of $\mathcal{P}$ along with s and t, and then run Dijkstra's shortest path algorithm on the graph to find a shortest s-t path. The best algorithms for constructing the visibility graph run in $O(n \log n + K)$

This research was supported in part by the NSF under grants CCF-2005323 and CCF-2300356.
Author's address: H. Wang, School of Computing, University of Utah, 50 Central Campus Drive, Salt Lake City, UT 84117; email: haitao.wang@utah.edu.

0004-5411/2023/03-ART11 $15.00
https://doi.org/10.1145/3580475

time [18] or in $O(n+h\log^{1+\epsilon} h+K)$ time [7, 30] for any constant $\epsilon > 0$ after the free space $\mathcal{F}$ is triangulated, where K is the number of edges of the visibility graph. Because $K = \Omega(n^2)$ in the worst case, the visibility graph method inherently takes quadratic time. To deal with the case where h is relatively small compared to n, a variation of the visibility graph method was proposed that first constructs a so-called *tangent graph* (or *relevant visibility graph*) and then finds a shortest s-t path in the graph. Using this method, Kapoor et al. [31] gave an algorithm of $O(n+h^2\log n)$ time; later Chen and Wang [6] derived a better algorithm of $O(n+h\log h+K')$ time, after the free space $\mathcal{F}$ is triangulated, where K' may be considered as the number of tangents among obstacles of $\mathcal{P}$ and $K' = O(h^2)$. Note that triangulating $\mathcal{F}$ can be done in $O(n\log n)$ time or in $O(n+h\log^{1+\epsilon} h)$ time for any constant $\epsilon > 0$ [2]. Hence, the running time of the preceding algorithm in the work of Chen and Wang [6] is still quadratic in the worst case.

Using the continuous Dijkstra method, Mitchell [38] made a breakthrough and achieved the first subquadratic algorithm of $O(n^{3/2+\epsilon})$ time for any constant $\epsilon > 0$. Also using the continuous Dijkstra approach plus a novel conforming subdivision of the free space, Hershberger and Suri [25] presented an algorithm of $O(n\log n)$ time and $O(n\log n)$ space; the running time is optimal when $h = \Theta(n)$ as $\Omega(n+h\log h)$ is a lower bound in the algebraic computation tree model (which can be obtained by a reduction from sorting; e.g., see Theorem 3 in the work of de Rezende et al. [14] for a similar reduction). Recently, by modifying Hershberger and Suri's algorithm, Wang [48] reduced the space to $O(n)$ while the running time is still $O(n\log n)$. In addition, Inkulu et al. [28] announced an algorithm of $O(n+h\log h\log n)$ time and $O(n)$ space, which is also based on the continuous Dijkstra approach.

All three continuous Dijkstra algorithms [25, 38, 48] construct the *shortest path map*, denoted by $SPM(s)$, for a source point s. $SPM(s)$ is of $O(n)$ size and can be used to answer shortest path queries. By building a point location data structure on $SPM(s)$ in additional $O(n)$ time [16, 32], given a query point t, the shortest path length from s to t can be computed in $O(\log n)$ time and a shortest s-t path can be output in time linear in the number of edges of the path.

The problem setting thus defined for $\mathcal{P}$ (i.e., h pairwise disjoint polygonal obstacles with a total of n vertices) is usually referred to as *polygonal domains* or *polygons with holes* in the literature. The problem in simple polygons is relatively easier [19, 20, 22, 24, 34] because there is a unique shortest path between any two points in a simple polygon. Guibas et al. [20] presented an algorithm that can construct a shortest path map in linear time. For the two-point shortest path query problem where both s and t are query points, Guibas and Hershberger [19, 22] built a data structure in linear time such that each query can be answered in $O(\log n)$ time. In contrast, the two-point query problem in polygonal domains is much more challenging (one reason is that there may be multiple topologically different shortest paths between two points in a polygonal domain): to achieve $O(\log n)$ time queries, the current best result uses $O(n^{11})$ space [10]; alternatively Chiang and Mitchell [10] gave a data structure of $O(n+h^5)$ space with $O(h\log n)$ query time. Refer to their work [10] for other data structures with a trade-off between space and query time.

The L_1 counterpart of the problem where the path length is measured in the L_1 metric also attracted much attention (e.g., [1, 9, 12, 13, 35, 37]). For polygons with holes, Mitchell [35, 37] gave an algorithm that can build a shortest path map for a source point in $O(n\log n)$ time and $O(n)$ space; for small h, Chen and Wang [9] proposed an algorithm of $O(n+h\log h)$ time and $O(n)$ space, after the free space is triangulated. For simple polygons, Bae and Wang [1] built a data structure in linear time that can answer each two-point L_1 shortest path query in $O(\log n)$ time. The two-point query problem in polygons with holes has also been studied [4, 5, 47]. To achieve $O(\log n)$ time queries, the current best result uses $O(n+h^2\log^3 h/\log\log h)$ space [47].

As discussed previously, the polygonal domain $\mathcal{P}$ is described by two parameters n and h. Since in many applications (e.g., geographic problems) h could be much smaller than n (Figure 1), the

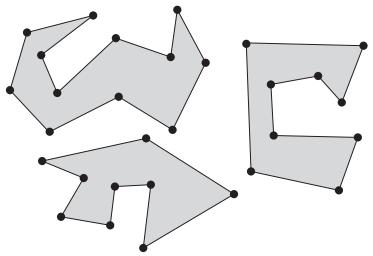

Fig. 1. Illustrating a polygonal domain of $h = 3$ obstacles and $n = 30$ vertices.

time complexities of algorithms for problems in $\mathcal{P}$ may be better measured by both n and h than by n itself. For example, comparing to an $O(n \log n)$ time algorithm, an algorithm of $O(n + h \log h)$ time would be preferred, not only because $n + h \log h$ is asymptotically smaller than $n \log n$ for sufficiently small h but also because an $O(n + h \log h)$ time algorithm suggests a very different view of the problem than an $O(n \log n)$ time algorithm. Indeed, an $O(n + h \log h)$ time algorithm conveys the idea that the boundary features of $\mathcal{P}$ only provide information defining the input and in some sense almost do not contribute to the complexity of the algorithm, while something akin to sorting needs to be done to the holes.

1.1 Our Result

In this article, we show that the problem of finding an Euclidean shortest path among obstacles in $\mathcal{P}$ is solvable in $O(n + h \log h)$ time and $O(n)$ space, after a triangulation of the free space $\mathcal{F}$ is given. If the time for triangulating $\mathcal{F}$ is included and the triangulation algorithm of Bar-Yehuda and Chazelle [2] is used, then the total time of the algorithm is $O(n + h \log^{1+\epsilon} h)$, for any constant $\epsilon > 0$.[1] With the assumption that the triangulation could be done in $O(n + h \log h)$ time, which has been an open problem and is beyond the scope of this work, our result settles Problem 21 in the Open Problem Project [15]. Our algorithm actually constructs the shortest path map $SPM(s)$ for the source point s in $O(n + h \log h)$ time and $O(n)$ space. We give an overview of our approach in the following.

The high-level scheme of our algorithm is similar to that for the L_1 case [47] in the sense that we first solve the *convex case* where all obstacles of $\mathcal{P}$ are convex and then extend the algorithm to the general case with the help of the extended corridor structure of $\mathcal{P}$ [4, 7–9, 29–31, 39].

The Convex Case. We first discuss the convex case. Let $\mathcal{V}$ denote the set of topmost, bottommost, leftmost, and rightmost vertices of all obstacles. Hence, $|\mathcal{V}| \leq 4h$. Using the algorithm of Hershberger and Suri [25], we build a conforming subdivision $\mathcal{S}$ on the points of $\mathcal{V}$, without considering the obstacle edges. Since $|\mathcal{V}| = O(h)$, the size of $\mathcal{S}$ is $O(h)$. Then, we insert the obstacle edges into $\mathcal{S}$ to build a conforming subdivision $\mathcal{S}'$ of the free space. The subdivision $\mathcal{S}'$ has $O(h)$ cells (in contrast, the conforming subdivision of the free space in the work of Hershberger and Suri [25] has $O(n)$ cells). Unlike the subdivision in their work [25] where each cell is of constant size, here the size of each cell of $\mathcal{S}'$ may not be constant but its boundary consists of $O(1)$ transparent edges and $O(1)$ convex chains (each of which belongs to the boundary of an obstacle of $\mathcal{P}$). Like the subdivision in the work of Hershberger and Suri [25], each transparent edge e of $\mathcal{S}'$ has a well-covering region $\mathcal{U}(e)$. In particular, for each transparent edge f on the boundary of $\mathcal{U}(e)$, the shortest path distance between e and f is at least $2 \cdot \max\{|e|, |f|\}$. Using $\mathcal{S}'$ as a guidance, we

[1] If randomization is allowed, the algorithm of Clarkson et al. [11] can compute a triangulation in $O(n \log^* n + h \log h)$ expected time. If all obstacles of $\mathcal{P}$ are convex, the triangulation can be done in $O(n + h \log h)$ time [27].

run the continuous Dijkstra algorithm as in the work of Hershberger and Suri [25] to expand the wavefront, starting from the source point s. A main challenge our algorithm needs to overcome (which is also a main difference between our algorithm and that in their work [25]) is that each cell in our subdivision $\mathcal{S}'$ may not be of constant size. One critical property our algorithm relies on is that the boundary of each cell of $\mathcal{S}'$ has $O(1)$ convex chains. Our strategy is to somehow treat each such convex chain as a whole. We also borrow some idea from the algorithm of Hershberger et al. [26] for computing shortest paths among curved obstacles. To guarantee the $O(n + h \log h)$ time, some global charging analysis is used. In addition, the tentative prune-and-search technique of Kirkpatrick and Snoeyink [33] is applied to perform certain operations related to bisectors, in logarithmic time each. Finally, the techniques presented in prior work [48] are utilized to reduce the space to $O(n)$. All these efforts lead to an $O(n + h \log h)$ time and $O(n)$ space algorithm to construct the shortest path map $SPM(s)$ for the convex case.

The General Case. We extend the convex case algorithm to the general case where obstacles may not be convex. To this end, we resort to the extended corridor structure of $\mathcal{P}$, which was used before for reducing the time complexities from n to h (e.g., [4, 7–9, 29–31, 39]). The structure partitions the free space $\mathcal{F}$ into an ocean $\mathcal{M}$, $O(n)$ bays, and $O(h)$ canals. Each bay is a simple polygon that shares an edge with $\mathcal{M}$. Each canal is a simple polygon that shares two edges with $\mathcal{M}$. But two bays or two canals, or a bay and a canal, do not share any edge. A common edge of a bay (or canal) with $\mathcal{M}$ is called a *gate*. Thus, each bay has one gate and each canal has two gates. Further, $\mathcal{M}$ is bounded by $O(h)$ convex chains. An important property related to shortest paths is that if both s and t are in $\mathcal{M}$, then any shortest s-t path must be in the union of $\mathcal{M}$ and all corridor paths, each of which is contained in a canal. As the boundary of $\mathcal{M}$ consists of $O(h)$ convex chains, by incorporating all corridor paths, we can easily extend our convex case algorithm to computing $SPM(\mathcal{M})$, the shortest path map $SPM(s)$ restricted to $\mathcal{M}$ (i.e., $SPM(\mathcal{M}) = SPM(s) \cap \mathcal{M}$). To compute the entire map $SPM(s)$, we expand $SPM(\mathcal{M})$ to all bays and canals through their gates. For this, we process each bay/canal individually. For each bay/canal C, expanding the map into C is actually a special case of the (additively)-weighted geodesic Voronoi diagram problem on a simple polygon where all sites are outside C and can influence C only through its gates. In summary, after a triangulation of $\mathcal{F}$ is given, building $SPM(\mathcal{M})$ takes $O(n + h \log h)$ time, and expanding $SPM(\mathcal{M})$ to all bays and canals takes additional $O(n + h \log h)$ time. The space of the algorithm is bounded by $O(n)$.

It should be noted that a fundamental difference between our algorithm and the $O(n + h \log h \log n)$-time algorithm of Inkulu et al. [28] is that our algorithm follows the framework of Hershberger and Suri [25] (as well as that of Hershberger et al. [26]) by using a conforming subdivision of the free space as an underlying structure to guide the wavefront expansion, whereas the algorithm of Inkulu et al. [28] does not use the conforming subdivision (it instead uses the triangulation as well as the corridor structure of the free space). In addition, our algorithm computes a shortest path map for the source point s, whereas the algorithm of Inkulu et al. [28] computes a shortest path from s to a single point t.

Outline. The rest of thearticle is organized as follows. Section 2 defines notation and introduces some concepts. Section 3 presents the algorithm for the convex case. The general case is discussed in Section 4.

2 PRELIMINARIES

For any two points a and b in the plane, denote by $\overline{ab}$ the line segment with a and b as endpoints; denote by $|\overline{ab}|$ the length of the segment.

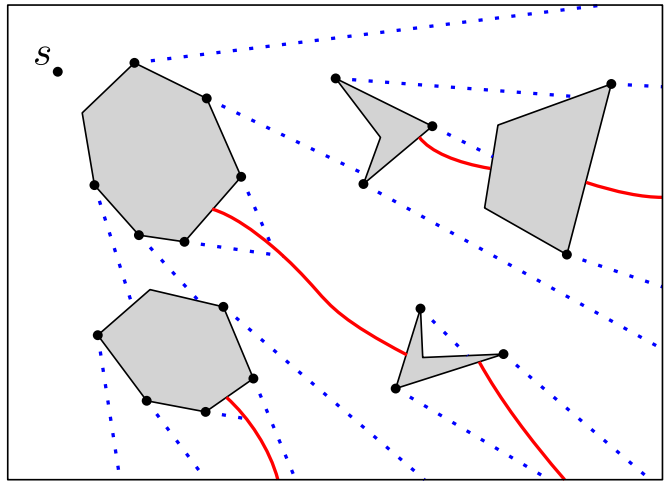

Fig. 2. Illustrating the shortest path map $SPM(s)$. The solid (red) curves are walls, and the (blue) dotted segments are windows. The anchor of each cell is also shown with a black point.

For any two points s and t in the free space $\mathcal{F}$, we use $\pi(s,t)$ to denote a shortest path from s to t in $\mathcal{F}$. In the case where shortest paths are not unique, $\pi(s,t)$ may refer to an arbitrary one. Denote by $d(s,t)$ the length of $\pi(s,t)$; we call $d(s,t)$ the *geodesic distance* between s and t. For two line segments e and f in $\mathcal{F}$, their geodesic distance is defined to be the minimum geodesic distance between any point on e and any point on f (i.e., $\min_{s\in e, t\in f} d(s,t)$); by slightly abusing the notation, we use $d(e,f)$ to denote their geodesic distance. For any path π in the plane, we use $|\pi|$ to denote its length.

For any compact region A in the plane, let ∂A denote its boundary. We use $\partial\mathcal{P}$ to denote the union of the boundaries of all obstacles of $\mathcal{P}$.

Throughout the article, we use s to refer to the source point. For convenience, we consider s as a degenerate obstacle in $\mathcal{P}$. We often refer to the vertices of $\mathcal{P}$ as *obstacle vertices* and refer to the edges of $\mathcal{P}$ as *obstacle edges*. For any point $t \in \mathcal{F}$, we call the adjacent vertex of t in $\pi(s,t)$ the *anchor* of t in $\pi(s,t)$.[2]

The *shortest path map* $SPM(s)$ of s is a decomposition of the free space $\mathcal{F}$ into maximal regions such that all points in each region R have the same anchor [25, 36] in their shortest paths from s (Figure 2). Each edge of $SPM(s)$ is either an obstacle edge fragment or a *bisecting curve*,[3] which is the locus of points p with $d(s,u) + |\overline{pu}| = d(s,v) + |\overline{pv}|$ for two obstacle vertices u and v. Each bisecting curve is in general a hyperbola; a special case happens if one of u and v is the anchor of the other, in which case their bisecting curve is a straight line. Following the notation in the work of Eriksson-Bique et al. [17], we differentiate between two types of bisecting curves: *walls* and *windows*. A bisecting curve of $SPM(s)$ is a *wall* if there exist two topologically different shortest paths from s to each point of the edge; otherwise (i.e., the preceding special case), it is a *window* (e.g., see Figure 2).

We make a general position assumption that for each obstacle vertex v, there is a unique shortest path from s to v, and for any point p in the plane, there are at most three different shortest paths from s to p. The assumption assures that each vertex of $SPM(s)$ has degree at most 3, and there

[2]Usually "predecessor" is used in the literature instead of "anchor," but here we reserve "predecessor" for other purposes.
[3]This is usually called *bisector* in the literature. Here we reserve the term *bisector* to be used later.

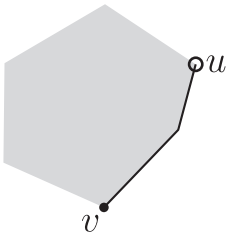

Fig. 3. Illustrating an elementary chain (the thick segments), which contains the vertex v but not u.

are at most three bisectors of $SPM(s)$ intersecting at a common point, which is sometimes called a *triple point* in the literature [17].

A curve in the plane is *x-monotone* if its intersection with any vertical line is connected; the *y-monotone* is defined similarly. A curve is xy-monotone if it is both x- and y-monotone.

The following observation will be used throughout the article without explicitly mentioning it again.

Observation 1. $n + h \log n = O(n + h \log h)$.

Proof. Indeed, if $h < n/\log n$, then $n + h \log n = \Theta(n)$, which is $O(n + h \log h)$; otherwise, $\log n = O(\log h)$ and $n + h \log n = O(n + h \log h)$. □

3 THE CONVEX CASE

In this section, we present our algorithm for the convex case where all obstacles of $\mathcal{P}$ are convex. The algorithm will be extended to the general case in Section 4.

For each obstacle $P \in \mathcal{P}$, the topmost, bottommost, leftmost, and rightmost vertices of P are called *rectilinear extreme vertices*. The four rectilinear extreme vertices partition ∂P into four portions, and each portion is called an *elementary chain*, which is convex and xy-monotone. For technical reasons that will be clear later, we assume that each rectilinear extreme vertex v belongs to the elementary chain counterclockwise of v with respect to the obstacle (i.e., v is the clockwise endpoint of the chain (Figure 3)). We use *elementary chain fragment* to refer to a portion of an elementary chain.

We introduce some notation in the following that is similar in spirit to those from Hershberger et al. [26] for shortest paths among curved obstacles.

Consider a shortest path $\pi(s, p)$ from s to a point p in the free space $\mathcal{F}$. It is not difficult to see that $\pi(s, p)$ is a sequence of elementary chain fragments and common tangents between obstacles of $\mathcal{P} \cup \{p\}$. We define the *predecessor* of p, denoted by $pred(p)$, to be the initial vertex of the last elementary chain fragment in $\pi(s, p)$ (Figure 4(a)). Note that since each rectilinear extreme vertex belongs to a single elementary chain, $pred(p)$ in $\pi(s, p)$ is unique. A special case happens if p is a rectilinear extreme vertex and $\pi(s, p)$ contains a portion of an elementary chain A clockwise of p. In this case, we let $pred(p)$ be endpoint of the fragment of A in $\pi(s, p)$ other than p (e.g., see Figure 4 (b)); in this way, $pred(p)$ is unique in $\pi(s, p)$. Note that p may still have multiple predecessors if there are multiple shortest paths from s to p. Intuitively, the reason we define predecessors as in the preceding is to treat each elementary chain somehow as a whole, which is essential for reducing the runtime of the algorithm from n to h.

The rest of this section is organized as follows. In Section 3.1, we compute a conforming subdivision $\mathcal{S}'$ of the free space $\mathcal{F}$. Section 3.2 introduces some basic concepts and notation for our algorithm. The wavefront expansion algorithm is presented in Section 3.3, with two key subroutines of the algorithm described in Sections 3.4 and 3.5, respectively. Section 3.6 analyzes the time

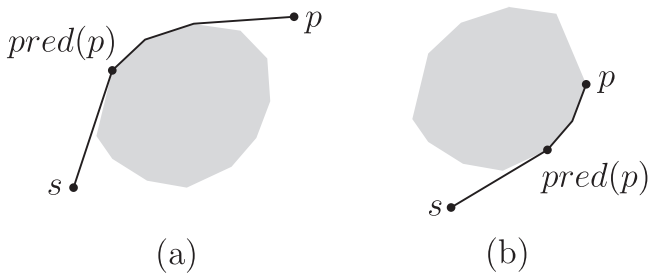

Fig. 4. Illustrating the predecessor.

complexity of the algorithm, where a technical lemma is proved separately in Section 3.7. Using the information computed by the wavefront expansion algorithm, Section 3.8 constructs the shortest path map $SPM(s)$. The overall algorithm runs in $O(n + h \log h)$ time and $O(n + h \log h)$ space. Section 3.9 reduces the space to $O(n)$ while keeping the same runtime, by using the techniques from prior work [48].

3.1 Computing a Conforming Subdivision of the Free Space

Let $\mathcal{V}$ denote the set of the rectilinear extreme vertices of all obstacles of $\mathcal{P}$. Hence, $|\mathcal{V}| = O(h)$. Using the algorithm of algorithm of Hershberger and Suri [25] (called the *HS* algorithm), we build a conforming subdivision $\mathcal{S}$ with respect to the vertices of $\mathcal{V}$, without considering the obstacle edges.

The subdivision $\mathcal{S}$, which is of size $O(h)$, is a quad-tree-style subdivision of the plane into $O(h)$ cells. Each cell of $\mathcal{S}$ is a square or a square annulus (i.e., an outer square with an inner square hole). Each vertex of $\mathcal{V}$ is contained in the interior of a square cell, and each square cell contains at most one vertex of $\mathcal{V}$. Each edge e of $\mathcal{S}$ is axis-parallel and *well-covered*—that is, there exists a set $C(e)$ of $O(1)$ cells of $\mathcal{S}$ such that their union $\mathcal{U}(e)$ contains e with the following properties: (1) the total complexity of all cells of $C(e)$ is $O(1)$, and thus the size of $\mathcal{U}(e)$ is $O(1)$; (2) for any edge f of $\mathcal{S}$ that is on $\partial\mathcal{U}(e)$ or outside $\mathcal{U}(e)$, the Euclidean distance between e and f (i.e., the minimum $|\overline{pq}|$ among all points $p \in e$ and $q \in f$) is at least $2 \cdot \max\{|e|, |f|\}$; and (3) $\mathcal{U}(e)$, which is called the *well-covering region* of e, contains at most one vertex of $\mathcal{V}$. In addition, each cell c of $\mathcal{S}$ has $O(1)$ edges on its boundary with the following *uniform edge property*: the lengths of the edges on the boundary of c differ by at most a factor of 4, regardless of whether c is a square or square annulus.

The subdivision $\mathcal{S}$ can be computed in $O(h \log h)$ time and $O(h)$ space [25].

Next we insert the obstacle edges into $\mathcal{S}$ to produce a conforming subdivision $\mathcal{S}'$ of the free space $\mathcal{F}$. In $\mathcal{S}'$, there are two types of edges: those introduced by the subdivision construction (which are in the interior of $\mathcal{F}$ except possibly their endpoints) and the obstacle edges; we call the former the *transparent edges* (which are axis-parallel) and the latter the *opaque edges*. The definition of $\mathcal{S}'$ is similar to the conforming subdivision of the free space used in the HS algorithm. A main difference is that here endpoints of each obstacle edge may not be in $\mathcal{V}$, a consequence of which is that each cell of $\mathcal{S}'$ may not be of constant size (while each cell in the subdivision of the HS algorithm is of constant size). However, each cell c of $\mathcal{S}'$ has the following property that is critical to our algorithm: the boundary ∂c consists of $O(1)$ transparent edges and $O(1)$ convex chains (each of which is a portion of an elementary chain (Figure 5)).

More specifically, $\mathcal{S}'$ is a subdivision of $\mathcal{F}$ into $O(h)$ cells. Each cell of $\mathcal{S}'$ is one of the connected components formed by intersecting $\mathcal{F}$ with an axis-parallel rectangle (which is the union of a set of adjacent cells of $\mathcal{S}$) or a square annulus of $\mathcal{S}$. Each cell of $\mathcal{S}'$ contains at most one vertex of $\mathcal{V}$. Each vertex of $\mathcal{V}$ is incident to a transparent edge. Each transparent edge e of $\mathcal{S}'$ is *well-covered*—that

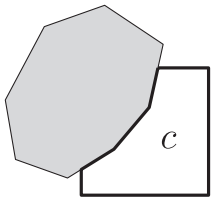

Fig. 5. Illustrating a cell c of $\mathcal{S}'$, which is bounded by bold segments. The gray region is an obstacle.

is, there exists a set $C(e)$ of $O(1)$ cells whose union $\mathcal{U}(e)$ contains e with the following property: for each transparent edge f on $\partial\mathcal{U}(e)$, the geodesic distance $d(e, f)$ between e and f is at least $2 \cdot \max\{|e|, |f|\}$. The region $\mathcal{U}(e)$ is called the *well-covering region* of e and contains at most one vertex of $\mathcal{V}$. Note that $\mathcal{S}'$ has $O(h)$ transparent edges.

In the following, we show how $\mathcal{S}'$ is produced from $\mathcal{S}$. The procedure is similar to that in the HS algorithm. We overlay the obstacle edges on top of $\mathcal{S}$ to obtain a subdivision $\mathcal{S}_{overlay}$. Because each edge of $\mathcal{S}$ is axis-parallel and all obstacle edges constitute a total of $O(h)$ elementary chains, each of which is xy-monotone, $\mathcal{S}_{overlay}$ has $O(h^2)$ faces. We say that a face of $\mathcal{S}_{overlay}$ is *interesting* if its boundary contains a vertex of $\mathcal{V}$ or a vertex of $\mathcal{S}$. We keep intact the interesting faces of $\mathcal{S}_{overlay}$ while deleting every edge fragment of $\mathcal{S}$ not on the boundary of any interesting cell. Further, for each cell c containing a vertex $v \in \mathcal{V}$, we partition c by extending vertical edges from v until the boundary of c. This divides c into at most three subcells. Finally we divide each of the two added edges incident to v into segments of length at most δ, where δ is the length of the shortest edge on the boundary of c. By the uniform edge property of $\mathcal{S}$, ∂c has $O(1)$ edges, whose lengths differ by at most a factor of 4; hence, dividing the edges incident to v as earlier produces only $O(1)$ vertical edges. The resulting subdivision is $\mathcal{S}'$.

As mentioned previously, the essential difference between our subdivision $\mathcal{S}'$ and the one in the HS algorithm is that the role of an obstacle edge in the HS algorithm is replaced by an elementary chain. Therefore, each opaque edge in the subdivision of the HS algorithm becomes an elementary chain fragment in our case. Hence, by the same analysis as in the HS algorithm (see Lemma 2.2 [25]), $\mathcal{S}'$ has the properties as described earlier and the well-covering region $\mathcal{U}(e)$ of each transparent edge e of $\mathcal{S}'$ is defined in the same way as in the HS algorithm.

The following lemma computes $\mathcal{S}'$. It should be noted that although $\mathcal{S}'$ is defined with the help of $\mathcal{S}_{overlay}$, $\mathcal{S}'$ is constructed directly without computing $\mathcal{S}_{overlay}$ first.

Lemma 3.1. *The conforming subdivision $\mathcal{S}'$ can be constructed in $O(n + h \log h)$ time and $O(n)$ space.*

Proof. We first construct $\mathcal{S}$ in $O(h \log h)$ time and $O(h)$ space [25]. In the following, we construct $\mathcal{S}'$ by inserting the obstacle edges into $\mathcal{S}$. The algorithm is similar to that in the HS algorithm (see Lemma 2.3 [25]). The difference is that we need to handle each elementary chain as a whole.

We first build a data structure so that for any query horizontal ray with origin in $\mathcal{F}$, the first obstacle edge of $\mathcal{P}$ hit by it can be computed in $O(\log n)$ time. This can be done by building a horizontal decomposition of $\mathcal{F}$—that is, extend a horizontal segment from each vertex until it hits $\partial\mathcal{P}$. As all obstacles of $\mathcal{P}$ are convex, the horizontal decomposition can be computed in $O(n + h \log h)$ time and $O(n)$ space [27]. By building a point location data structure [16, 32] on the horizontal decomposition in additional $O(n)$ time, each horizontal ray shooting can be answered in $O(\log n)$ time. Similarly, we can construct the vertical decomposition of $\mathcal{F}$ in $O(n + h \log h)$ time and $O(n)$ space so that each vertical ray shooting can be answered in $O(\log n)$ time.

The edges of $\mathcal{S}'$ are obstacle edges, transparent edges incident to the vertices of $\mathcal{S}$, and transparent edges subdivided on the vertical segments incident to the vertices of $\mathcal{V}$. To identify the second type of edges, we trace the boundary of each interesting cell separately. Starting from a vertex v of $\mathcal{S}$, we trace along each edge incident to v. Using the preceding ray-shooting data structure, we determine whether the next cell vertex is a vertex of $\mathcal{S}$ or the first point on $\partial\mathcal{P}$ hit by the ray. As $\mathcal{S}$ has $O(h)$ edges and vertices, this tracing takes $O(h \log n)$ time in total. Tracing along obstacle edges is done by starting from each vertex of $\mathcal{V}$ and following each of its incident elementary chains. For each elementary chain, the next vertex is either the next obstacle vertex on e, where e is the current tracing edge of the elementary chain, or the intersection of e with a transparent edge of the current cell. Hence, tracing all elementary chains takes $O(n)$ time in total. The third type of edges can be computed in linear time by local operations on each cell containing a vertex of $\mathcal{V}$.

Finally, we assemble all these edges together to obtain an adjacency structure for $\mathcal{S}'$. For this, we could use a plane sweep algorithm as in the HS algorithm. However, that would take $O(n \log n)$ time, as there are $O(n)$ edges in $\mathcal{S}'$. To obtain an $O(n + h \log h)$ time algorithm, we propose the following approach. During the preceding tracing of elementary chains, we record the fragment of the chain that lies in a single cell. Since each such portion may not be of constant size, we represent it by a line segment connecting its two endpoints; note that this segment does not intersect any other cell edges because the elementary chain fragment is on the boundary of an obstacle (and thus the segment is inside the obstacle). Then, we apply the plane sweep algorithm to stitch all edges with each elementary chain fragment in a cell replaced by a segment as earlier. The algorithm takes $O(h \log h)$ time and $O(h)$ space. Finally, for each segment representing an elementary chain portion, we locally replace it by the chain fragment in linear time. Hence, this step takes $O(n)$ time altogether for all such segments. As such, the total time for computing the adjacency information for $\mathcal{S}'$ is $O(n + h \log n)$, which is $O(n + h \log h)$. Clearly, the space complexity of the algorithm is $O(n)$. □

3.2 Basic Concepts and Notation

Our shortest path algorithm uses the continuous Dijkstra method. The algorithm initially generates a wavefront from s, which is a circle centered at s. During the algorithm, the wavefront consists of all points of $\mathcal{F}$ with the same geodesic distance from s (Figure 6). We expand the wavefront until all points of the free space are covered. The conforming subdivision $\mathcal{S}'$ is utilized to guide the wavefront expansion. Our wavefront expansion algorithm follows the high-level scheme as the HS algorithm. The difference is that our algorithm somehow considers each elementary chain as a whole, which is in some sense similar to the algorithm of Hershberger et al. [26] (called the *HSY* algorithm). Indeed, the HSY algorithm considers each xy-monotone convex arc as a whole, but the difference is that each arc in the HSY algorithm is of constant size, whereas in our case each elementary chain may not be of constant size. As such, techniques from both the HS algorithm and the HSY algorithm are borrowed.

We use τ to denote the geodesic distance from s to all points in the wavefront. One may also think of τ as a parameter representing time. The algorithm simulates the expansion of the wavefront as time increases from 0 to ∞. The wavefront comprises a sequence of *wavelets*, each emanating from a *generator*. In the HS algorithm, a generator is simply an obstacle vertex. Here, since we want to treat an elementary chain as a whole, similar to the HSY algorithm (also similar to the concept "bunch" introduced in the work of Inkulu et al. [28]), we define a generator as a couple $\alpha = (A, a)$, where A is an elementary chain and a is an obstacle vertex on A, and further a clockwise or counterclockwise direction of A is designated for α; a has a weight $w(a)$ (one may consider $w(a)$ as the geodesic distance between s and a). We call a the *initial vertex* of the generator α.

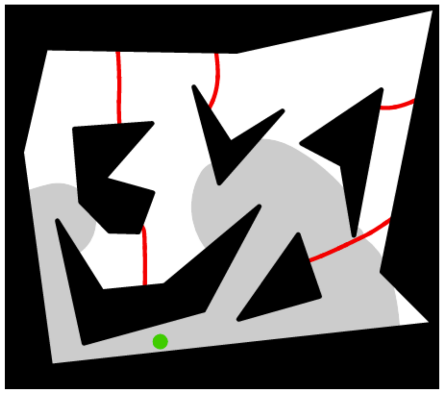

Fig. 6. Illustrating the wavefront. The black region are obstacles. The green point is s. The red curves are bisecting curves of $SPM(s)$. The gray region is the free space that has been covered by the wavefront. The boundary between the white region and the gray region is the wavefront. The figure is generated using the applet from Hershberger et al. [23].

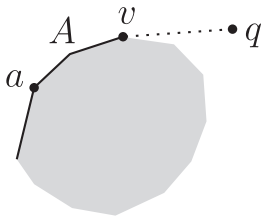

Fig. 7. Illustrating a generator $\alpha = (A, a)$. A consists of the thick segments and is designated in the clockwise direction around the obstacle. q is a reachable point through vertex v.

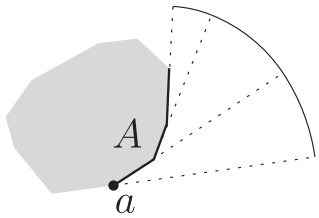

Fig. 8. Illustrating the wavelet generated by $\alpha = (A, q)$, designated in the counterclockwise direction. The wavelet has three pieces each of which is a circular arc.

We say a point q is *reachable* by a generator $\alpha = (A, a)$ if one can draw a path in $\mathcal{F}$ from a to q by following A in the designated direction to a vertex v on A such that $\overline{vq}$ is tangent to A and then following the segment $\overline{vq}$ (Figure 7). The (weighted) distance between the generator α and q is the length of this path plus $w(a)$; by slightly abusing the notation, we use $d(\alpha, q)$ to denote the distance. From the definition of reachable points, the vertex a partitions A into two portions and only the portion following the designated direction is relevant (e.g., in Figure 7, only the portion containing the vertex v is relevant). Henceforth, unless otherwise stated, we use A to refer to its relevant portion only and we call A the *underlying chain* of α. In this way, the initial vertex a becomes an endpoint of A. For convenience, sometimes we do not differentiate α and A. For example, when we say "the tangent from q to α," we mean "the tangent from q to A"; in addition, when we say "a vertex of α," we mean "a vertex of A."

The wavefront can thus be represented by the sequence of generators of its wavelets. A wavelet generated by a generator $\alpha = (A, a)$ at time τ is a contiguous set of reachable points q such that $d(\alpha, q) = \tau$ and $d(\alpha', q) \geq \tau$ for all other generators α' in the wavefront; we also say that q is claimed by α. Note that as A may not be of constant size, a wavelet may not be of constant size either; it actually consists of a contiguous sequence of circular arcs centered at the obstacle vertices A (Figure 8). If a point q is claimed by α, then $d(s, q) = d(\alpha, q) = \tau$ and the predecessor $pred(q)$ of q is a; sometimes for convenience, we also say that the generator α is the predecessor

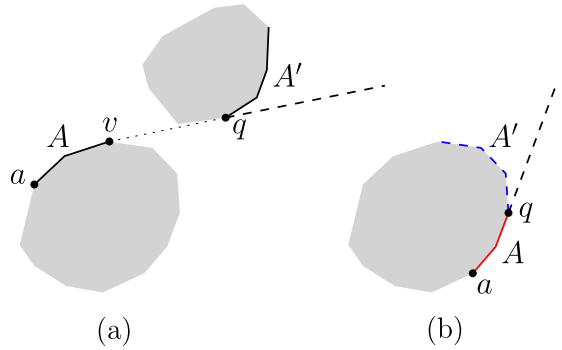

Fig. 9. Illustrating a new generator $\alpha' = (A', q)$. (a) A general case where both A and A' are marked with thick segments. (b) A special case where A is marked with solid (red) segments and A' is marked with dashed (blue) segments.

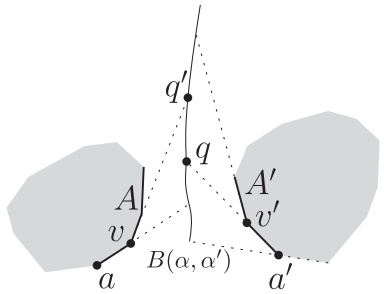

Fig. 10. Illustrating the bisector $B(\gamma, \gamma')$ defined by two generators $\gamma = (A, q)$ and $\gamma = (A', q')$. The portion between q and q' is a hyperbola defined by v and v'.

of q. If q is on an elementary chain A' and the tangent from q to a is also tangent to A', then a new generator (A', q) is added to the wavefront (Figure 9(a)). A special case happens when q is the counterclockwise endpoint of A (and thus q does not belong to A); in this case, a new generator $\alpha' = (A', q)$ is also added, where A' is the elementary chain that contains q (e.g., see Figure 9(b)).

As τ increases, the points bounding the adjacent wavelets trace out the bisectors that form the edges of the shortest path map $SPM(s)$. The *bisector* between the wavelets of two generators α and α', denoted by $B(\alpha, \alpha')$, consists of points q with $d(\alpha, q) = d(\alpha', q)$. Note that since α and α' may not be of constant size, $B(\alpha, \alpha')$ may not be of constant size either (in contrast, the bisector in HSY's algorithm is always of constant size; note that a similar concept "I-curves" is introduced in the work of Inkulu et al. [28]). More specifically, $B(\alpha, \alpha')$ has multiple pieces, each of which is a hyperbola defined by two obstacle vertices $v \in \alpha$ and $v' \in \alpha'$ such that the hyperbola consists of all points that have two shortest paths from s with v and v' as the anchors in the two paths, respectively (Figure 10). A special case happens if α' is a generator created by the wavelet of α such as that illustrated in Figure 9(a), then $B(\alpha, \alpha')$ is the half-line extended from q along the direction from v to q (the dashed segment in Figure 9(a)); we call such a bisector an *extension bisector*. Note that in the case illustrated in Figure 9(b), $B(\alpha, \alpha')$, which is also an extension bisector, is the half-line extended from q along the direction from v to q (the dashed segment in Figure 9(b)), where v is the obstacle vertex adjacent to q in A.

A wavelet gets eliminated from the wavefront if the two bisectors bounding it intersect, which is called a *bisector event*. Specifically, if α_1, α, and α_2 are three consecutive generators of the

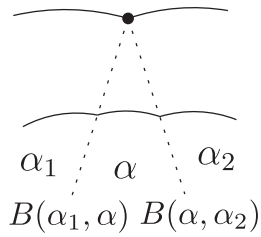

Fig. 11. Illustrating the intersection of two bisectors.

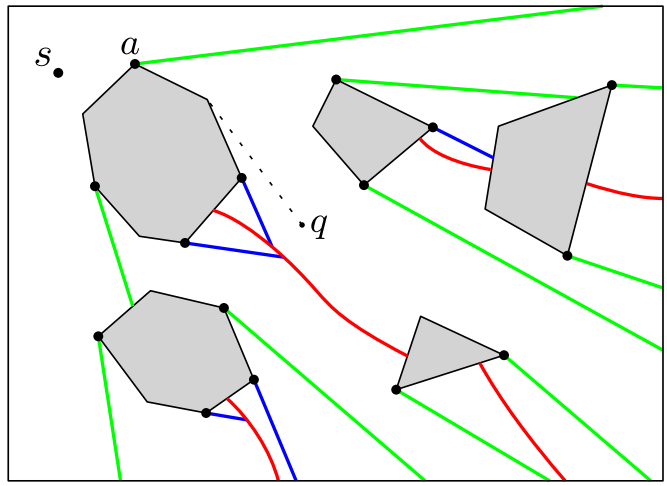

Fig. 12. Illustrating the map $SPM'(s)$. The red curves are non-extension bisectors. The green and blue segments are extension bisectors, where the green and blue ones are Type (a) and (b) as illustrated in Figure 9. The predecessor in each cell is also shown with a black point. For example, all points in the cell containing q have a as their predecessor.

wavefront, the wavelet generated by α will be eliminated when $B(\alpha_1, \alpha)$ intersects $B(\alpha, \alpha_2)$ (Figure 11). Wavelets are also eliminated by collisions with obstacles and other wavelets in front of it. If a bisector $B(\alpha, \alpha')$ intersects an obstacle, their intersection is also called a *bisector event*.

Let $SPM'(s)$ be the subdivision of $\mathcal{F}$ by the bisectors of all generators (Figure 12). The intersections of bisectors and intersections between bisectors and obstacle edges are vertices of $SPM'(s)$. Each bisector connecting two vertices is an edge of $SPM'(s)$, called a *bisector edge*. As discussed before, a bisector, which consists of multiple hyperbola pieces, may not be of constant size. Hence, a bisector edge e of $SPM'(s)$ may not be of constant size. For differentiation, we call each hyperbola piece of e a *hyperbolic-arc*. In addition, if the boundary of an obstacle P contains more than one vertex of $SPM'(s)$, then the chain of edges of ∂P connecting two adjacent vertices of $SPM'(s)$ also forms an edge of $SPM'(s)$, called a *convex chain edge*.

The following lemma shows that $SPM'(s)$ is very similar to $SPM(s)$. Refer to Figure 13 for $SPM(s)$.

Lemma 3.2. *Each extension bisector edge of $SPM'(s)$ is a window of $SPM(s)$. The union of all non-extension bisector edges of $SPM'(s)$ is exactly the union of all walls of $SPM(s)$. $SPM'(s)$ can be obtained from $SPM(s)$ by removing all windows except those that are extension bisectors of $SPM'(s)$.*

Proof. We first prove that each extension bisector edge e of $SPM'(s)$ is a window of $SPM(s)$. By definition, one endpoint of e, denoted by u, is an obstacle vertex such that e is a half-line extended

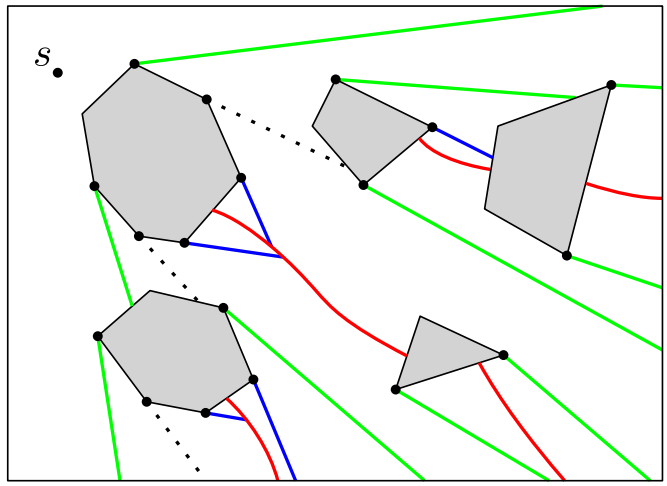

Fig. 13. Illustrating the map $SPM(s)$. The dashed segments are windows that are not in $SPM'(s)$; removing them becomes $SPM'(s)$. The anchor of each cell is also shown.

from u in the direction from v to u for another obstacle vertex v. Hence, for each point $p \in e$, there is a shortest s-p path $\pi(s,p)$ such that $\overline{vp}$ is a segment of $\pi(s,p)$. As $u \in \overline{vp}$, p must be on a window w of $SPM(s)$. This proves that $e \subseteq w$. However, for any point $p \in w$, there is a shortest s-p path $\pi(s,p)$ that contains $\overline{vp}$. Since $u \in \overline{vp}$, p must be on the extension bisector edge e of $SPM'(s)$, and thus $w \subseteq e$. Therefore, $e = w$. This proves that each extension bisector edge of $SPM'(s)$ is a window of $SPM(s)$.

We next prove that the union of all non-extension bisector edges of $SPM'(s)$ is exactly the union of all walls of $SPM(s)$:

- We first show that the union of all non-extension bisector edges of $SPM'(s)$ is a subset of the union of all walls of $SPM(s)$.
  Let q be a point on a non-extension bisector $B(\alpha,\alpha')$ of two generators $\alpha = (A,a)$ and $\alpha = (A',a')$. Let v and v' be the tangent points on A and A' from q, respectively. Hence, q has two shortest paths from s, one containing $\overline{qv}$ and the other containing $\overline{qv'}$. Since $B(\alpha,\alpha')$ is not an extension bisector, $v' \notin \overline{qv}$ and $v \notin \overline{qv'}$. This means that the two paths are combinatorially different, and thus q is on a wall of $SPM(s)$. This proves that the union of all non-extension bisector edges of $SPM'(s)$ is a subset of the union of all walls of $SPM(s)$.
- We then show that the union of all walls of $SPM(s)$ is a subset of the union of all non-extension bisector edges of $SPM'(s)$.
  Let q be a point on a wall of $SPM(s)$. By definition, there are two obstacle vertices v and v' such that there are two combinatorially different shortest paths from s to q whose anchors are v and v', respectively. Further, since the two paths are combinatorially different and also due to the general position assumption, $v' \notin \overline{qv}$ and $v \notin \overline{qv'}$. Therefore, q must be on the bisector edge of the two generators α and α' in $SPM'(s)$ whose underlying chains containing v and v', respectively. Further, since $v \notin \overline{qv'}$ and $v' \notin \overline{qv}$, the bisector of α and α' is not an extension bisector. This proves that the union of all walls of $SPM(s)$ is a subset of the union of all non-extension bisector edges of $SPM'(s)$.

The preceding proves that the first and second statements of the lemma. The third statement immediately follows the first two. □

Corollary 3.3. *The combinatorial size of $SPM'(s)$ is $O(n)$.*

Proof. By Lemma 3.2, $SPM'(s)$ is a subset of $SPM(s)$. As the combinatorial size of $SPM(s)$ is $O(n)$ [25, 36], the combinatorial size of $SPM'(s)$ is also $O(n)$. □

Lemma 3.4. *The subdivision $SPM'(s)$ has $O(h)$ faces, vertices, and edges. In particular, $SPM'(s)$ has $O(h)$ bisector intersections and $O(h)$ intersections between bisectors and obstacle edges.*

Proof. By Lemma 3.2, for any vertex of $SPM'(s)$ that is the intersection of two non-extension bisectors, it is also the intersection of two walls of $SPM(s)$, which is a *triple point*. Let $SPM''(s)$ refer to $SPM'(s)$ excluding all extension bisectors. We define a planar graph G corresponding to $SPM''(s)$ as follows. Each obstacle defines a node of G, and each triple point also defines a node of G. For any two nodes of G, if they are connected by a chain of bisector edges of $SPM'(s)$ such that the chain does not contain any other node of G, then G has an edge connecting the two nodes. It is proved in previous work [46] that G has $O(h)$ vertices, faces, and edges.

It is not difficult to see that a face of $SPM''(s)$ corresponds to a face of G and thus the number of faces of $SPM''(s)$ is $O(h)$. For each bisector intersection of $SPM''(s)$, it must be a triple point and thus it is also a node of G. As G has $O(h)$ nodes, the number of bisector intersections of $SPM''(s)$ is $O(h)$. For each intersection v between a bisector and an obstacle edge in $SPM''(s)$, G must has an edge e corresponding a chain of bisector edges and v is the endpoint of the chain; we charge v to e. As e as two incident nodes of G, e can be charged at most twice. Since G has $O(h)$ edges, the number of intersections between bisectors and obstacle edges in $SPM''(s)$ is $O(h)$.

We next prove that the total number of extension bisector edges of $SPM'(s)$ is $O(h)$. For each extension bisection edge e, it belongs to one of the two cases (a) and (b) illustrated in Figure 9. For Case (b), one endpoint of e is a rectilinear extreme vertex. Since each rectilinear extreme vertex can define at most two extension bisector edges and there are $O(h)$ rectilinear extreme vertices, the total number of Case (b) extension bisectors is $O(h)$. For Case (a), e is an extension of a common tangent of two obstacles; we say that e is *defined* by the common tangent. Note that all such common tangents are interior disjoint, as they belong to shortest paths from s encoded by $SPM'(s)$. We now define a planar graph G' as follows. The node set of G' consists of all obstacles of $\mathcal{P}$. Two obstacles have an edge in G' if they have at least one common tangent that defines an extension bisector. Since all such common tangents are interior disjoint, G' is a planar graph. Apparently, no two nodes of G' share two edges and no node of G' has a self-loop. Therefore, G' is a simple planar graph. Since G' has h vertices, the number of edges of G' is $O(h)$. By the definition of G', each pair of obstacles whose common tangents define extension bisectors have an edge in G'. Because each pair of obstacles can define at most four common tangents and thus at most four extension bisectors and also because G' has $O(h)$ edges, the total number of Case (a) extension bisectors is $O(h)$.

The bisector intersections of $SPM'(s)$ that are not vertices of $SPM''(s)$ involve extension bisectors of $SPM'(s)$. The intersections between bisectors and obstacle edges in $SPM'(s)$ that are not in $SPM''(s)$ also involve extension bisectors. Since all extension bisectors and all edges of $SPM''(s)$ are interior disjoint, each extension bisector can involve in at most two bisector intersections and at most two intersections between bisectors and obstacle edges. Because the total number of extension bisector edges of $SPM'(s)$ is $O(h)$, the number of bisector intersections involving extension bisectors is $O(h)$ and the number of intersections between extension bisectors and obstacle edges is also $O(h)$. Therefore, comparing to $SPM''(s)$, $SPM'(s)$ has $O(h)$ additional vertices and $O(h)$ additional edges. Note that the number of convex chain edges of $SPM'(s)$ is $O(h)$, for $SPM'(s)$ has $O(h)$ vertices. The lemma thus follows. □

Corollary 3.5. *There are $O(h)$ bisector events and $O(h)$ generators in $SPM'(s)$.*

Proof. Each bisector event is either a bisector intersection or an intersection between a bisector and an obstacle edge. By Lemma 3.4, there are $O(h)$ bisector intersections and $O(h)$ intersections between bisectors and obstacle edges. As such, there are $O(h)$ bisector events in $SPM'(s)$.

By definition, each face of $SPM'(s)$ has a unique generator. Since $SPM'(s)$ has $O(h)$ faces by Lemma 3.4, the total number of generators is $O(h)$. □

By the definition of $SPM'(s)$, each cell C of $SPM'(s)$ has a unique generator $\alpha = (A, a)$, all points of the cell are reachable from the generator, and a is the predecessor of all points of C (e.g., see Figure 12; all points in the cell containing q have a as their predecessor). Hence, for any point $q \in C$, we can compute $d(s, q) = d(\alpha, q)$ by computing the tangent from q to A. Thus, $SPM'(s)$ can also be used to answer shortest path queries. In fact, given $SPM'(s)$, we can construct $SPM(s)$ in additional $O(n)$ time by inserting the windows of $SPM(s)$ to $SPM'(s)$, as shown in the following lemma.

Lemma 3.6. *Given $SPM'(s)$, $SPM(s)$ can be constructed in $O(n)$ time (e.g., see Figures 12 and 13).*

Proof. It suffices to insert all windows of $SPM(s)$ (except those that are also extension bisectors of $SPM'(s)$) to $SPM'(s)$. We consider each cell C of $SPM'(s)$ separately. Let $\alpha = (A, a)$ be the generator of C. Our goal is to extend each obstacle edge of A along the designated direction of A until the boundary of C. To this end, since all points of C are reachable from α, the cell C is "star-shaped" with respect to the tangents of A (along the designated direction) and the extension of each obstacle edge of A intersects the boundary of C at most once. Hence, the order of the endpoints of these extensions is consistent with the order of the corresponding edges of A. Therefore, these extension endpoints can be found in order by traversing the boundary of C, which takes linear time in the size of C. Since the total size of all cells of $SPM'(s)$ is $O(n)$ by Corollary 3.3, the total time of the algorithm is $O(n)$. □

In light of Lemma 3.6, we will focus on computing the map $SPM'(s)$.

3.3 The Wavefront Expansion Algorithm

To simulate the wavefront expansion, we compute the wavefront passing through each transparent edge of the conforming subdivision $\mathcal{S}'$. As in the HS algorithm, since it is difficult to compute a true wavefront for each transparent edge e of $\mathcal{S}'$, a key idea is to compute two one-sided wavefronts (called *approximate wavefronts*) for e, each representing the wavefront coming from one side of e. Intuitively, an approximate wavefront from one side of e is what the true wavefront would be if the wavefront were blocked off at e by considering e as an opaque segment (with open endpoints).

In the following, unless otherwise stated, a wavefront at a transparent edge e of $\mathcal{S}'$ refers to an approximate wavefront. Let $W(e)$ denote a wavefront at e. To make the description concise, as there are two wavefronts at e, depending on the context, $W(e)$ may refer to both wavefronts—that is, the discussion on $W(e)$ applies to both wavefronts. For example, "compute the wavefronts $W(e)$" means "compute both wavefronts at e."

For each transparent edge e of $\mathcal{S}'$, define $input(e)$ as the set of transparent edges on the boundary of the well-covering region $\mathcal{U}(e)$, and define $output(e) = \{g \mid e \in input(g)\} \cup input(e)$.[4] Because $\partial\mathcal{U}(e')$ for each transparent edge e' of $\mathcal{S}'$ has $O(1)$ transparent edges, both $|input(e)|$ and $|output(e)|$ are $O(1)$.

The Wavefront Propagation and Merging Procedures. The wavefronts $W(e)$ at e are computed from the wavefronts at edges of $input(e)$; this guarantees the correctness because e is in $\mathcal{U}(e)$ (and thus any shortest path $\pi(s, p)$ must cross some edge $f \in input(e)$ for any point $p \in e$). After

[4]We include $input(e)$ in $output(e)$ mainly for the proof of Lemma 3.23, which relies on $output(e)$ having a cycle enclosing e.

the wavefronts $W(e)$ at e are computed, they will pass to the edges of $output(e)$. In addition, the geodesic distances from s to both endpoints of e will be computed. Recall that $\mathcal{V}$ is the set of rectilinear extreme vertices of all obstacles and each vertex of $\mathcal{V}$ is incident to a transparent edge of $\mathcal{S}'$. As such, after the algorithm is finished, geodesic distances from s to all vertices of $\mathcal{V}$ will be available. The process of passing the wavefronts $W(e)$ at e to all edges $g \in output(e)$ is called the *wavefront propagation procedure*, which will compute the wavefront $W(e, g)$, where $W(e, g)$ is the portion of $W(e)$ that passes to g through the well-covering region $\mathcal{U}(g)$ of g if $e \in input(g)$ and through $\mathcal{U}(e)$ otherwise (in this case, $g \in input(e)$); whenever the procedure is invoked on e, we say that e is *processed*. The wavefronts $W(e)$ at e are constructed by merging the wavefronts $W(f, e)$ for edges $f \in input(e)$; this procedure is called the *wavefront merging procedure*.

The Main Algorithm. The transparent edges of $\mathcal{S}'$ are processed in a rough time order. The wavefronts $W(e)$ of each transparent edge e are constructed at the time $\tilde{d}(s, e) + |e|$, where $\tilde{d}(s, e)$ is the minimum geodesic distance from s to the two endpoints of e. Define $covertime(e) = \tilde{d}(s, e) + |e|$. The value $covertime(e)$ will be computed during the algorithm. Initially, for each edge e whose well-covering region $\mathcal{U}(e)$ contains s, $W(e)$ and $covertime(e)$ are computed directly (and set $covertime(e) = \infty$ for all other edges); we refer to this step as the *initialization step*, which will be elaborated in the following. The algorithm maintains a timer τ and processes the transparent edges e of $\mathcal{S}'$ following the order of $covertime(e)$.

The main loop of the algorithm works as follows. As long as $\mathcal{S}'$ has an unprocessed transparent edge, we do the following. First, among all unprocessed transparent edges, choose the one e with minimum $covertime(e)$ and set $\tau = covertime(e)$. Second, call the wavefront merging procedure to construct the wavefronts $W(e)$ from $W(f, e)$ for all edges $f \in input(e)$ satisfying $covertime(f) < covertime(e)$; compute $d(s, v)$ from $W(e)$ for each endpoint v of e. Third, process e—that is, call the wavefront propagation procedure on $W(e)$ to compute $W(e, g)$ for all edges $g \in output(e)$; in particular, compute the time τ_g when the wavefronts $W(e)$ first encounter an endpoint of g and set $covertime(g) = \min\{covertime(g), \tau_g + |g|\}$.

The details of the wavefront merging procedure and the wavefront propagation procedure will be described in Sections 3.4 and 3.5, respectively.

The Initialization Step. We provide the details on the initialization step. Consider a transparent edge e whose well-covering region $\mathcal{U}(e)$ contains s. To compute $W(e)$, we only consider straight-line paths from s to e inside $\mathcal{U}(e)$. If $\mathcal{U}(e)$ does not have any island inside, then the points of e that can be reached from s by straight-line paths in $\mathcal{U}(e)$ form an interval of e and the interval can be computed by considering the tangents from s to the elementary chains on the boundary of $\mathcal{U}(e)$. Since $\mathcal{U}(e)$ has $O(1)$ cells of $\mathcal{S}'$, $\partial\mathcal{U}(e)$ has $O(1)$ elementary chain fragments and thus computing the interval on e takes $O(\log n)$ time. If $\mathcal{U}(e)$ has at least one island inside, then e may have multiple such intervals. As $\mathcal{U}(e)$ is the union of $O(1)$ cells of $\mathcal{S}'$, the number of islands inside $\mathcal{U}(e)$ is $O(1)$. Hence, e has $O(1)$ such intervals, which can be computed in $O(\log n)$ time altogether. These intervals form the wavefront $W(e)$—that is, each interval corresponds to a wavelet with generator s. From $W(e)$, the value $covertime(e)$ can be immediately determined. More specifically, for each endpoint v of e, if one of the wavelets of $W(e)$ covers v, then the segment $\overline{sv}$ is in $\mathcal{U}(e)$ and thus $d(s, v) = |\overline{sv}|$; otherwise, we set $d(s, v) = \infty$. In this way, we find an upper bound for $\tilde{d}(s, e)$ and set $covertime(e)$ to this upper bound plus $|e|$.

The Algorithm Correctness. At the time $\tilde{d}(s, e) + |e|$, all edges $f \in input(e)$ whose wavefronts $W(f)$ contribute a wavelet to $W(e)$ must have already been processed. This is due to the property of the well-covering regions of $\mathcal{S}'$ that $d(e, f) \geq 2 \cdot \max\{|e|, |f|\}$ since f is on $\partial\mathcal{U}(e)$. The proof is the same as that of Lemma 4.2 [25], so we omit it. Note that this also implies that the geodesic distance

$d(s, v)$ is correctly computed for each endpoint v of e. Therefore, after the algorithm is finished, geodesic distances from s to endpoints of all transparent edges of $\mathcal{S}'$ are correctly computed.

Artificial Wavelets. As in the HS algorithm, to limit the interaction between wavefronts from different sides of each transparent edge e, when a wavefront propagates across e (i.e., when $W(e)$ is computed in the wavefront merging procedure), an *artificial wavelet* is generated at each endpoint v of e, with weight $d(s, v)$. This is to eliminate a wavefront from one side of e if it arrives at e later than the wavefront from the other side of e. We will not explicitly discuss this anymore when we describe the algorithm.

Topologically Different Paths. In the wavefront propagation procedure to pass $W(e)$ to all edges $g \in output(e)$, $W(e)$ travels through the cells of the well-covering region $\mathcal{U}$ of either g or e. Since $\mathcal{U}$ may not be simply connected (e.g., the square annulus), there may be multiple topologically different shortest paths between e and g inside $\mathcal{U}$; the number of such paths is $O(1)$ as $\mathcal{U}$ is the union of $O(1)$ cells of $\mathcal{S}'$. We propagate $W(e)$ in multiple components of $\mathcal{U}$, each corresponding to a topologically different shortest path and defined by the particular sequence of transparent edges it crosses. These wavefronts are later combined in the wavefront merging step at g. This also happens in the initialization step, as discussed earlier.

Claiming a Point. During the wavefront merging procedure at e, we have a set of wavefronts $W(f, e)$ that reach e from the same side for edges $f \in input(e)$. We say that a wavefront $W(f, e)$ *claims* a point $p \in e$ if $W(f, e)$ reaches p before any other wavefront. Further, for each wavefront $W(f, e)$, the set of points on e claimed by it forms an interval (the proof is similar to that of Lemma 4.4 [25]). Similarly, a wavelet of a wavefront *claims* a point p of e if the wavelet reaches p before any other wavelet of the wavefront, and the set of points on e claimed by any wavelet of the wavefront also forms an interval. For convenience, if a wavelet claims a point, we also say that the generator of the wavelet claims the point.

Before presenting the details of the wavefront merging procedure and the wavefront propagation procedure in the next two sections, we first discuss the data structure (for representing elementary chains, generators, and wavefronts) and a monotonicity property of bisectors, which will be used later in our algorithm.

3.3.1 The Data Structure. We use an array to represent each elementary chain. Then, a generator $\alpha = (A, a)$ can be represented by recording the indices of the two end vertices of its underlying chain A. In this way, a generator takes $O(1)$ additional space to record and binary search on A can be supported in $O(\log n)$ time (e.g., given a point p, find the tangent from p to A). In addition, we maintain the lengths of the edges in each elementary chain so that given any two vertices of the chain, the length of the subchain between the two vertices can be obtained in $O(1)$ time. All this preprocessing can be done in $O(n)$ time and space for all elementary chains.

For a wavefront $W(e)$ of one side of e, it is a list of generators $\alpha_1, \alpha_2, \ldots$ ordered by the intervals of e claimed by these generators. Note that these generators are in the same side of e. Formally, we say that a generator α is in one side of e if the initial vertex of α lies in that side of the supporting line of e. We maintain these generators by a balanced binary search tree so that the following operations can be supported in logarithmic time each. Let W be a wavefront with generators $\alpha_1, \alpha_2, \ldots, \alpha_k$:

Insert: Insert a generator α to W. In our algorithm, α is inserted either in the front of α_1 or after α_k.

Delete: Delete a generator α_i from W, for any $1 \le i \le k$.

Split: Split W into two sublists at some generator α_i so that the first i generators form a wavefront and the rest form another wavefront.

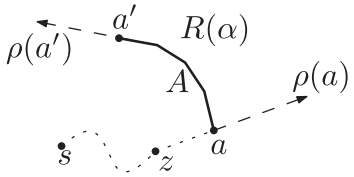

Fig. 14. Illustrating the reachable region $R(\alpha)$ of a generator $\alpha = (A, a)$.

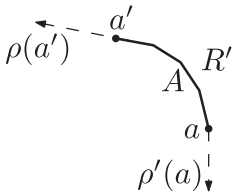

Fig. 15. Illustrating the ray $\rho'(a)$ and the region R'.

Concatenate: Concatenate W with another list W' of generators so that all generators of W are in the front of those of W' in the new list.

We will show later that each wavefront involved in our algorithm has $O(h)$ generators. Therefore, each of the preceding operations can be performed in $O(\log h)$ time. We make the tree fully persistent by path-copying [43]. In this way, each operation on the tree will cost $O(\log h)$ additional space but the older version of the tree will be kept intact (and operations on the old tree can still be supported).

3.3.2 *The Monotonicity Property of Bisectors.*

Lemma 3.7. *Suppose $\alpha_1 = (A_1, a_1)$ and $\alpha_2 = (A_2, a_2)$ are two generators on the same side of an axis-parallel line ℓ (i.e., a_1 and a_2 are on the same side of ℓ; it is possible that ℓ intersects A_1 and A_2). Then, the bisector $B(\alpha_1, \alpha_2)$ intersects ℓ at no more than one point.*

Proof. Consider a generator $\alpha = (A, a)$. Recall that a is an endpoint of A. Let a' be the other endpoint of A. We consider the general case where A has at least one edge, and thus $a \neq a'$; the other case can be handled analogously. We define $R(\alpha)$ as the set of points in the plane that are reachable from α (without considering any other obstacles), and we call it the *reachable region* of α. The reachable region $R(\alpha)$ is bounded by A, a ray $\rho(a)$ with origin a, and another ray $\rho(a')$ with origin a' (Figure 14). Specifically, $\rho(a)$ is along the direction from z to a, where z is the anchor of a in the shortest path from s to a; $\rho(a')$ is the ray from the adjacent obstacle vertex of a' on A' to a'.

We claim that the intersection $\ell \cap R(\alpha)$ must be an interval of ℓ. Indeed, since A is xy-monotone, without loss of generality, we assume that a' is to the northwest of a. Let $\rho'(a)$ be the vertically downward ray from a (Figure 15). Observe that the concatenation of $\rho(a')$, A, and $\rho'(a)$ is xy-monotone; let R' be the region of the plane to the right of their concatenation. Since the boundary of R' is xy-monotone and ℓ is axis-parallel, $R' \cap \ell$ must be an interval of ℓ. Notice that $\rho(a)$ must be in R', and thus it partitions R' into two subregions, one of which is $R(\alpha)$. Since ℓ intersects $\rho(a)$ at most once and $R' \cap \ell$ is an interval, $R(\alpha) \cap \ell$ must also be an interval.

According to the preceding claim, both $\ell \cap R(\alpha_1)$ and $\ell \cap R(\alpha_2)$ are intervals of ℓ. Let I denote the common intersection of the two intervals. Since $B(\alpha_1, \alpha_2) \subseteq R(\alpha_1)$ and $B(\alpha_1, \alpha_2) \subseteq R(\alpha_2)$, we obtain that $B(\alpha_1, \alpha_2) \cap \ell \subseteq I$.

Assume to the contrary that $B(\alpha_1, \alpha_2) \cap \ell$ contains two points, say, q_1 and q_2. Then, both points are in I. Let v_1 and u_1 be the tangents points of q_1 on A_1 and A_2, respectively. Let v_2 and u_2 be the tangents points of q_2 on A_1 and A_2, respectively (Figure 16). As $I = \ell \cap R(\alpha_1) \cap R(\alpha_2)$ and I contains both q_1 and q_2, if we move a point q from q_1 to q_2 on ℓ, the tangent from q to A_1 will continuously change from $\overline{q_1 v_1}$ to $\overline{q_2 v_2}$ and the tangent from q to A_2 will continuously change from $\overline{q_1 u_1}$ to $\overline{q_2 u_2}$. Since A_1 and A_2 are in the same side of ℓ and each of A_1 and A_2 has a fixed designated direction (i.e., either clockwise or counterclockwise), either $\overline{q_1 u_1}$ intersects $\overline{q_2 v_2}$ in their interiors or $\overline{q_1 v_1}$ intersects $\overline{q_2 u_2}$ in their interiors; without loss of generality, we assume that it is the former case.

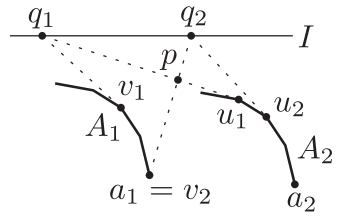

Fig. 16. Illustrating the proof of Lemma 3.7.

Let p be the intersection of $\overline{q_1u_1}$ and $\overline{q_2v_2}$ (e.g., see Figure 16). Since $q_1 \in B(\alpha_1, \alpha_2)$, points of $\overline{q_1u_1}$ other than q_1 have only one predecessor, which is a_2. As $p \in \overline{q_1u_1}$ and $p \neq q_1$, p has only one predecessor a_2. Similarly, since $q_2 \in B(\alpha_1, \alpha_2)$ and $p \in \overline{q_2v_2}$, a_1 is also p's predecessor. We thus obtain a contradiction. □

Note that similar monotonicity properties of bisectors were also used in previous work (e.g., [25, 26, 28]).

Corollary 3.8. *Suppose $\alpha_1 = (A_1, a_1)$ and $\alpha_2 = (A_2, a_2)$ are two generators both below a horizontal line ℓ. Then, the portion of the bisector $B(\alpha_1, \alpha_2)$ above ℓ is y-monotone.*

Proof. For any horizontal line ℓ' above ℓ, since both generators are below ℓ, they are also below ℓ'. By Lemma 3.7, $B(\alpha_1, \alpha_2) \cap \ell'$ is either empty or a single point. The corollary thus follows. □

3.4 The Wavefront Merging Procedure

In this section, we present the details of the wavefront merging procedure. Given all contributing wavefronts $W(f, e)$ of $f \in \mathit{input}(e)$ for $W(e)$, the goal of the procedure is to compute $W(e)$. The algorithm follows the high-level scheme of the HS algorithm (i.e., Lemma 4.6 of Hershberger and Suri [25]), but the implementation details are quite different.

We only consider the wavefronts $W(f, e)$ and $W(e)$ for one side of e since the algorithm for the other side is analogous. Without loss of generality, we assume that e is horizontal and all wavefronts $W(f, e)$ are coming from below e. We describe the algorithm for computing the interval of e claimed by $W(f, e)$ if only one other wavefront $W(f', e)$ is present. The common intersection of these intervals of all such f' is the interval of e claimed by $W(f, e)$. Since $|\mathit{input}(e)| = O(1)$, the number of such f' is $O(1)$.

We first determine whether the claim of $W(f, e)$ is to the left or right of that of $W(f', e)$. To this end, depending on whether both $W(f, e)$ and $W(f', e)$ reach the left endpoint v of e, there are two cases. Note that the intervals of e claimed by $W(f, e)$ and $W(f', e)$ are available from $W(f, e)$ and $W(f', e)$; let I_f and $I_{f'}$ denote these two intervals, respectively:

- If both I_f and $I_{f'}$ contain v (i.e., both $W(f, e)$ and $W(f', e)$ reach v), then we compute the (weighted) distances from v to the two wavefronts. This can be done as follows. Since $v \in I_f$, v must be reached by the leftmost generator α of $W(f, e)$. We compute the distance $d(\alpha, v)$ by computing the tangent from v to α in $O(\log n)$ time. Similarly, we compute $d(\alpha', v)$, where α' is leftmost generator of $W(f', e)$. If $d(\alpha, v) \leq d(\alpha', v)$, then the claim of $W(f, e)$ is to the left of that of $W(f', e)$; otherwise, the claim of $W(f, e)$ is to the right of that of $W(f', e)$.
- If both I_f and $I_{f'}$ do not contain v, then the order of the left endpoints of I_f and $I_{f'}$ gives the answer.

Without loss of generality, we assume that the claim of $W(f,e)$ is to the left of that of $W(f',e)$. We next compute the interval I of e claimed by $W(f,e)$ with respect to $W(f',e)$. Note that the left endpoint of I is the left endpoint of I_f. Hence, it remains to find the right endpoint of I, as follows.

Let ℓ_e be the supporting line of e. Let α be the rightmost generator of $W(f,e)$, and let α' be the leftmost generator of $W(f',e)$. Let q_1 be the left endpoint of the interval on e claimed by α in $W(f,e)$—that is, q_1 is the intersection of the bisector $B(\alpha_1,\alpha)$ and e, where α_1 is the left neighboring generator of α in $W(f,e)$. Similarly, let q_2 be the right endpoint of the interval on e claimed by α' in $W(f',e)$—that is, q_2 is the intersection of e and the bisector $B(\alpha',\alpha_1')$, where α_1' is the right neighboring generator of α' in $W(f',e)$. Let q_0 be the intersection of the bisector $B(\alpha,\alpha')$ and e. We assume that the three points q_i, $i=0,1,2$ are available, and we will discuss later that each of them can be computed in $O(\log n)$ time by a *bisector-line intersection operation* given in Lemma 3.10. If q_0 is between q_1 and q_2, then q_0 is the right endpoint of I and we can stop the algorithm. If q_0 is to the left of q_1, then we delete α from $W(f,e)$. If q_0 is to the right of q_2, then we delete α' from $W(f',e)$. In either case, we continue the same algorithm by redefining α or α' (and recomputing q_i for $i=0,1,2$).

Clearly, the preceding algorithm takes $O((1+k)\log n)$ time, where k is the number of generators that are deleted. We apply the algorithm on f and other f' in $input(e)$ to compute the corresponding intervals for f. The common intersection of all these intervals is the interval of e claimed by $W(f,e)$. We do so for each $f \in input(e)$, after which $W(e)$ is obtained. Since the size of $input(e)$ is $O(1)$, we obtain the following lemma.

Lemma 3.9. *Given all contributing wavefronts $W(f,e)$ of edges $f \in input(e)$ for $W(e)$, we can compute the interval of e claimed by each $W(f,e)$ and thus construct $W(e)$ in $O((1+k)\log n)$ time, where k is the total number of generators in all wavefronts $W(f,e)$ that are absent from $W(e)$.*

Lemma 3.10 (Bisector-Line Intersection Operation). *Each bisector-line intersection operation can be performed in $O(\log n)$ time.*

Proof. Given two generators $\alpha_1=(A_1,a_1)$ and $\alpha_2=(A_2,a_2)$ below a horizontal line ℓ, the goal of the operation is to compute the intersection between the bisector $B(\alpha_1,\alpha_2)$ and ℓ. By Lemma 3.7, $B(\alpha_1,\alpha_2)\cap\ell$ is either empty or a single point.

We apply a prune-and-search technique from Kirkpatrick and Snoeyink [33]. To avoid the lengthy background explanation, we follow the notation in their work [33] without definition. We will rely on Theorem 3.6 in their work [33] as well. To do so, we need to define a decreasing function f and an increasing function g.

We first compute the intersection of ℓ and the reachable region $R(\alpha_1)$ of α_1, as defined in the proof of Lemma 3.7. This can be easily done by computing the intersections between ℓ and the boundary of $R(\alpha_1)$ in $O(\log n)$ time. Let I_1 be their intersection, which is an interval of ℓ as proved in Lemma 3.7. Similarly, we compute the intersection I_2 of ℓ and the reachable region $R(\alpha_2)$ of α_2. Let $I=I_1\cap I_2$. For each endpoint of I, we compute its tangent point on A_1 (along its designated direction) to determine the portion of A_1 whose tangent rays intersect I and let A denote the portion. Similarly, we determine the portion B of A_2 whose tangent rays intersect I. All the preceding takes $O(\log n)$ time.

We parameterize over $[0,1]$ each of the two convex chains A and B in clockwise order—that is, each value of $[0,1]$ corresponds to a slope of a tangent at a point on A (respectively, B). For each point a of A, we define $f(a)$ to be the parameter of the point $b\in B$ such that the tangent ray of A at a along the designated direction and the tangent ray of B at b along the designated direction intersect at a point on the bisector $B(\alpha_1,\alpha_2)$ (Figure 17, left); if the tangent ray at a does not intersect $B(\alpha_1,\alpha_2)$, then define $f(a)=1$. For each point b of B, we define $g(b)$ to be the parameter of the point $a\in A$ such that the tangent ray of A at a and the tangent ray of B at b intersect at a point

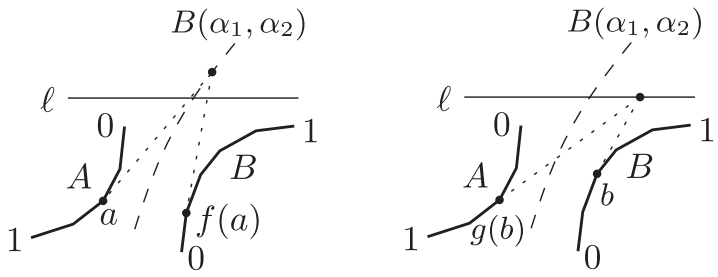

Fig. 17. Illustrating the definitions of $f(a)$ and $g(b)$.

on the line ℓ (e.g., see Figure 17, right). One can verify that f is a continuous decreasing function while g is a continuous increasing function (the tangent at an obstacle vertex of A and B is not unique but the issue can be handled [33]). The fixed-point of the composition of the two functions $g \cdot f$ corresponds to the intersection of ℓ and $B(\alpha_1, \alpha_2)$, which can be computed by applying the prune-and-search algorithm of Theorem 3.6 [33].

As both chains A and B are represented by arrays (of size $O(n)$), to show that the algorithm can be implemented in $O(\log n)$ time, it suffices to show that given any $a \in A$ and any $b \in B$, we can determine whether $f(a) \geq b$ in $O(1)$ time and determine whether $g(b) \geq a$ in $O(1)$ time.

To determine whether $f(a) \geq b$, we do the following. We first find the intersection q of the tangent ray of A at a and the tangent ray of B at b. Then, $f(a) \geq b$ if and only if $d(\alpha_1, q) \leq d(\alpha_2, q)$. Notice that $d(\alpha_1, q) = w(a_1) + |\widehat{a_1 a}| + |\overline{aq}|$, where $|\widehat{a_1 a}|$ is the length of the portion of A_1 between a_1 and a. Recall that we have a data structure on each elementary chain C such that given any two vertices on C, the length of the subchain of C between the two vertices can be computed in $O(1)$ time. Using the data structure, $|\widehat{a_1 a}|$ can be computed in constant time. Since $w(a_1)$ is already known, $d(\alpha_1, q)$ can be computed in constant time. So is $d(\alpha_2, q)$. In the case where the two tangent rays do not intersect, either the tangent ray of A at a intersects the backward extension of the tangent ray of B at b or the tangent ray of B at b intersects the backward extension of the tangent ray of A at a. In the former case, $f(a) \leq b$ holds, whereas in the latter case, $f(a) \geq b$ holds. Hence, whether $f(a) \geq b$ can be determined in constant time.

To determine whether $g(b) \geq a$, we do the following. Find the intersection p_a between ℓ and the tangent ray of A at a and the intersection p_b between ℓ and the tangent ray of B at b. If p_a is to the left of p_b, then $g(b) \geq a$; otherwise, $g(b) \leq a$. Note that by the definition of A, the tangent ray at any point of A intersects I; the same is true for B. Hence, whether $g(b) \geq a$ can be determined in constant time.

The preceding algorithm returns a point q in $O(\log n)$ time. If the intersection of ℓ and $B(\alpha_1, \alpha_2)$ exists, then q is the intersection. Because we do not know whether the intersection exists, we finally validate q by computing $d(\alpha_1, q)$ and $d(\alpha_2, q)$ in $O(\log n)$ time as well as checking whether $q \in \ell$. The point q is valid if and only if $d(\alpha_1, q) = d(\alpha_2, q)$ and $q \in \ell$. □

3.5 The Wavefront Propagation Procedure

In this section, we discuss the wavefront propagation procedure, which is to compute the wavefront $W(e, g)$ for all transparent edges $g \in output(e)$ based on $W(e)$. Consider a transparent edge $g \in output(e)$. The wavefront $W(e, g)$ refers to the portion of $W(e)$ that arrives at g through the well-covering region $\mathcal{U}(g)$ of g if $e \in input(g)$ and through $\mathcal{U}(e)$ otherwise (in the latter case, $g \in input(e)$). We will need to handle the bisector events—that is, the intersections between bisectors and the intersections between bisectors and obstacle edges. The HS algorithm processes the bisector events in *temporal* order—that is, in order of the simulation time τ. The HSY algorithm

instead proposes a simpler approach that processes the events in *spatial order*—that is, in order of their geometric locations. We will adapt the HSY's spatial-order method.

Recall that each wavefront $W(e)$ is represented by a list of generators, which are maintained in the leaves of a fully persistent balanced binary search tree $T(e)$. We further assign each generator a "next bisector event," which is the intersection of its two bounding bisectors (it is set to null if the two bisectors do not intersect). More specifically, for each bisector α, we assign it the intersection of the two bisectors $B(\alpha_l, \alpha)$ and $B(\alpha, \alpha_r)$, where α_l and α_r are α's left and right neighboring generators in $W(e)$, respectively; we store the intersection at the leaf for α. Our algorithm maintains a variant that the next bisector event for each generator in $W(e)$ has already been computed and stored in $T(e)$. We further endow the tree $T(e)$ with additional node fields so that each internal node stores a value that is equal to the minimum (respectively, maximum) x-coordinate (respectively, y-coordinate) among all bisector events stored at the leaves of the subtree rooted at the node. Using these extra values, we can find from a query range of generators the generator whose bisector event has the minimum/maximum x- or y-coordinate in logarithmic time.

The propagation from $W(e)$ to g through $\mathcal{U}$ is done cell by cell, where $\mathcal{U}$ is either $\mathcal{U}(e)$ or $\mathcal{U}(g)$. We start propagating $W(e)$ to the adjacent cell c of e in $\mathcal{U}$ to compute the wavefront passing through all edges of c. Then by using the computed wavefronts on the edges of c, we iteratively run the algorithm on cells of $\mathcal{U}$ adjacent to c. As $\mathcal{U}$ has $O(1)$ cells, the propagation passes through $O(1)$ cells. Hence, the essential ingredient of the algorithm is to propagate a single wavefront, say, $W(e)$, across a single cell c with e on its boundary. Depending on whether c is an empty rectangle, there are two cases.

3.5.1 c Is an Empty Rectangle. We first consider the case where c is an empty rectangle—that is, there is no island inside c and c does not intersect any obstacle. Without loss of generality, we assume that e is an edge on the bottom side of c, and thus all generators of $W(e)$ are below e. Our goal is to compute $W(e, g)$ (i.e., the generators of $W(e)$ claiming g) for all other edges g of c. Our algorithm is similar to the HSY algorithm in the high level, but the low-level implementations are quite different. The main difference is that each bisector in the HSY algorithm is of constant size, although this is not the case in our problem. Due to this, it takes constant time to compute the intersection of two bisectors in the HSY algorithm, whereas in our problem, this costs $O(\log n)$ time.

The technical crux of the algorithm is to process the intersections in c among the bisectors of generators of $W(e)$. Since all generators of $W(e)$ are below e, their bisectors in c are y-monotone by Corollary 3.8. This is a critical property our algorithm relies on. Due to the property, we only need to compute $W(e, g)$ for all edges g on the left, right, and top sides of c. Another helpful property is that since we propagate $W(e)$ through e inside c, if a generator of α of $W(e)$ claims a point $q \in c$, then the tangent from q to α must cross e; we refer to this as the *propagation property*. Due to this property, the points of c claimed by α must be to the right of the tangent ray from the left endpoint of e to α (the direction of the ray is from the tangent point to the left endpoint of e), as well as to the left of the tangent ray from the right endpoint of e to α (the direction of the ray is from the tangent point to the right endpoint of e). We call the former ray the *left bounding ray* of α and the latter the *right bounding ray* of α. As such, for the leftmost generator of $W(e)$, we consider its left bounding ray as its left bounding bisector; similarly, for the rightmost generator of $W(e)$, we consider its right bounding ray as its right bounding bisector.

Starting from e, we use a horizontal line segment ℓ to sweep c upward until its top side. At any moment during the algorithm, the algorithm maintains a subset $W(\ell)$ of generators of $W(e)$ for ℓ by a balanced binary search tree $T(\ell)$; initially, $W(\ell) = W(e)$ and $T(\ell) = T(e)$. Let $[x_1, x_2] \times [y_1, y_2]$ denote the coordinates of c. Using the extra fields on the nodes of the tree $T(\ell)$, we compute a

maximal prefix $W_1(\ell)$ (respectively, $W_2(\ell)$) of generators of $W(\ell)$ such that the bisector events assigned to all generators in it have x-coordinates less than x_1 (respectively, larger than x_2). Let $W_m(\ell)$ be the remaining elements of $W(\ell)$. By definition, the first and last generators of $W_m(\ell)$ have their bisector events with x-coordinates in $[x_1, x_2]$. As all bisectors are y-monotone in c, the lowest bisector intersection in c above ℓ must be the "next bisector event" b associated with a generator in $W_m(\ell)$, which can be found in $O(\log n)$ time using the tree $T(\ell)$. We advance ℓ to the y-coordinate of b by removing the generator α associated with the event b. Finally, we recompute the next bisector events for the two neighbors of α in $W(\ell)$. Specifically, let α_l and α_r be the left and right neighboring generators of α in $W(\ell)$, respectively. We need to compute the intersection of the two bounding bisectors of α_l and update the bisector event of α_l to this intersection. Similarly, we need to compute the intersection of the bounding bisectors of α_r and update the bisector event of α_r to this intersection. Lemma 3.11 shows that each of these bisector intersections can be computed in $O(\log n)$ time by a bisector-bisector intersection operation, using the tentative prune-and-search technique of Kirkpatrick and Snoeyink [33]. Note that if α is the leftmost generator, then α_r becomes the leftmost after α is deleted, in which case we compute the left bounding ray of α_r as its left bounding generator. If α is the rightmost generator, the process is symmetric.

Lemma 3.11 (Bisector-Bisector Intersection Operation). *Each bisector-bisector intersection operation can be performed in $O(\log n)$ time.*

Proof. We are given a horizontal line ℓ and three generators α_1, α_2, and α_3 below ℓ such that they claim points on ℓ in this order. The goal is to compute the intersection of the two bisectors $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$ above ℓ, or determine that such an intersection does not exist. Using the tentative prune-and-search technique of Kirkpatrick and Snoeyink [33], we present an $O(\log n)$ time algorithm.

To avoid the lengthy background explanation, we follow the notation in the work of Kirkpatrick and Snoeyink [33] without definition. We will rely on Theorem 3.9 in their work [33]. To this end, we need to define three continuous and decreasing functions f, g, and h. We define them in a way similar to Theorem 4.10 in their work [33] for finding a point equidistant to three convex polygons. Indeed, our problem may be considered as a weighted case of their problem because each point in the underlying chains of the generators has a weight that is equal to its weighted distance from its generator.

Let A, B, and C be the underlying chains of α_1, α_2, and α_3, respectively.

We parameterize over $[0, 1]$ each of the three convex chains A, B, and C from one end to the other in clockwise order—that is, each value of $[0, 1]$ corresponds to a slope of a tangent at a point on the chains. For each point a of A, we define $f(a)$ to be the parameter of the point $b \in B$ such that the tangent ray of A at a (following the designated direction) and the tangent ray of B at b intersect at a point on the bisector $B(\alpha_1, \alpha_2)$ (e.g., see Figure 17, left); if the tangent ray at a does not intersect $B(\alpha_1, \alpha_2)$, then define $f(a) = 1$. In a similar manner, we define $g(b)$ for $b \in B$ with respect to C and define $h(c)$ for $c \in C$ with respect to A. One can verify that all three functions are continuous and decreasing (the tangent at an obstacle vertex of the chains is not unique, but the issue can be handled [33]). The fixed-point of the composition of the three functions $h \cdot g \cdot f$ corresponds to the intersection of $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$, which can be computed by applying the tentative prune-and-search algorithm of Theorem 3.9 [33].

To guarantee that the algorithm can be implemented in $O(\log n)$ time, since each of the chains A, B, and C is represented by an array, we need to show that given any $a \in A$ and any $b \in B$, we can determine whether $f(a) \geq b$ in $O(1)$ time. This can be done in the same way as in the proof of Lemma 3.10. Similarly, given any $b \in B$ and $c \in C$, we can determine whether $g(b) \geq c$ in $O(1)$

time, and given any $c \in C$ and $a \in A$, we can determine whether $h(c) \geq a$ in $O(1)$ time. Therefore, applying Theorem 3.9 [33] can compute the intersection of $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$ in $O(\log n)$ time.

The preceding algorithm is based on the assumption that the intersection of the two bisectors exists. However, we do not know whether that is true or not. To determine it, we finally validate the solution as follows. Let q be the point returned by the algorithm. We compute the distances $d(\alpha_i, q)$ for $i = 1, 2, 3$. The point q is a true bisector intersection if and only if the three distances are equal. Finally, we return q if and only if q is above ℓ. □

The algorithm finishes once ℓ is at the top side of c. At this moment, no bisector events of $W(\ell)$ are in c. Finally, we run the following *wavefront splitting step* to split $W(\ell)$ to obtain $W(e, g)$ for all edges g on the left, right, and top sides of c. Our algorithm relies on the following observation. Let ζ be the union of the left, top, and right sides of c.

Lemma 3.12. *The list of generators of $W(\ell)$ are exactly those in $W(\ell)$ claiming ζ in order.*

Proof. It suffices to show that during the sweeping algorithm whenever a bisector $B(\alpha_1, \alpha_2)$ of two generators α_1 and α_2 intersects ζ, it will never intersect ∂c again. Let q be such an intersection. Let ζ_l, ζ_t, and ζ_r be the left, top, and right sides of c, respectively.

If q is on ζ_t, then since both α_1 and α_2 are below e, they are also below ζ_t. By Lemma 3.7, $B(\alpha_1, \alpha_2)$ will not intersect the supporting line of ζ_t again and thus will not intersect ∂c again.

If q is on ζ_l, then we claim that both generators α_1 and α_2 are to the right of the supporting line ℓ_l of ζ_l. Indeed, since both generators claim q, the bounding rays (i.e., the left bounding ray of the leftmost generator of $W(\ell)$ and the right bounding ray of the rightmost generator of $W(\ell)$ during the sweeping algorithm) guarantee the propagation property: the tangents from q to the two generators must cross e. Therefore, both generators must be to the right of ℓ_l. By Lemma 3.7, $B(\alpha_1, \alpha_2)$ will not intersect the supporting line of ζ_l again and thus will not intersect ∂c again.

If q is on ζ_r, the analysis is similar to the preceding second case. □

In light of the preceding lemma, our wavefront splitting step algorithm for computing $W(e, g)$ of all edges $g \in \zeta$ works as follows. Consider an edge $g \in \zeta$. Without loss of generality, we assume that the points of ζ are clockwise around c so that we can talk about their relative order.

Let p_l and p_r be the front and rear endpoints of g, respectively. Let α_l and α_r be the generators of $W(\ell)$ claiming p_l and p_r, respectively. Then all generators of $W(\ell)$ to the left of α_l including α_l form the wavefront for all edges of ζ in the front of g; all generators of $W(\ell)$ to the right of α_r including α_r form the wavefront for all edges of ζ after g; and all generators of $W(\ell)$ between α_l and α_r including α_l and α_r form $W(e, g)$. Hence, once α_l and α_r are known, $W(e, g)$ can be obtained by splitting $W(\ell)$ in $O(\log n)$ time. It remains to compute α_l and α_r. In the following, we only discuss how to compute the generator α_l since α_r can be computed analogously.

Starting from the root v of $T(\ell)$, we determine the intersection q between $B(\alpha_1, \alpha_2)$ and ζ, where α_1 is the rightmost generator in the left subtree of v and α_2 is the leftmost generator of the right subtree of v. If q is in the front of g on ζ, then we proceed to the right subtree of v; otherwise, we proceed to the left subtree of v.

It is easy to see that the runtime of the algorithm is bounded by $O(\eta \cdot \log n)$ time, where η is the time for computing q. In the HSY algorithm, each bisector is of constant size and an oracle is assumed to exist that can compute q in $O(1)$ time. In our problem, since a bisector may not be of constant size, it is not clear how to bound η by $O(1)$. But η can be bounded by $O(\log n)$ using the bisector-line intersection operation in Lemma 3.10. Thus, α_l can be computed in $O(\log^2 n)$ time. However, this is not sufficient for our purpose, as this would lead to an overall $O(n + h \log^2 h)$ time algorithm. We instead use the following *binary search plus bisector tracing* approach.

During the wavefront expansion algorithm, for each pair of neighboring generators $\alpha = (A, a)$ and $\alpha' = (A', a')$ in a wavefront (e.g., $W(e)$), we maintain a special point $z(\alpha, \alpha')$ on the bisector $B(\alpha, \alpha')$. For example, in the preceding sweeping algorithm, whenever a generator α is deleted from $W(\ell)$ at a bisector event $b = B(\alpha_l, \alpha) \cap B(\alpha, \alpha_r)$, its two neighbors α_l and α_r now become neighboring in $W(\ell)$. Then, we initialize $z(\alpha_l, \alpha_r)$ to b (the tangent points from b to α_l and α_r are also associated with b). During the algorithm, the point $z(\alpha_l, \alpha_r)$ will move on $B(\alpha, \alpha')$ further away from the two defining generators α and α' and the movement will trace out the hyperbolic-arcs of the bisector. We call $z(\alpha, \alpha')$ the *tracing-point* of $B(\alpha, \alpha')$. Our algorithm maintains a variant that the tracing-point of each bisector of $W(\ell)$ is below the sweeping line ℓ (initially, the tracing-point of each bisector of $W(e)$ is below e).

With the help of the preceding z-points, we compute the generator α_l as follows. Like the preceding algorithm, starting from the root v of $T(\ell)$, let α_1 and α_2 be the two generators as defined previously. To compute the intersection q between $B(\alpha_1, \alpha_2)$ and ζ, we trace out the bisector $B(\alpha_1, \alpha_2)$ by moving its tracing-point $z(\alpha_1, \alpha_2)$ upward (each time trace out a hyperbolic-arc of $B(\alpha_1, \alpha_2)$) until the current tracing hyperbolic-arc intersects ζ at q. If q is in the front of e on ζ, then we proceed to the right subtree of v; otherwise, we proceed to the left subtree of v.

After $W(e, g)$ is obtained, we compute $W(e, g')$ for other edges g' on ζ using the same algorithm as earlier. For the time analysis, observe that each bisector hyperbolic-arc will be traced out at most once in the wavefront splitting step for all edges of ζ because the tracing-point of each bisector will never move "backward."

This finishes the algorithm for propagating $W(e)$ through the cell c. Except for the final wavefront splitting step, the algorithm runs in $O((1+h_c)\log n)$ time, where h_c is the number of bisector events in c. Because c has $O(1)$ edges, the wavefront splitting step takes $O(\log n + n_c)$ time, where n_c is the number of hyperbolic-arcs of bisectors that are traced out.

3.5.2 c Is Not an Empty Rectangle. We now discuss the case in which the cell c is not an empty rectangle. In this case, c has a square hole inside or/and the boundary of c contains obstacle edges. Without loss of generality, we assume that e is on the bottom side of c.

If c contains a square hole, then we partition c into four subcells by cutting c with two lines parallel to e, each passing through an edge of the hole. If c has obstacle edges on its boundary, recall that these obstacles edges belong to $O(1)$ convex chains (each of which is a fragment of an elementary chain); we further partition c by additional edges parallel to e so that each resulting subcell contains at most two convex chains, one on the left side and the other on the right side. Since ∂c has $O(1)$ convex chains, $O(1)$ additional edges are sufficient to partition c into $O(1)$ subcells as earlier. Then, we propagate e through the subcells of c, one by one. In the following, we describe the algorithm for one such subcell. By slightly abusing the notation, we still use c to denote the subcell with e on its bottom side.

Since ∂c has obstacle edges, the propagation algorithm becomes more complicated. As in the HSY algorithm, comparing with the algorithm for the previous case, there are two new bisector events:

- First, a bisector may intersect a convex chain (and thus intersect an obstacle). The HSY algorithm does not explicitly compute these bisector events because such an oracle is not assumed to exist. In our algorithm, however, because the obstacles in our problem are polygonal, we can explicitly determine these events without any special assumption. This is also a reason that the high-level idea of our algorithm is simpler than the HSY algorithm.
- Second, new generators may be created at the convex chains. We still sweep a horizontal line ℓ from e upward. Let $W(\ell)$ be the current wavefront at some moment during the algorithm. Consider two neighboring generators α_1 and α_2 in $W(\ell)$ with α_1 on the left of α_2. We use

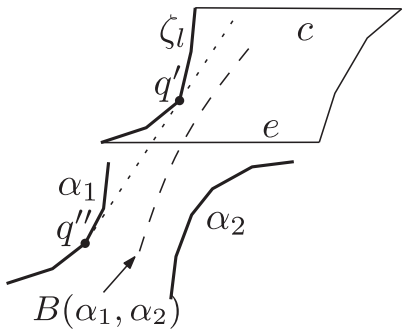

Fig. 18. Illustrating the creation of a new generator at q'.

ζ_l to denote the convex chain on the left side of c. Let q' be the tangent point on ζ_l of the common tangent between ζ_l and α_1, and let q'' be the tangent point on α_1 (Figure 18). If $d(\alpha_1, q') < d(\alpha_2, q')$, then a new generator α on ζ_l with initial vertex q' and weight equal to $d(\alpha_1, q')$ is created (designated counterclockwise direction) and inserted into $W(\ell)$ right before α_1. The bisector $B(\alpha, \alpha_1)$ is the ray emanating from q' and extending away from q''. The region to the left of the ray has α as its predecessor. When the sweeping line ℓ is at q', all wavelets in $W(\ell)$ to the left of α_1 have already collided with ζ_l and thus the first three generators of $W(\ell)$ are α, α_1, and α_2.

In what follows, we describe our sweeping algorithm to propagate $W(e)$ through c. We begin with an easier case where only the left side of c is a convex chain, denoted by ζ_l (and the right side is a vertical transparent edge, denoted by ζ_r). We use ζ_t to denote the top side of c, which is a transparent edge. As in the previous case, we sweep a line ℓ from e upward until the top side ζ_t. During the algorithm, we maintain a list $W(\ell)$ of generators by a balanced binary search tree $T(\ell)$. Initially, $W(\ell) = W(e)$ and $T(\ell) = T(e)$.

We compute the intersection q of the convex chain ζ_l and the bisector $B(\alpha_1, \alpha_2)$, for the leftmost bisectors of α_1 and α_2 of $W(\ell)$. We call it the *bisector-chain intersection operation*. The following lemma shows that this operation can be performed in $O(\log n)$ time, by using the tentative prune-and-search technique of Kirkpatrick and Snoeyink [33].

Lemma 3.13 (Bisector-Chain Intersection Operation). *Each bisector-chain intersection operation can be performed in $O(\log n)$ time.*

Proof. We are given a convex chain ζ_l above a horizontal line ℓ and two generators $\alpha_1 = (A_1, a_1)$ and $\alpha_2 = (A_2, a_2)$ below ℓ such that they claim points on ℓ in this order. The goal is to compute the intersection of $B(\alpha_1, \alpha_2)$ and ζ_l, or determine that they do not intersect. In the following, using the tentative prune-and-search technique of Kirkpatrick and Snoeyink [33], we present an $O(\log n)$ time algorithm.

To avoid the lengthy background explanation, we follow the notation in the work of Kirkpatrick and Snoeyink [33] without definition. We will rely on Theorem 3.9 in their work [33]. To this end, we need to define three continuous and decreasing functions f, g, and h.

Suppose q is the intersection of $B(\alpha_1, \alpha_2)$ and ζ_l. Let p_1 and p_2 be the tangent points from q to A_1 and A_2, respectively. Then, $\overline{qp_1}$ (respectively, $\overline{qp_2}$) does not intersect ζ_l other than q. We determine the portion C of ζ_l such that for each point $p \in C$, its tangent to A_1 does not intersect ζ_l other than p. Hence, $q \in C$. C can be determined by computing the common tangents between ζ_l and A_1, which can be done in $O(\log n)$ time [21, 41]. In addition, we determine the portion B of A_2

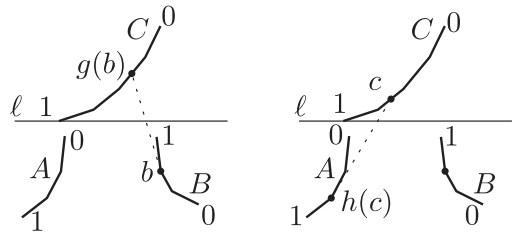

Fig. 19. Illustrating the definitions of $g(b)$ and $h(c)$.

such that the tangent ray at any point of B must intersect C. This can be done by computing the common tangents between C and A_2 in $O(\log n)$ time [21, 41]. Let $A = A_1$.

We parameterize over $[0, 1]$ each of the three convex chains A, B, and C from one end to the other in clockwise order—that is, each value of $[0, 1]$ corresponds to a slope of a tangent at a point on the chains A and B, whereas each value of $[0, 1]$ corresponds to a point of C. For each point a of A, we define $f(a)$ to be the parameter of the point $b \in B$ such that the tangent ray of A at a (following the designated direction of α_1) and the tangent ray of B at b intersect at a point on the bisector $B(\alpha_1, \alpha_2)$ (e.g., see Figure 17, left); if the tangent ray at a does not intersect $B(\alpha_1, \alpha_2)$, then define $f(a) = 1$. For each point b of B, define $g(b)$ to be the parameter of the point $c \in C$ such that $\overline{cb}$ is tangent to B at b (Figure 19, left); note that by the definition of B, the tangent ray from any point of B must intersect C and thus such a point $c \in C$ must exist. For each point $c \in C$, define $h(c)$ to be the parameter of the point of $a \in A$ such that $\overline{ac}$ is tangent to A at a (e.g., see Figure 19, right); note that by the definition of C, such a point $a \in A$ must exist and $\overline{ac}$ does not intersect C other than c. One can verify that all three functions are continuous and decreasing (the tangent at an obstacle vertex of the chains is not unique, but the issue can be handled [33]). The fixed-point of the composition of the three functions $h \cdot g \cdot f$ corresponds to the intersection q of $B(\alpha_1, \alpha_2)$ and ζ_l, which can be computed by applying the tentative prune-and-search algorithm of Theorem 3.9 [33].

To make sure that the algorithm can be implemented in $O(\log n)$ time, since each convex chain is part of an elementary chain and thus is represented by an array, it suffices to show the following: (1) given any $a \in A$ and any $b \in B$, whether $f(a) \geq b$ can be determined in $O(1)$ time; (2) given any $b \in B$ and any $c \in C$, whether $g(b) \geq c$ can be determined in $O(1)$ time; and (3) given any $c \in C$ and any $a \in A$, whether $h(c) \geq a$ can be determined in $O(1)$ time. We prove them in the following.

Indeed, for (1), it can be done in the same way as in the proof of Lemma 3.10. For (2), $g(b) \geq c$ if and only if c is to the right of the tangent ray of B at b, which can be easily determined in $O(1)$ time. For (3), $h(c) \geq a$ if and only c is to the right of the tangent ray of A at a, which can be easily determined in $O(1)$ time.

Therefore, applying the tentative prune-and-search technique in Theorem 3.9 [33] can compute q in $O(\log n)$ time. Note that the preceding algorithm is based on the assumption that the intersection of $B(\alpha_1, \alpha_2)$ and ζ_l exists. However, we do not know whether this is true or not. To determine that, we finally validate the solution as follows. Let q be the point returned by the algorithm. We first determine whether $q \in \zeta_l$. If not, then the intersection does not exist. Otherwise, we further compute the two distances $d(\alpha_i, q)$ for $i = 1, 2$ in $O(\log n)$ time. If the distances are equal, then q is the true intersection; otherwise, the intersection does not exist. □

If the intersection q of ζ_l and $B(\alpha_1, \alpha_2)$ does not exist, then we compute the tangent between ζ_l and α_1, which can be done in $O(\log n)$ time [41]; let q' be the tangent point at ζ_l. Regardless of

whether q exists or not, we compute the lowest bisector intersection b in c above ℓ in the same way as in the algorithm for the previous case where c is an empty rectangle. Depending on whether q exists or not, we proceed as follows. For any point p in the plane, let $y(p)$ denote the y-coordinate of p:

(1) If q exists, then depending on whether $y(q) \leq y(b)$, there are two subcases. If $y(q) \leq y(b)$, then we process the bisector event q: remove α_1 from $W(\ell)$ and then recompute q, q', and b. Otherwise, we process the bisector event at b in the same way as in the previous case and then recompute q, q', and b.
(2) If q does not exist, then depending on whether $y(b) \leq y(q')$, there are two subcases. If $y(b) \leq y(q')$, then we process the bisector event at b in the same way as before and then recompute q, q', and b. Otherwise, we insert a new generator $\alpha = (A, q')$ to $W(\ell)$ as the leftmost generator, where A is the fragment of the elementary chain containing ζ_l from q' counterclockwise to the end of the chain, and α is designated the counterclockwise direction of A and the weight of q' is $d(\alpha_1, q')$ (e.g., see Figure 18). The ray from q' in the direction from q'' to q' is the bisector of α and α_1, where q'' is the tangent point on α_1 of the common tangent between α_1 and ζ_l. We initialize the tracing-point $z(\alpha, \alpha_1)$ of $B(\alpha, \alpha_1)$ to q'. Finally, we recompute q, q', and b.

Once the sweep line ℓ reaches the top side ζ_t of c, the algorithm stops. Finally, as in the previous case, we run a wavefront splitting step. Because the left side ζ_l consists of obstacle edges, we split $W(\ell)$ to compute $W(e, g)$ for all transparent edges g on the top side ζ_t and the right side ζ_r of c. The algorithm is the same as the previous case.

The preceding discusses the case where only the left side ζ_l of c is a convex chain. For the general case where both the left and right sides of c are convex chains, the algorithm is similar. The difference is that we have to compute a point p corresponding to q and a point p' corresponding to q' on the right side ζ_r of c. More specifically, p is the intersection of $B(\alpha_2', \alpha_1')$ with ζ_r, where α_2' and α_1' are the two rightmost generators of W. If p does not exist, then we compute the common tangent between ζ_r and α_1', and p' is the tangent point on ζ_r.

In the following, if q does not exist, we let $y(q)$ be ∞; otherwise, q' is not needed and we let $y(q')$ be ∞. We apply the same convention to p and p'. We define b as the bisector event in the same way as before. In each step, we process the lowest point r of $\{q, q', b, p, p'\}$. If r is q or p, we process it in the same way as before for q. If r is q' or p', we process it in the same way as before for q'. If r is b, we process it in the same way as before. After processing r, we recompute the five points. Each step takes $O(\log n)$ time. After the sweep line ℓ reaches the top side ζ_t of c, $W(\ell)$ is $W(e, \zeta_t)$ for the top side ζ_t of c because both the left and right sides of c are obstacle edges. Finally, we run the wavefront splitting step on $W(\ell)$ to compute the wavefronts $W(e, g)$ for all transparent edges g on ζ_t.

In summary, propagating $W(e)$ through c takes $O((1 + h_c) \cdot \log n + n_c)$ time, where h_c is the number of bisector events (including both the bisector-bisector intersection events and the bisector-obstacle intersection events) and n_c is the number of hyperbolic-arcs of bisectors that are traced out in the wavefront splitting step.

We use the following lemma to summarize the algorithm for both cases (i.e., regardless of whether c is an empty rectangle or not).

Lemma 3.14. *Suppose $W(e)$ is a wavefront on a transparent edge of a cell c of the subdivision $\mathcal{S}'$. Then, $W(e)$ can be propagated through c to all other transparent edges of c in $O((1 + h_c) \log n + n_c)$ time, where h_c is the number of bisector events (including both the bisector-bisector intersection events and bisector-obstacle intersection events) and n_c is the number of hyperbolic-arcs of bisectors that are traced out in the wavefront splitting step.*

3.6 Time Analysis

In this section, we show that the running time of our wavefront expansion algorithm described previously is bounded by $O(n + h \log h)$. For this and also for the purpose of constructing the shortest path map $SPM'(s)$ later in Section 3.8, as in the HS algorithm, we mark generators in the way that if a generator α is involved in a true bisector event of $SPM(s)$ (either a bisector-bisector intersection or a bisector-obstacle intersection) in a cell c of the subdivision $\mathcal{S}'$, then α is guaranteed to be in a set of marked generators for c (but a marked generator for c may not actually participate in a true bisector event in c). The generator marking rules are presented next, which are consistent with those in the HS algorithm:

Generator marking rules:

(1) For any generator $\alpha = (A, a)$, if its initial vertex a lies in a cell c, then mark α in c.
(2) Let e be a transparent edge and let $W(e)$ be a wavefront coming from some generator α's side of e.
   (a) If α claims an endpoint b of e in $W(e)$, or if it would do so except for an artificial wavefront, then mark α in all cells c incident to b.
   (b) If α's claim in $W(e)$ is shortened or eliminated by an artificial wavelet, then mark α for c, where c is the cell having e as an edge and on α's side of e.
(3) Let e and g be two transparent edges with $g \in output(e)$. Mark a generator α of $W(e)$ in both cells having e as an edge if one of the following cases happens:
   (a) α claims an endpoint of g in $W(e, g)$.
   (b) α participates in a bisector event either during the wavefront propagation procedure for computing $W(e, g)$ from $W(e)$, or during the wavefront merging procedure for computing $W(g)$. Note that α is also considered to participate in a bisector event if its claim on g is shortened or eliminated by an artificial wavelet.
(4) If a generator α of $W(e)$ claims part of an obstacle edge during the wavefront propagation procedure for propagating $W(e)$ toward $output(e)$ (this includes the case in which α participates in a bisector-obstacle intersection event), then mark α in both cells having e as an edge.

Note that each generator may be marked multiple times, and each mark applies to one instance of the generator. As such, when we say "the number of marked generators," we mean the number of instances of the marked generators.

Lemma 3.15. *The total number of marked generators during the algorithm is at most $O(h)$.*

Because the proof of Lemma 3.15 is quite technical and lengthy, we devote the entirety of Section 3.7 to it. In the rest of this section, we use Lemma 3.15 to show that the running time of our wavefront expansion algorithm is bounded by $O(n + h \log h)$. Our goal is to prove the following lemma.

Lemma 3.16. *The wavefront expansion algorithm runs in $O(n + h \log h)$ time and space.*

First of all, by Lemma 3.1, constructing the conforming subdivision $\mathcal{S}'$ can be done in $O(n + h \log h)$ time and $O(n)$ space.

The wavefront expansion algorithm has two main subroutines: the wavefront merging procedure and the wavefront propagation procedure.

The wavefront merging procedure is to construct $W(e)$ based on $W(f, e)$ for the edges $f \in input(e)$. By Lemma 3.9, this step takes $O((1+k) \log n)$ time, where k is the total number of generators in all wavefronts $W(f, e)$ that are absent from $W(e)$. According to the algorithm, if a generator α is absent from $W(e)$, it must be deleted at a bisector event. Thus, α must be marked by Rule 3(b).

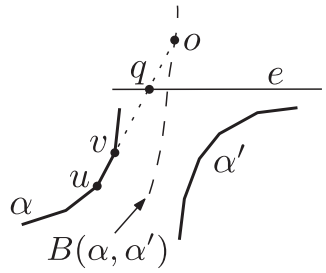

Fig. 20. Illustrating the definitions of u, v, q, and o.

Due to Lemma 3.15, the total sum of k in the entire algorithm is $O(h)$. As such, the wavefront merging procedure in the entire algorithm takes $O(h \log n)$ time in total.

The wavefront propagation procedure is to compute $W(e, g)$ by propagating $W(e)$ to all edges $g \in output(e)$ either through $\mathcal{U}(g)$ or $\mathcal{U}(e)$. By Lemma 3.14, the running time of the procedure is $O((1 + h_c) \log n + n_c)$ time, where h_c is the number of bisector events (including both the bisector-bisector intersection events and bisector-obstacle intersection events) and n_c is the number of hyperbolic-arcs of bisectors that are traced out in the wavefront splitting step. For each bisector-bisector intersection event, at least one involved generator is marked by Rule 3(b). For each bisector-obstacle intersection event, at least one involved generator is marked by Rule 4. Hence, by Lemma 3.15, the total sum of h_c in the entire algorithm is $O(h)$. In addition, Lemma 3.17 shows that the total sum of n_c in the entire algorithm is $O(n)$. Therefore, the wavefront propagation procedure in the entire algorithm takes $O(n + h \log n)$ time in total.

Lemma 3.17. *The total number of traced hyperbolic-arcs of the bisectors in the entire algorithm is* $O(n)$.

Proof. First of all, notice that each extension bisector consists of a single hyperbolic-arc, which is on a straight line. As each generator is marked by Rule 1, by Lemma 3.15, the total number of generators created in the algorithm is $O(h)$. Since each generator can define at most one extension bisector, the number of hyperbolic-arcs on extension bisectors is at most $O(h)$. In the following, we focus on hyperbolic-arcs of non-extension bisectors. Instead of counting the number of traced hyperbolic-arcs, we will count the number of their endpoints.

Consider a hyperbolic-arc endpoint o that is traced out. According to our algorithm, o is traced out during the wavefront propagation procedure for propagating $W(e)$ to compute $W(e, g)$ for some transparent edge e and $g \in output(e)$. Suppose o belongs to a non-extension bisector $B(\alpha, \alpha')$ of two generators α and α' in $W(e)$. Then, o must be defined by an obstacle edge $\overline{uv}$ of either α or α'—that is, the ray $\rho(u, v)$ emanating from v along the direction from u to v (which is consistent with the designated direction of the generator that contains $\overline{uv}$) hits $B(\alpha, \alpha')$ at o (Figure 20). Without loss of generality, we assume that $\overline{uv}$ belongs to α. In the following, we argue that $\overline{uv}$ can define $O(1)$ hyperbolic-arc endpoints that are traced out during the entire algorithm (a hyperbolic-arc endpoint defined by $\overline{uv}$ is counted twice as it is traced out twice), which will prove Lemma 3.17, as there are $O(n)$ obstacle edges in total.

We first discuss some properties. Since $o \in B(\alpha, \alpha')$, both α and α' claim o. As both generators are in $W(e)$, the ray $\rho(u, v)$ must cross e, say, at a point q (e.g., see Figure 20). Because o is traced out when we propagate $W(e)$ to compute $W(e, g)$, $\overline{oq}$ must be in either $\mathcal{U}(g)$ or $\mathcal{U}(e)$ (i.e., $\overline{oq} \subseteq \mathcal{U}(e) \cup \mathcal{U}(g)$). We call (e, g) the *defining pair* of o for $\overline{uv}$. According to our algorithm, during the propagation from $W(e)$ to g, $\overline{uv}$ defines only one hyperbolic-arc endpoint, because it is uniquely determined by the wavefront $W(e)$. As such, to prove that $\overline{uv}$ can define $O(1)$ hyperbolic-arc endpoints that

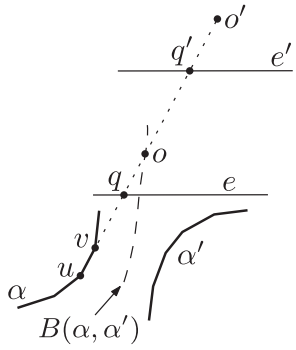

Fig. 21. Illustrating the definitions of u, v, q, and o.

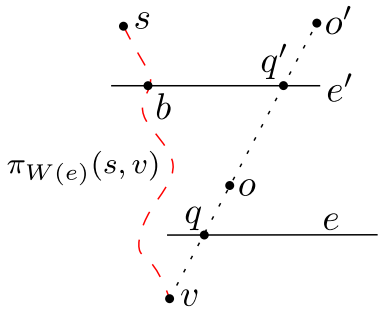

Fig. 22. The dashed (red) path is $\pi_{W(e)}(s, v)$, which crosses e' at b.

are traced out during the entire algorithm, it suffices to show that there are at most $O(1)$ defining pairs for $\overline{uv}$. Let Π denote the set of all such defining pairs. We prove $|\Pi| = O(1)$ in the following.

For each pair $(e', g') \in \Pi$, according to the preceding discussion, $\overline{uv}$ and (e', g') define a hyperbolic-arc endpoint o' such that o' is on the ray $\rho(u, v)$ and $\overline{vo'}$ crosses e' at a point q'. Without loss of generality, we assume that (e, g) is a pair that minimizes the length $|\overline{vq'}|$. Let $W(e')$ refer to the wavefront at e' whose propagation to g' traces out o'.

We partition Π into two subsets Π_1 and Π_2, where Π_1 consists of all pairs (e', g') of Π such that $\overline{qo}$ intersects e' and $\Pi_2 = \Pi \setminus \Pi_1$. Since $\overline{oq} \subseteq \mathcal{U}(e) \cup \mathcal{U}(g)$, each well-covering region contains $O(1)$ cells, and $|output(e')| = O(1)$ for each transparent edge e', the size of Π_1 is $O(1)$.

For Π_2, we further partition it into two subsets Π_{21} and Π_{22}, where Π_{21} consists of all pairs (e', g') of Π_2 such that e is in the well-covering region $\mathcal{U}(e')$ of e' or $e' \in \mathcal{U}(e) \cup \mathcal{U}(g)$, and $\Pi_{22} = \Pi_2 \setminus \Pi_{21}$. Since e is in $\mathcal{U}(e')$ for a constant number of transparent edges e', each of $\mathcal{U}(e)$ and $\mathcal{U}(g)$ contains $O(1)$ cells, and $|output(e')| = O(1)$ for each e', it holds that $|\Pi_{21}| = O(1)$. In the following, we argue that $\Pi_{22} = \emptyset$, which will prove $|\Pi| = O(1)$.

Assume to the contrary that $|\Pi_{22}| \neq \emptyset$ and let (e', g') be a pair of Π_{22}. Since $(e', g') \in \Pi_2$, by the definition of Π_2, e' does not intersect $\overline{oq}$. By the definition of e, $\overline{vq} \setminus \{q\}$ does not intersect e'. Recall that q' is the intersection of e' and $\rho(u, v)$. Therefore, the points v, q, o, q', and o' appear on the ray $\rho(u, v)$ in this order (Figure 21). Further, since $(e', g') \in \Pi_{22}$, by the definition of Π_{22}, e is not in $\mathcal{U}(e')$ and e' is not in $\mathcal{U}(e) \cup \mathcal{U}(g)$.

Without loss of generality, we assume that e' is horizontal and the wavefront $W(e')$ is from below e' (thus, v is below e' while o' is above e'). Let $\mathcal{F}'$ be the modified free space by replacing e' with an opaque edge of open endpoints. Since the generator in $W(e')$ that contains v claims q', $\pi'(s, q')$ is a shortest path from s to q' in $\mathcal{F}'$, where $\pi'(s, q')$ is the path following the wavefront $W(e')$. Since v is in the generator of $W(e')$ that claims q', v is the anchor of q' in $\pi'(s, q')$—that is, the edge of the path incident to q' is $\overline{vq'}$. Let $\pi'(s, v)$ be the subpath of $\pi'(s, q')$ between s' and v. Then, $\pi'(s, v)$ is a shortest path from s to v in $\mathcal{F}'$.

Recall that v is in the generator α of $W(e)$. Let $\pi_{W(e)}(s, v)$ be the path from s to v following $W(e)$. We claim that $|\pi'(s, v)| = |\pi_{W(e)}(s, v)|$ (Figure 22). Assume to the contrary this is not true. Then, either $|\pi'(s, v)| < |\pi_{W(e)}(s, v)|$ or $|\pi_{W(e)}(s, v)| < |\pi'(s, v)|$:

- If $|\pi'(s, v)| < |\pi_{W(e)}(s, v)|$, then since $\pi_{W(e)}(s, v)$ is a shortest path from s to v in the modified free space by considering e as an opaque edge of open endpoints, $\pi'(s, v)$ must cross the interior of e. This means that $\pi'(s, q') = \pi'(s, v) \cup \overline{vq'}$ crosses e twice. Because $\pi'(s, q')$ is a shortest path in $\mathcal{F}'$ and $e \in \mathcal{F}'$, it cannot cross e twice, a contradiction.

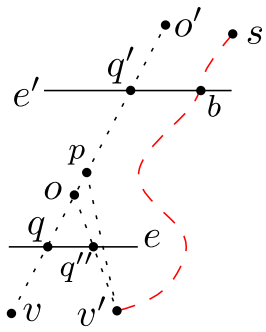

Fig. 23. The dashed (red) path is $\pi_{W(e)}(s, v')$, which crosses e' at b.

- If $|\pi_{W(e)}(s,v)| < |\pi'(s,v)|$, then since $\pi'(s,v)$ is a shortest path from s to v in $\mathcal{F}'$, the path $\pi_{W(e)}(s,v)$ cannot be in $\mathcal{F}'$ and thus must cross the interior of e', say, at a point b (e.g., see Figure 21). Let $\pi_{W(e)}(s,b)$ be the subpath of $\pi_{W(e)}(s,v)$ between s and b. Let $\pi_{W(e)}(b,q')$ be a shortest path from b to q' along e' by considering it as an opaque edge with open endpoints. Note that if b and q' are on different sides of e', then $\pi_{W(e)}(b,q')$ must be through an open endpoint of e'. It is not difficult to see that $|\pi_{W(e)}(b,q')| \leq |e'|$. Let $\pi_{W(e)}(s,q')$ be the concatenation of $\pi_{W(e)}(s,b)$ and $\pi_{W(e)}(b,q')$. Hence, $\pi_{W(e)}(s,q')$ is a path in $\mathcal{F}'$. Since $\pi'(s,q')$ is a shortest path from s to q' in $\mathcal{F}'$, it holds that $|\pi'(s,q')| \leq |\pi_{W(e)}(s,q')|$.

  Notice that $|\pi_{W(e)}(s,q')| = |\pi_{W(e)}(s,b)| + |\pi_{W(e)}(b,q')| \leq |\pi_{W(e)}(s,v)| + |e'| < |\pi'(s,v)| + |e'|$.

  However, $|\pi'(s,q')| = |\pi'(s,v)| + |\overline{vq'}| \geq |\pi'(s,v)| + |\overline{qq'}|$. We claim that $|\overline{qq'}| \geq 2|e'|$. Indeed, since e is outside $\mathcal{U}(e')$, $q \in e$, and $q' \in e'$, $\overline{qq'}$ must cross $\partial\mathcal{U}(e')$ at a point b'. By the property of well-covering regions of $\mathcal{S}'$, $|\overline{b'q'}| \geq 2|e'|$. Since $|\overline{qq'}| \geq |\overline{b'q'}|$, we obtain $|\overline{qq'}| \geq 2|e'|$. In light of the claim, we have $|\pi'(s,q')| \geq |\pi'(s,v)| + 2|e'| > |\pi'(s,v)| + |e'| > |\pi_{W(e)}(s,q')|$. But this incurs contradiction since $|\pi'(s,q')| \leq |\pi_{W(e)}(s,q')|$.

Therefore, $|\pi'(s,v)| = |\pi_{W(e)}(s,v)|$ holds.

As $q' \notin \overline{qo}$, we define p as a point on $\overline{oq'} \setminus \{o\}$ infinitely close to o (Figure 23). Hence, $p \in \overline{vq'}$. Since $\pi'(s,q') = \pi'(s,v) \cup \overline{vq'}$ is a shortest path from s to q' in $\mathcal{F}'$, $\pi'(s,v) \cup \overline{vp}$ is a shortest path from s to p to $\mathcal{F}'$.

Recall that o is on the non-extension bisector $B(\alpha,\alpha')$ of two generators α and α' in $W(e)$ and v is on α. Let v' be the anchor of o in α' (e.g., see Figure 23). Since $|\pi'(s,v)| = |\pi_{W(e)}(s,v)|$, we have $|\pi'(s,v)| + |\overline{vo}| = |\pi_{W(e)}(s,v)| + |\overline{vo}| = |\pi_{W(e)}(s,v')| + |\overline{v'o}|$, where $\pi_{W(e)}(s,v')$ is the path from s to v' following $W(e)$. By the definition of p and because $B(\alpha,\alpha')$ is a non-extension bisector, it holds that $|\pi_{W(e)}(s,v)| + |\overline{vp}| > |\pi_{W(e)}(s,v')| + |\overline{v'p}|$. As $|\pi'(s,v)| = |\pi_{W(e)}(s,v)|$, we have $|\pi'(s,v)| + |\overline{vp}| > |\pi_{W(e)}(s,v')| + |\overline{v'p}|$. Since $\pi'(s,v) \cup \overline{vp}$ is a shortest path from s to p in $\mathcal{F}'$, $\pi_{W(e)}(s,v') \cup \overline{v'p}$ cannot be a path from s to p in $\mathcal{F}'$. Therefore, $\pi_{W(e)}(s,v') \cup \overline{v'p}$ must intersect the interior of e'.

We claim that $\overline{v'p}$ cannot intersect e'. Indeed, since $\overline{v'p}$ is covered by the wavefront $W(e)$ when $W(e)$ propagates to g in either $\mathcal{U}(e)$ or $\mathcal{U}(g)$, $\overline{v'p}$ is in $\mathcal{U}(e) \cup \mathcal{U}(g)$. Since $(e',g') \in \Pi_{22}$ and by the definition of Π_{22}, e' is not in $\mathcal{U}(e) \cup \mathcal{U}(g)$. Therefore, $\overline{v'p}$ cannot intersect e'.

The preceding claim implies that $\pi_{W(e)}(s,v')$ must intersect the interior of e' at a point, say, b (e.g., see Figure 23). Let $\pi_{W(e)}(s,b)$ be the subpath of $\pi_{W(e)}(s,v')$ between s and b. Let $\pi_{W(e)}(b,q')$ be a path from b to q' along e' by considering e' as an opaque edge of open endpoints. As discussed earlier, $|\pi_{W(e)}(b,q')| \leq |e'|$. Hence, $\pi_{W(e)}(s,b) \cup \pi_{W(e)}(b,q')$ is a

path in $\mathcal{F}'$, whose length is at most $|\pi_{W(e)}(s,b)| + |e'| \leq |\pi_{W(e)}(s,v')| + |e'|$. However, $|\pi_{W(e)}(s,v')| + |\overline{v'o}| + |\overline{oq'}| = |\pi'(s,v)| + |\overline{vo}| + |\overline{oq'}| = |\pi'(s,v)| + |\overline{vq'}|$. Since $(e',g') \in \Pi_{22}$ and by the definition of Π_{22}, e is outside $\mathcal{U}(e')$. Because $q' \in e'$ and $\overline{ov'}$ crosses e at a point q'', by the property of well-covering regions of $\mathcal{S}'$, $|\overline{q''o}| + |\overline{oq'}| \geq 2|e'|$. Therefore, we obtain

$$\begin{aligned}
|\pi'(s,v)| + |\overline{vq'}| &= |\pi_{W(e)}(s,v')| + |\overline{v'o}| + |\overline{oq'}| \\
&\geq |\pi_{W(e)}(s,v')| + |\overline{q''o}| + |\overline{oq'}| \\
&\geq |\pi_{W(e)}(s,v')| + 2|e'| \geq |\pi_{W(e)}(s,b)| + 2|e'| \\
&> |\pi_{W(e)}(s,b)| + |e'| \geq |\pi_{W(e)}(s,b)| + |\pi_{W(e)}(b,q')|.
\end{aligned}$$

This means that $\pi_{W(e)}(s,b) \cup \pi_{W(e)}(b,q')$ is a path from s to q' in $\mathcal{F}'$ that is shorter than the path $\pi'(s,v) \cup \overline{vq'}$. But this incurs a contradiction, as $\pi'(s,v) \cup \overline{vq'}$ is a shortest path from s to q' in $\mathcal{F}'$.

This completes the proof of the lemma. □

In summary, the total time of the wavefront expansion algorithm is $O(n + h\log n)$, which is $O(n + h\log h)$.

For the space complexity of the algorithm, since each wavefront $W(e)$ is maintained by a persistent binary tree, each bisector event costs additional $O(\log h)$ space. As discussed earlier, the total sum of $k+h_c$ in the entire algorithm, which is the total number of bisector events, is $O(h)$. Thus, the space cost by persistent binary trees is $O(h\log h)$. The space used by other parts of the algorithm is $O(n)$. Hence, the space complexity of the algorithm is $O(n + h\log h)$. This proves Lemma 3.16.

3.7 Proving Lemma 3.15

In this section, we prove Lemma 3.15. The proof follows the same strategy in the high level as that in the HS algorithm, although many details are different. We start with the following lemma.

Lemma 3.18. *Suppose Π is the set of pairs (e,B) of transparent edges e and bisectors B such that B crosses e in the wavefront $W(e)$ of e during our wavefront expansion algorithm but the same crossing does not occur in SPM$'(s)$. Then, $|\Pi| = O(h)$.*

Proof. Let $\alpha_1 = (A_1, a_1)$ and $\alpha_2 = (A_2, a_2)$ be the two generators of B. Recall that the well-covering region $\mathcal{U}(e)$ of e is the union of $O(1)$ cells of $\mathcal{S}'$ and each cell has $O(1)$ elementary chain fragments on its boundary. Hence, $\mathcal{U}(e)$ has $O(1)$ convex chains on its boundary. Recall also that $W(e)$ is computed by merging all contributing wavefronts $W(f,e)$ for $f \in \mathit{input}(e)$ in the wavefront merging procedure, and $W(f,e)$ is computed by propagating $W(f)$ to e through $\mathcal{U}(e)$ in the wavefront propagation procedure.

We first argue that the number of pairs (e,B) of Π in the case where at least one of α_1 and α_2 is in $\mathcal{U}(e)$ is $O(h)$. Indeed, according to our wavefront propagation procedure, if a subcell $c \in \mathcal{U}(e)$ is an empty rectangle, then no new generators will be created when $W(f)$ is propagating through c; otherwise, although multiple generators may be created, at most two (one on each side of c) are in the wavefront existing c. As $\mathcal{U}(e)$ has $O(1)$ cells and each cell may be partitioned into $O(1)$ subcells during the wavefront propagation procedure, only $O(1)$ generators of $W(e)$ are inside $\mathcal{U}(e)$. Since each generator of $W(e)$ may define at most two bisectors in $W(e)$ and the total number of transparent edges of $\mathcal{S}'$ is $O(h)$, the number of pairs (e,B) of Π such that at least one generator of B is in $\mathcal{U}(e)$ is at most $O(h)$.

In the following, we assume that both α_1 and α_2 are outside $\mathcal{U}(e)$—that is, their initial vertices a_1 and a_2 are outside $\mathcal{U}(e)$. Let q be the intersection of B and e. Let $\pi'(a_1,q)$ denote the path from q to a_1 following the tangent from q to A_1 and then to a_1 along A_1; define $\pi'(a_2,q)$ similarly. Clearly, both α_1 and α_2 claim q in $W(e)$. Since α_1 is outside $\mathcal{U}(e)$, α_1 must be in $W(f_1)$ for some

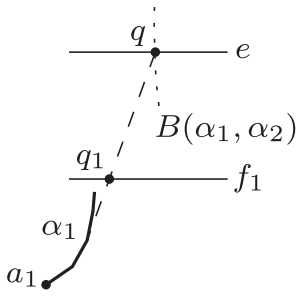

Fig. 24. Illustrating q and q_1.

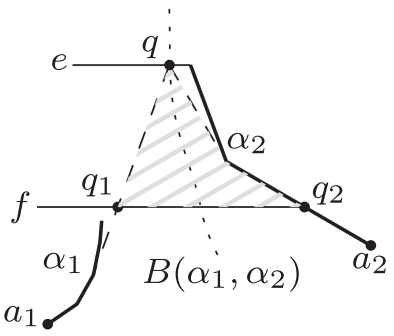

Fig. 25. Illustrating the psuedo-triangle $\triangle(q, q_1, q_2)$ (the shaded region).

transparent edge f_1 of $\partial\mathcal{U}(e)$ so that α_1 of $W(e)$ is from $W(f_1, e)$. Since α_1 is outside $\mathcal{U}(e)$, $\pi'(a_1, q)$ must intersects f_1, say, at point q_1 (Figure 24), and $\pi'(q_1, q)$ is inside $\mathcal{U}(e)$, where $\pi'(q_1, q)$ is the subpath of $\pi'(a_1, q)$ between q_1 and q. In addition, f_1 is processed (for the wavefront propagation procedure) earlier than e because $W(f_1)$ contributes to $W(e)$.

We claim that q_1 is claimed by α_1 in $SPM'(s)$. Assume to the contrary that this is not true. Then, let $\pi(s, q_1)$ be a shortest path from s to q_1 in the free space $\mathcal{F}$:

- If $\pi(s, q_1)$ does not intersect e, then $\pi(s, q_1)$ is in the modified free space $\mathcal{F}'$ by replacing e with an opaque edge of open endpoints. Since α_1 claims q on e, $\pi = \pi_{W(e)}(s, a_1) \cup \pi'(a_1, q)$ is a shortest path from s to q in the modified free space $\mathcal{F}'$, where $\pi_{W(e)}(s, a_1)$ is the path from s to a_1 following the wavefront $W(e)$. Since q_1 is in π and q_1 is not claimed by α_1 in $SPM'(s)$, if we replace the portion between s and q_1 in π by $\pi(s, q_1)$, we obtain a shorter path from s to q in $\mathcal{F}'$ than π. But this incurs a contradiction since π is a shortest path from s to q in $\mathcal{F}'$.
- If $\pi(s, q_1)$ intersects e, say, at a point b, then since $f_1 \in \partial\mathcal{U}(e)$ and thus $d(f_1, e) \geq 2 \cdot \max\{|f_1|, |e|\}$ by the properties of the well-covering regions of $\mathcal{S}'$, we show in the following that e must be processed earlier than f_1, which incurs a contradiction because f_1 is processed earlier than e.

  Indeed, $covertime(e) = \tilde{d}(s, e) + |e| \leq d(s, b) + \frac{1}{2} \cdot |e| + |e| = d(s, b) + \frac{3}{2} \cdot |e|$. However, $covertime(f_1) = \tilde{d}(s, f_1) + |f_1| \geq d(s, q_1) - \frac{1}{2} \cdot |f_1| + |f_1| = d(s, b) + d(b, q_1) + \frac{1}{2} \cdot |f_1|$. Since $f_1 \in \partial\mathcal{U}(e)$, $b \in e$, and $q_1 \in f_1$, $d(b, q_1) \geq d(f_1, e) \geq 2 \cdot |e|$. Hence, we obtain $covertime(f_1) > covertime(e)$ and thus e must be processed earlier than f_1.

Therefore, q_1 must be claimed by α_1 in $SPM(s)$.

We define f_2, q_2, and $\pi'(q, q_2)$ analogously with respect to α_2. Similarly, we can show that q_2 is claimed by α_2 in $SPM'(s)$.

For each $f \in \partial\mathcal{U}(e)$, the generators of $W(f)$ that are also in $W(e)$ may not form a single subsequence of the generator list of $W(e)$, but they must form a constant number of (maximal) subsequences. Indeed, since $\mathcal{U}(e)$ is the union of $O(1)$ cells of $\mathcal{S}'$, the number of islands in $\mathcal{U}(e)$ is $O(1)$. Thus, the number of topologically different paths from f to e in $\mathcal{U}(e)$ is $O(1)$; each such path will introduce at most one subsequence of generators of $W(e)$ that are also from $W(f)$. Therefore, the generators of $W(f)$ that are also in $W(e)$ form $O(1)$ subsequences of the generator list of $W(e)$.

Since $\partial\mathcal{U}(e)$ has $O(1)$ transparent edges, the generator list of $W(e)$ can be partitioned into $O(1)$ subsequences each of which is from $W(f)$ following topologically the same path for a single transparent edge f of $\partial\mathcal{U}(e)$. Due to this property, we have the following observation: the number of pairs of adjacent generators of $W(e)$ that are from different subsequences is $O(1)$.

Due to the preceding observation, the number of pairs of edges f_1 and f_2 on $\partial\mathcal{U}(e)$ in the case where $f_1 \neq f_2$, or $f_1 = f_2$ but $\pi'(q, q_1)$ and $\pi'(q, q_2)$ are topologically different in $\mathcal{U}(e)$ is only $O(1)$. Therefore, the total number of pairs (e, B) of Π in that case is $O(h)$. In the following, it suffices to consider the case where $f_1 = f_2$, and $\pi'(q, q_1)$ and $\pi'(q, q_2)$ are topologically the same in $\mathcal{U}(e)$; for reference purpose, we use Π' to denote the subset of pairs (e, B) of Π with the preceding property. In the following, we prove $|\Pi'| = O(h)$.

Let $f = f_1 = f_2$. Since $\pi'(q, q_1)$ and $\pi'(q, q_2)$ are topologically the same in $\mathcal{U}(e)$, the region bounded by $\pi'(q, q_1)$, $\pi'(q, q_2)$, and $\overline{q_1q_2}$ must be in $\mathcal{U}(e)$; we call the preceding region a *pseudo-triangle*, for both $\pi'(q, q_1)$ and $\pi'(q, q_2)$ are convex chains, and we use $\triangle(q, q_1, q_2)$ to denote it (Figure 25). Because (e, B) is not an incident pair in $SPM'(s)$, the point q must be claimed by a different generator α in $SPM'(s)$, which must be from the side of e different than α_1 and α_2. We proved earlier that q_1 is claimed by α_1 and q_2 is claimed by α_2 in $SPM'(s)$. Hence, there must be at least one bisector event in $SPM'(s)$ that lies in the interior of the pseudo-triangle $\triangle(q, q_1, q_2)$.[5] We charge the early demise of B to any one of these bisector events in $\triangle(p, q_1, q_2)$.

Note that the path $\pi'(q, q_1)$ (respectively, $\pi'(q, q_2)$) is in a shortest path from s to q following the wavefront $W(e)$ in the modified free space by replacing e with an opaque edge of open endpoints. Hence, if Π' has other pairs (e, B') whose first element is e, then the corresponding paths $\pi'(q, q_1)$ and $\pi'(q, q_2)$ of all these pairs are disjoint, and thus the corresponding pseudo-triangles $\triangle(q, q_1, q_2)$ are also disjoint in $\mathcal{U}(e)$. Hence, each bisector event of $SPM'(s)$ inside $\triangle(q, q_1, q_2)$ is charged at most once for all pairs of Π' that have e as the first element. Since each cell of $\mathcal{S}'$ belongs to the well-covering region $\mathcal{U}(e)$ of $O(1)$ transparent edges e, each bisector event of $SPM'(s)$ is charged $O(1)$ times for all pairs of Π'. Because $SPM'(s)$ has $O(h)$ bisector events by Corollary 3.5, the number of pairs of Π' is at most $O(h)$.

This completes the proof of the lemma. □

Armed with Lemma 3.18, we prove the subsequent five lemmas, which together lead to Lemma 3.15.

Lemma 3.19. *The total number of marked generators by Rule 2(a) and Rule 3 is $O(h)$.*

Proof. Suppose α is a generator marked by Rule 2(a) for a transparent edge e. Then, α must be the first or last non-artificial generator of $W(e)$. Hence, at most two generators of $W(e)$ can be marked by Rule 2(a). As $\mathcal{S}'$ has $O(h)$ transparent edges, the total number of generators marked by Rule 2(a) is $O(h)$.

Suppose α is a generator marked by Rule 3(a) for the two transparent edges (e, g) with $g \in output(e)$. Since α claims an endpoint of g, α must be the first or last generator of $W(e, g)$. Hence, at most two generators of $W(e, g)$ can be marked by Rule 3(a). Since $|output(e)| = O(1)$, at most $O(1)$ generators can be marked for the pairs of transparent edges with e as the first edge. Since $\mathcal{S}'$ has $O(h)$ transparent edges, the total number of generators marked by Rule 3(a) is $O(h)$.

Suppose α is a generator marked by Rule 3(b) for the two transparent edges (e, g) with $g \in output(e)$. Note that α is a generator in $W(e)$. We assume that α is not the first or last non-artificial generators of $W(e)$; the first and last non-artificial generators of $W(e)$, countered separately, sum to $O(h)$ for all transparent edges e of $\mathcal{S}'$. Let α_1 and α_2 be the two neighboring generators of α in $W(e)$. Thus, both α_1 and α_2 are non-artificial. We assume that the bisectors $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$ intersect e in $SPM'(s)$; by Lemma 3.18, there are only $O(h)$ bisector and transparent edge intersections that appear in some wavefront but not in $SPM'(s)$.

[5] This is also due to that for any point $p \in \overline{q_1q_2}$, the shortest path $\pi(s, p)$ cannot intersect e; this can be proved by a similar argument to the preceding for proving that q_1 is claimed by α_1 in $SPM'(s)$. This observation implies that α cannot claim any points on $\overline{q_1q_2}$ in $SPM'(s)$.

Since α is marked by Rule 3(b), at least one of $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$ fails to reach the boundary of $D(e)$ during the algorithm, where $D(e)$ is the union of cells through which $W(e)$ is propagated to all edges $g' \in output(e)$. Note that $D(e) \subseteq \mathcal{U}(e) \cup \bigcup_{g' \in output(e)} \mathcal{U}(g')$, which contains $O(1)$ cells of $\mathcal{S}'$ as $|output(e)| = O(1)$ and the well-covering region of each transparent edge is the union of $O(1)$ cells of $\mathcal{S}'$; in addition, each cell of $D(e)$ is within a constant number of cells of e. Without loss of generality, we assume that $B(\alpha_1, \alpha)$ does not reach the boundary of $D(e)$ and a bisector event on $B(\alpha_1, \alpha)$ is detected during the algorithm. The detected bisector event on $B(\alpha_1, \alpha)$ also implies that an actual bisector event in $SPM'(s)$ happens to $B(\alpha_1, \alpha)$ no later than the detected event; we charge the marking of α to that bisector actual endpoint (i.e., a vertex) in $SPM'(s)$, and the endpoint is in $D(e)$ because $B(\alpha_1, \alpha)$ intersects e in $SPM'(s)$. Since each cell of $D(e)$ is within a constant number of cells of e, each cell of $\mathcal{S}'$ belongs to $D(e')$ for a constant number of transparent edges e' of $\mathcal{S}'$. Therefore, each vertex of $SPM'(s)$ is charged $O(1)$ times. Since $SPM'(s)$ has $O(h)$ vertices by Lemma 3.4, the total number of generators marked by Rule 3(b) is $O(h)$. □

Lemma 3.20. *The total number of marked generators by Rule 1 is $O(h)$.*

Proof. According to our algorithm, generators are only created during the wavefront propagation procedure, which is to propagate $W(e)$ to compute $W(e, g)$ for all edges $g \in output(e)$ through $\mathcal{U}$, where $\mathcal{U}$ is $\mathcal{U}(e)$ or $\mathcal{U}(g)$. The procedure has two cases depending on whether a cell c of $\mathcal{U}$ is an empty rectangle. If c is an empty rectangle, then no generators will be created in c. Otherwise, c may be partitioned into $O(1)$ subcells, each of which may have a convex chain on its left side and/or its right side. Let c be such a subcell. Without loss of generality, we assume that the current wavefront W is propagating in c from bottom to top. We consider the generators created on the left side ζ_l of c.

According to our algorithm, a generator on ζ_l is created by the leftmost generator of the current wavefront W. As such, if the leftmost wavelet of W does not create a generator, then no generator on ζ_l will be created. Otherwise, let α be a generator created on ζ_l, which becomes the leftmost generator of W at the point of creation. Let α' be the right neighbor of α in W. According to our algorithm, if α does not involve in any bisector event in c—that is, the bisector $B(\alpha, \alpha')$ does not intersect any other bisector in c during the propagation of W in c, then no more new generators will be created on ζ_l. Otherwise, we charge the creation of α to the bisector event involving $B(\alpha, \alpha')$; each such event can be charged at most twice (one for a generator on ζ_l and the other for a generator created on the right side of c). Recall that for each bisector event, a generator is marked by Rule 3(b).

In light of the preceding discussion, since each cell of $\mathcal{S}'$ belongs to the well-covering region $\mathcal{U}(e')$ of $O(1)$ transparent edges e', the total number of generators marked by Rule 1 is $O(h)+O(k)$, where k is the total number of generators marked by Rule 3(b), which is $O(h)$ by Lemma 3.19. The lemma thus follows. □

Lemma 3.21. *The total number of marked generators by Rule 4 is $O(h)$.*

Proof. Let α be the generator and e be the transparent edge specified in the rule statement. According to Rule 4, α is marked because it claims part of an obstacle edge during the wavefront propagation procedure for propagating $W(e)$ to compute $W(e, g)$ for edges $g \in output(e)$. As in the proof of Lemma 3.19, define $D(e)$ as the union of cells through which $W(e)$ is propagated to all edges $g \in output(e)$. Recall that $D(e)$ contains $O(1)$ cells of $\mathcal{S}'$, and each cell of $D(e)$ is within a constant number of cells of e, which implies that each cell of $\mathcal{S}'$ belongs to $D(e')$ for a constant number of transparent edges e' of $\mathcal{S}'$.

First of all, for the case where α is a generator created during the wavefront propagation procedure, the total number of such marked generators is no more than the total number of marked generators by Rule 1, which is $O(h)$ by Lemma 3.20.

Second, for the case where α is the first or last generator in $W(e)$, the total number of such generators is clearly $O(h)$ since $\mathcal{S}'$ has $O(h)$ transparent edges e.

Third, for the case where α claims a rectilinear extreme vertex, the total number of such marked generators is $O(h)$. To see this, since $D(e)$ has $O(1)$ cells of $\mathcal{S}'$ and each cell contains at most one rectilinear extreme vertex, $D(e)$ contains $O(1)$ rectilinear extreme vertices. Therefore, at most $O(1)$ generators will be marked in $D(e)$. Since each cell of $\mathcal{S}'$ belongs to $D(e')$ for $O(1)$ transparent edges e' of $\mathcal{S}'$ and $\mathcal{S}'$ contains $O(h)$ transparent edges e, the total number of marked generators in this case is $O(h)$.

Finally, any Rule 4 marked generator α that does not belong to any of the preceding cases has the following property: α does not claim a rectilinear extreme vertex, and is not the first or last non-artificial generator in $W(e)$, and is not a generator created in $D(e)$. Let α' be a neighbor of α in $W(e)$. Due to the preceding property, α' is a non-artificial generator. We can assume that $B(\alpha', \alpha)$ intersects e in $SPM'(s)$; by Lemma 3.18, there are only $O(h)$ bisector and transparent edge intersections that appear in some wavefront but not in $SPM'(s)$. Hence, in $SPM'(s)$, $B(\alpha', \alpha)$ terminates in $D(e)$, either on an obstacle edge or in a bisector event before reaching an obstacle edge. In either case, we charge the mark of α at e to this endpoint of $B(\alpha', \alpha)$ in $SPM'(s)$, which is vertex of $SPM'(s)$. Because each cell of $\mathcal{S}'$ belongs to $D(e')$ for a constant number of transparent edges e' of $\mathcal{S}'$, each vertex of $SPM'(s)$ is charged at most $O(1)$ times. As $SPM'(s)$ has $O(h)$ vertices by Lemma 3.4, the total number of marked generators by Rule 4 is $O(h)$. □

Lemma 3.22. *The total number of marked generators by Rule 2(b) is $O(h)$.*

Proof. Let α be the generator and e be the transparent edge specified in the rule statement.

First of all, the total number of marked generators in the case where α is in $\mathcal{U}(e)$ is $O(h)$, because the number of Rule 1 marked generators is $O(h)$ by Lemma 3.20 and $\mathcal{U}(e)$ has $O(1)$ cells. In the following, we consider the other case where α is outside $\mathcal{U}(e)$.

Since α is outside $\mathcal{U}(e)$, there is a transparent edge f on $\partial\mathcal{U}(e)$ (i.e., $f \in input(e)$) such that α is in $W(f)$ and it is in $W(e)$ because $W(f)$ is propagated to e through $\mathcal{U}(e)$. Since α's claim is shortened or eliminated by an artificial wavefront, there must be a bisector event involving α during the computation of $W(f, e)$ from $W(f)$. Hence, α is also marked by Rule 3(b). We charge α to this Rule 3(b) mark for f. Since there are $O(h)$ generators marked by Rule 3(b) by Lemma 3.19, the total number of marked generators of Rule 2(b) is $O(h)$. □

3.8 Computing the Shortest Path Map *SPM(s)*

In this section, we compute the shortest path map $SPM(s)$. To this end, we need to mark generators following our rules during our wavefront merging and propagation procedures. As the total number of all marked generators is $O(h)$, marking these generators does not change the running time of our algorithm asymptotically. In the following, we show that $SPM(s)$ can be constructed in $O(n + h \log h)$ time with the help of the marked generators. In light of Lemma 3.6, we will focus on constructing $SPM'(s)$.

We first show in Section 3.8.1 that the marked generators are sufficient in the sense that if a generator participates in a true bisector event of $SPM'(s)$ in a cell c of $\mathcal{S}'$, then α must be marked in c. Then, we present the algorithm for constructing $SPM'(s)$, which consists of two main steps. The first main step is to identify all vertices of $SPM'(s)$ and the second one is to compute the edges of $SPM'(s)$ and assemble them to obtain $SPM'(s)$. The first main step is described in Section 3.8.2, whereas the second one is discussed in Section 3.8.3.

3.8.1 Correctness of the Marked Generators. In this section, we show that if a generator participates in a bisector event of $SPM'(s)$ in a cell c of $\mathcal{S}'$, then α must be marked in c. Our algorithm

in the next section for computing $SPM'(s)$ relies on this property. Our proof strategy again follows the high-level structure of the HS algorithm. We first prove the following lemma.

Lemma 3.23. *Let α be a generator in an approximate wavefront $W(e)$ for some transparent edge e. Suppose there is a point $p \in e$ that is claimed by α in $W(e)$ but not in $SPM'(s)$ (because the approximate wavefront from the other side of e reaches p first). Then, α is marked in the cell c on α's side of e.*

Proof. Assume to the contrary that α is not marked in c. Then, α must have two neighboring non-artificial generators α_1 and α_2 in $W(e)$ since otherwise Rule 2 would apply. In addition, because all transparent edges of $\partial\mathcal{U}(e)$ are in $output(e)$, the two bisectors $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$ must exist the well-covering region $\mathcal{U}(e)$ of e through the same transparent edge $g \in \partial\mathcal{U}(e)$, since otherwise Rule 3 or 4 would apply. Further, the region R bounded by $B(\alpha_1, \alpha)$, $B(\alpha, \alpha_2)$, e, and g must be a subset of $\mathcal{U}(e)$. Indeed, if R contains an island not in $\mathcal{U}(e)$, then α would claim an endpoint of a boundary edge of the island, in which case Rule 3 would apply.

Let $\alpha' = (A, a)$ be the true predecessor of p in $SPM'(s)$. Without loss of generality, we assume that e is horizontal and α is below e and thus α' is above e. Let $\pi'(a, p)$ be the path from p along its tangent to A and then following A to a.

We first consider the case where α' is not in the interior of the well-covering region $\mathcal{U}(e)$—that is, the initial vertex a is not in the interior of $\mathcal{U}(e)$. Since R is a subset of of $\mathcal{U}(e)$ without non-$\mathcal{U}(e)$ islands, a is not in the interior of R and thus $\pi'(a, p)$ intersects ∂R, say, at a point q. In the following, we argue that α has been involved in a bisector event detected by our algorithm and thus marked in c. Let g' be the portion of g between its intersections with $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$. Depending on whether $q \in g'$, there are two subcases:

- If $q \in g'$ (Figure 26), then $\tilde{d}(s, g) \le d(s, a) + |\pi'(a, q)| + |g|/2$, where $\pi'(a, q)$ is the subpath of $\pi'(a, p)$ between a and q. Hence, the time τ_g when artificial wavefronts originating from the endpoints of g cover g is no later than $\tilde{d}(s, g) + |g| \le d(s, a) + |\pi'(a, q)| + 3|g|/2$. Because $g \in \partial\mathcal{U}(e)$, $|\overline{pq}| \ge 2 \cdot \max\{|g|, |e|\}$. Hence, $\tau_g \le d(s, a) + |\pi'(a, q)| + 3|g|/2 < d(s, a) + |\pi'(a, q)| + |\overline{qp}| = d(s, a) + |\pi'(a, p)| < d_{W(e)}(s, p)$, where $d_{W(e)}(s, p)$ is the length of the path from s to p following the wavefront $W(e)$; the last inequality holds because α' is the true predecessor of p while α is not.

  However, the time τ_e when the wavefront $W(e)$ reaches an endpoint of e cannot be earlier than $d_{W(e)}(s, p) - |e|$. Hence, the wavelet from α cannot reach g earlier than $d_{W(e)}(s, p) - |e| + d(e, g) \ge d_{W(e)}(s, p) - |e| + 2|e| \ge d_{W(e)}(s, p) + |e|$, which is larger than τ_g since $\tau_g < d_{W(e)}(s, p)$ as proved previously. Therefore, by the time the wavelet from α reaches g, the artificial wavelets of g have already covered g, eliminating the wavelet from α from reaching g. Thus, α must be marked by Rule 3(b).
- If $q \notin g'$, then q is on one of the bisectors $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$. Let $\pi(s, a)$ be a shortest path from s to a and let $\pi(s, p) = \pi(s, a) \cup \pi'(a, p)$, which is a shortest path from s to p. If $\pi(s, p)$ intersects g, then we can use the same analysis as earlier to show that the wavelet from α will be eliminated from reaching g by the artificial wavelets of g and thus α must be marked by Rule 3(b). In the following, we assume that $\pi(s, p)$ does not intersect g.

  Without of loss of generality, we assume that $q \in B(\alpha_1, \alpha)$ (Figure 27). Since $\pi'(a, p)$ is a subpath of the shortest path $\pi(s, p)$, every point of $\pi'(a, p)$ has α' as its predecessor. As $q \in \pi'(a, p)$, the predecessor of q is α'. Let $\pi(s, q)$ be the subpath of $\pi(s, p)$ between s and q, which is a shortest s-q path. Since $\pi(s, p)$ does not intersect g, $\pi(s, q)$ does not intersect g. Hence, $\pi(s, q)$ is a shortest s-q path in the modified free space $\mathcal{F}'$ by replacing g with an opaque edge of open endpoints. Therefore, during the wavefront propagation procedure for computing $W(e, g)$ or during the wavefront merging procedure for computing $W(g)$, the

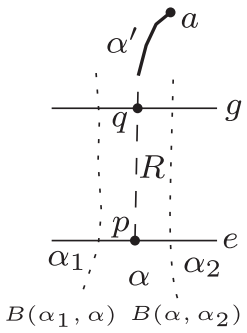

Fig. 26. Illustrating the case where $q \in g'$.

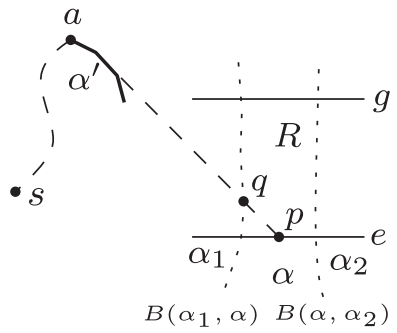

Fig. 27. Illustrating the case where $q \in B(\alpha_1, \alpha)$.

point q, which is on the bisector $B(\alpha_1, \alpha)$, must be claimed by α'. Hence, a bisector event must be detected for $B(\alpha_1, \alpha)$ during the computation of $W(e, g)$ or $W(g)$. In either case, α must be marked by Rule 3(b).

We next consider the case where α' lies inside $\mathcal{U}(e)$—that is, its initial vertex a is inside $\mathcal{U}(e)$. If a is not between the two bisectors $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$, then $\pi'(a, p)$ must intersect one of the bisectors and thus we can use a similar argument as in the preceding second case to show that α must be marked. Otherwise, a is in the region R. This implies that not all points in R are claimed by α when $W(e)$ is propagating to g, and therefore a bisector event involving α must happen during the wavefront propagation procedure to propagate $W(e)$ to compute $W(e, g)$ and thus α is marked by Rule 3(b). □

With the help of the preceding lemma, we prove the following lemma.

Lemma 3.24. *If a generator α participates in a bisector event of $SPM'(s)$ in a cell c of $\mathcal{S}'$, then α must be marked in c.*

Proof. If a bisector has an endpoint on an obstacle edge of c, it either emanates from an obstacle vertex a on the edge (i.e., the bisector is an extension bisector) or is defined by two generators that claim part of the opaque edge. In the first case, a is the initial vertex of a new created generator in c and thus the generator is marked by Rule 1. In the second case, both generators are marked by Rule 4.

Let α be a generator that participates in a bisector event of $SPM'(s)$ in a cell c of $\mathcal{S}'$. Assume to the contrary that α is not marked for c. Then, by Rule 2(a), there must be transparent edges e and f on the boundary of c such that both $W(e)$ and $W(f)$ contain the three consecutive generators α_1, α, and α_2. Without loss of generality, we assume that $W(e)$ enters c and $W(f)$ leaves c. Let R be the region of c bounded by e, f, and the two bisectors $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$ (Figure 28). The region R must be a subset of c. Indeed, if R contains an island not in c, then α would claim an endpoint of a boundary edge of the island, in which case Rule 3 would apply.

Since α participates in a bisector event of $SPM'(s)$ in c, at least one point p in R is not claimed by α in $SPM'(s)$. Let $\alpha' = (A, a)$ be the true predecessor of p. Note that the initial vertex a must be outside R since otherwise a bisector event involving α must happen when $W(e)$ is propagating through c and thus α would be marked by Rule 3(b). Let $\pi'(a, p)$ be the path from p along its tangent to A and then following A to a. Since a is outside R and p is inside R, $\pi'(a, p)$ must intersect the boundary of R.

Because α_1, α, and α_2 are three consecutive generators of $W(e)$, no generator other than α on the same side of e as α claims any point of R. Thus, $\pi'(a, p)$ does not cross e. Let b_1 and b_2 be

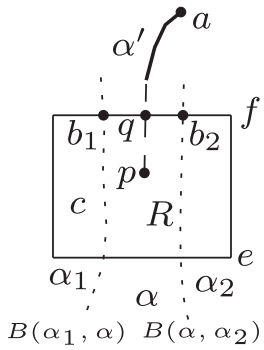

Fig. 28. Illustrating the proof of Lemma 3.24.

the intersections of f with $B(\alpha_1, \alpha)$ and $B(\alpha, \alpha_2)$, respectively. Then, if α claims both b_1 and b_2 in $SPM'(s)$, then $\pi'(a, p)$ cannot cross either bisector on ∂R, and thus it must cross $\overline{b_1 b_2}$, say, at a point q (e.g., see Figure 28). Note that q satisfies the hypothesis of Lemma 3.23, and thus α is marked for c. If α does not claim either b_1 or b_2, then that point satisfies the hypothesis of Lemma 3.23 and thus α is marked for c. □

3.8.2 Computing the Vertices of $SPM'(s)$. In this section, we compute the vertices of $SPM'(s)$. The next section will compute the edges of $SPM'(s)$ and assemble them to obtain $SPM'(s)$.

Computing Active Regions. Because the wavefronts are represented by persistent trees, after the preceding wavefront expansion algorithm finishes, the wavefronts $W(e)$ for all transparent edges e of $\mathcal{S}'$ are still available. In addition, for each cell c and each transparent edge e of c, a set of marked generators in $W(e)$ are known. Using these marked generators, we first break c into *active* and *inactive* regions such that no vertices of $SPM'(s)$ lie in the inactive regions. Note that each unmarked generator does not participate in a bisector event in $SPM'(s)$ by Lemma 3.24. If α_1 and α_2 are neighboring generators on ∂c such that one of them is marked while the other is unmarked, their bisector belongs to $SPM'(s)$. Therefore, all such generators are disjoint and they together partition c into regions such that each region is claimed only by marked generators or only by unmarked generators; the former regions are active while the latter are inactive. We can compute these active regions as follows.

Since the wavefronts $W(e)$ for all transparent edges e of c are available, the list of all generators ordered along the boundary of c is also available. For each pair of adjacent generators α_1 and α_2, if one of them is marked and the other is unmarked, then we explicitly compute the portion of their bisector $B(\alpha_1, \alpha_2)$ in c. We describe the details of this step in the following.

First of all, observe that α_1 and α_2 must be from $W(e)$ for the same transparent edge e of c, since otherwise each generator must claim an endpoint of their own transparent edge and thus must have been marked by Rule 2(a). Without loss of generality, we assume that e is horizontal and both α_1 and α_2 are below e. Note that we cannot afford computing the entire bisector $B(\alpha_1, \alpha_2)$ and then determining its portion in c because the running time would be proportional to the size of $B(\alpha_1, \alpha_2)$—that is, the number of hyperbolic-arcs of $B(\alpha_1, \alpha_2)$. Instead, we first compute the intersection b of $B(\alpha_1, \alpha_2)$ and e, which can be done in $O(\log n)$ time by the bisection-line intersection operation in Lemma 3.10. Note that b is one endpoint of the portion of $B(\alpha_1, \alpha_2)$ in c. To determine the other endpoint b', we do the following. Our goal is to compute b' in $O(|B_c| + \log n)$ time, with $B_c = B(\alpha_1, \alpha_2) \cap c$. To this end, we could trace B_c in c from b, and for each hyperbolic-arc of B_c, we determine whether it intersects ∂c. Recall that ∂c consists of $O(1)$ transparent edges

and at most $O(1)$ convex chains. To achieve the desired runtime, we need to determine whether each hyperbolic-arc of B_c intersects ∂c in $O(1)$ time. However, it is not clear to us whether this is possible as the size of each convex chain may not be of $O(1)$ size. To circumvent the issue, we use the following strategy. Before tracing B_c, we first compute the intersection between $B(\alpha_1, \alpha_2)$ and each convex chain on ∂c, which can be done in $O(\log n)$ time by the bisector-chain intersection operation in Lemma 3.13. Among all intersections, let b'' be the closest one to α. Then, we start to trace B_c from b, for each hyperbolic-arc e', we determine whether e' intersects each of the transparent edges of ∂c. If not, we further check whether e' contains b''. If $b'' \notin e'$, then e' is in c and we continue the tracing; otherwise, b' is b'' and we stop the algorithm. If e' intersects at least one of the transparent edges of ∂c, then among all such intersections as well as b'', b' is the one closest to α, which can be determined in $O(\log n)$ time by computing their tangents to α. In this way, computing the portion of $B(\alpha_1, \alpha_2)$ in c can be done in $O(\log n + n_c(\alpha_1, \alpha_2))$ time, where $n_c(\alpha_1, \alpha_2)$ is the number of hyperbolic-arcs of $B(\alpha_1, \alpha_2)$ in c.

In this way, all active regions of c can be computed in $O(h_c \log n + n_c)$ time, where h_c is the total number of marked generators in the wavefronts $W(e)$ of all transparent edges e of c and n_c is the total number of hyperbolic-arcs of the bisector boundaries of these active regions.

Computing the Vertices of $SPM'(s)$ in Each Active Region. In what follows, we compute the vertices of $SPM'(s)$ in each active region R of c.

The boundary ∂R consists of $O(1)$ pieces, each of which is a transparent edge fragment, an elementary chain fragment, or a bisector in $SPM'(s)$. Unlike the HS algorithm, where each of these pieces is of $O(1)$ size, in our problem both an elementary chain fragment and a bisector may not be of constant size. Let e be a transparent edge of R. Without loss of generality, we assume that e is horizontal and R is locally above e. Without considering other wavefronts, we use $W(e)$ to partition R into subregions, called *Voronoi faces*, such that each face has a unique predecessor in $W(e)$. We use $Vor(e)$ to denote the partition of R. Note that if α is the predecessor of a Voronoi face, then for each point p in the face, its tangent to α must cross e and we assign p a *weight* that is equal to $d(\alpha, p)$. In addition, it is possible that these faces together may not cover the entire R; for those points outside these faces, their weights are ∞.

We now discuss how to compute the partition $Vor(e)$. To this end, we can use our wavefront propagation procedure to propagate $W(e)$ inside R. To apply the algorithm, one issue is that the boundary of R may contain bisectors, which consists of hyperbolic-arcs instead of polygonal segments. To circumvent the issue, an observation is that the bisectors of $W(e)$ do not intersect the bisectors on ∂R. Indeed, for each bisector B of ∂R, one of its defining generators is unmarked and thus does not participate in any bisector event in c, and therefore B does not intersect any bisector in c. In light of the observation, we can simply apply the wavefront propagation procedure to propagate $W(e)$ to c instead of R to partition c into Voronoi faces, denoted by $Vor_c(e)$. The preceding observation implies that each bisector on the boundary of R must lie in a single face of $Vor_c(e)$. Hence, we can simply cut $Vor_c(e)$ to obtain $Vor(e)$ using the bisectors on the boundary of R. When we apply the wavefront propagation procedure to propagate $W(e)$ to c, here we add an initial step to compute the intersection of e and $B(\alpha, \alpha')$ for each pair of neighboring generators α and α' of $W(e)$ and use it as the initial tracing-point $z(\alpha, \alpha')$. Since each such intersection can be computed in $O(\log n)$ time by the bisector-line intersection operation in Lemma 3.10, this initial step takes $O(h_e \log n)$ time, where h_e is the number of generators of $W(e)$. Hence, the total time for propagating $W(e)$ into c to obtain $Vor_c(e)$ is $O(h_e \log n + n_B(e))$, where $n_B(e)$ is the number of hyperbolic-arcs of the bisectors of $W(e)$ in c. After having $Vor_c(e)$, cutting it to obtain $Vor(e)$ can be easily done in $O(h_e + n_B(e) + n_R)$ time, where n_R is the number of hyperbolic-arcs on the bisectors of ∂R. Therefore, the total time for computing the partition $Vor(e)$ is $O(h_e \log n + n_B(e) + n_R)$

time. Note that since the bisectors of ∂R do not intersect any bisectors involving the generators of $W(e)$, all bisector events of $W(e)$ in c are actually in R.

After having the partition $Vor(e)$ for all transparent edges e of R, we can now compute vertices of $SPM'(s)$ in R. Consider a transparent edge e of ∂R. We process it as follows. For each transparent edge f of ∂R other than e, we merge the two partitions $Vor(e)$ and $Vor(f)$ by using the merge step from the standard divide-and-conquer Voronoi diagram algorithm to compute the subregion of R closer to $W(e)$ than to $W(f)$. This can be done in $O(h_e + n_B(e) + n_R + h_f + n_B(f))$ time by finding a finding a curve γ in R that consists of points equal to $W(e)$ and $W(f)$, and the algorithm is similar to the HS algorithm.

Intersecting the results for all such f produces the region $R(e)$ claimed by $W(e)$ in R. Intersecting $R(e)$ with $Vor(e)$ gives the vertices of $SPM'(s)$ in R that $W(e)$ contributes. Repeating the preceding for all transparent edges e of ∂R gives the vertices of $SPM'(s)$ in R. Since ∂R has $O(1)$ transparent edges, the total time is $O(h_R \log n + n_B(R) + n_R)$, where h_R is the number of generators in all wavefronts of all transparent edges of ∂R and $n_B(R)$ is the total number of hyperbolic-arcs in R on the bisectors of all wavefronts of all transparent edges of ∂R.

We do the preceding for all active regions R of c, then all vertices of $SPM'(s)$ in c can be computed. The total time is $O(h_c \log n + n_R(c) + n_B(c))$, where h_c is the total number of marked generators in the wavefronts $W(e)$ of all transparent edges e of c, $n_R(c)$ is the total number of hyperbolic-arcs of the bisector boundaries of the active regions of c, and $n_B(c)$ is the total number of hyperbolic-arcs in c on the bisectors of the wavefronts of all transparent edges of c. Processing all cells c of $\mathcal{S}'$ as earlier gives all vertices of $SPM'(s)$. For the running time, the total sum of $n_R(c)$ among all cells $c \in \mathcal{S}'$ is $O(n)$ because each hyperbolic-arc of a bisector on the boundary of any active region also appears in $SPM'(s)$, whose total size is $O(n)$ by Corollary 3.3. The total sum of h_c among all cells c is $O(h)$ by Lemma 3.15. For $n_B(c)$, Lemma 3.17 is essentially a proof that the total sum of $n_B(c)$ over all cells c is $O(n)$. To see this, since c is a cell incident to e, the wavefront $W(e)$ will be propagated into c during the wavefront propagation procedure to propagate $W(e)$ to edges of $output(e)$, and thus the hyperbolic-arcs of the bisectors of $W(e)$ are counted in the proof analysis of Lemma 3.17. In summary, the total time for computing all vertices of $SPM'(s)$ is $O(n + h \log n)$, which is $O(n + h \log h)$.

3.8.3 Constructing $SPM'(s)$. With all vertices of $SPM'(s)$ computed previously, as in the HS algorithm, we next compute the edges of $SPM'(s)$ separately and then assemble them to obtain $SPM'(s)$. One difference is that the HS algorithm uses a standard plane sweep algorithm to assemble all edges to obtain $SPM(s)$, which takes $O(n \log n)$ time as there are $O(n)$ edges in $SPM(s)$. To achieve the desired $O(n + h \log h)$ time bound, here instead we propose a different algorithm.

We first discuss how to compute the edges of $SPM'(s)$, given the vertices of $SPM'(s)$. Recall that each vertex v of $SPM'(s)$ is either an intersection between a bisector and an obstacle edge, or an intersection of bisectors. By our general position assumption, in the former case, v is in the interior of an obstacle edge; in the latter case, v is the intersection of three bisectors (i.e., a *triple point*). During our preceding algorithm for computing vertices of $SPM'(s)$, we associate each vertex v with the two generators of the bisector that contains v (more specifically, we associate v with the initial vertices of the generators). We create a list of all vertices of $SPM'(s)$, each identified by a *key* consisting of its two generators. If a vertex v is a triple point, then we put it in the list for three times (each time with a different pair of generators); if v is a bisector-obstacle intersection, then it is put in the list once. We now sort all vertices of $SPM'(s)$ with their keys; by traversing the sorted list, we can group together all vertices belong to the same bisector. This sorting takes $O(h \log n)$ time, as there are $O(h)$ vertices in $SPM'(s)$.

We take all $SPM'(s)$ vertices in the same bisector and sort them along the bisector determined by their weighted distances from the generators of the bisector (each of these distances can be computed in additional $O(\log n)$ time by computing the tangents from the vertex to the generators).

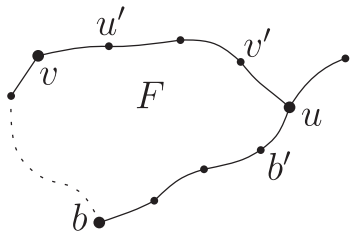

Fig. 29. Illustrating v, u, u', v', b', and b.

These sorting takes $O(h \log n)$ time altogether. The preceding computes the bisector edges e of $SPM'(s)$ that connect two adjacent vertices. In fact, each edge e is only implicitly determined in the sense that the hyperbolic-arcs of e are not explicitly computed. In addition, for each obstacle P, we sort all vertices of $SPM'(s)$ on ∂P to compute the convex chain edges of $SPM'(s)$ connecting adjacent vertices of $SPM'(s)$ on ∂P. This sorting can be easily done in $O(n + h \log n)$ time for all obstacles.

For each vertex v of $SPM'(s)$, the preceding computes its adjacent vertices. As discussed earlier, due to the general position assumption, v has at most three adjacent vertices. We next define a set $E(v)$ of at most three points for v. For each adjacent vertex u of v in $SPM'(s)$, let $e(v, u)$ be the edge of $SPM'(s)$ connecting them. $e(v, u)$ is either a bisector edge or a convex chain edge. In the former case, we refer to each hyperbolic-arc of $e(v, u)$ as a *piece* of $e(v, u)$; in the latter case, we refer to each obstacle edge of $e(v, u)$ as a *piece*. If we traverse $e(v, u)$ from v to u, the endpoint u' of the first piece (incident to v) is added to $E(v)$; we define $h(u')$ to be u. Since v is adjacent to at most three vertices in $SPM'(s)$, $|E(v)| \leq 3$. In fact, $E(v)$ is exactly the set of vertices adjacent to v in the shortest path map $SPM(s)$.

With all edges of $SPM'(s)$ and the sets $E(v)$ of all vertices v of $SPM'(s)$ computed previously, we construct the **Doubly-Connected-Edge-List (DCEL)** of $SPM'(s)$, as follows. For convenience, we assume that there is a bounding box that contains all obstacles so that no bisector edge of $SPM'(s)$ will go to infinity and thus each face of $SPM'(s)$ is bounded.

For each vertex v of $SPM'(s)$, we store its coordinate in the DCEL data structure. As $|E(v)| \leq 3$, there are at most three faces in $SPM'(s)$ incident to v, and we construct them one by one. For each point u' in $E(v)$, we construct the face F clockwise (with respect to v) incident to the edge $e(v, u)$ of $SPM'(s)$ connecting v and $u = \phi(u')$, as follows (Figure 29). We first trace out the edge $e(v, u)$, which is either a bisector edge or a convex chain edge. In the former case, we compute the hyperbolic-arcs of $e(v, u)$, one at a time, until we reach u, and add them to the DCEL data structure. Each hyperbolic-arc can be computed in constant time using the two generators of the bisector. In the latter case, we trace out the obstacle edges of $e(v, u)$, one at a time and add them to the DCEL data structure. Let v' be the last endpoint of the piece of $e(v, u)$ containing u (e.g., see Figure 29). Note that v' is in the set $E(u)$. Let b' be the first point of $E(u)$ counterclockwise around u after v' (e.g., see Figure 29). Let $b = \phi(b')$, which is a vertex of $SPM'(s)$ adjacent to u. Hence, the edge $e(u, b)$ of $SPM'(s)$ connecting u to b is incident to the face F. We trace out the edge $e(u, b)$ in the same way as earlier. When we reach b, we continue to trace the next edge of F. Since each face of $SPM'(s)$ is bounded, eventually we will arrive back to the vertex v again.[6] This finishes the construction of the face F. We do the same for all other faces of $SPM'(s)$, after which the DCEL data structure for

[6]We could lift the assumption that each face of $SPM'(s)$ is bounded in the following way. During the preceding algorithm for constructing F, if F is unbounded, then we will reach a bisector edge that extends to the infinity. If that happens, then we construct other edges of F from the other direction of v. More specifically, starting from the first point of $E(v)$

$SPM'(s)$ is constructed. For the running time, since each edge of $SPM'(s)$ is traced at most twice, by Corollary 3.3, the total time of the preceding procedure for constructing the DCEL data structure is $O(n)$.

In summary, $SPM'(s)$ can be computed in $O(n + h \log h)$ time. By Lemma 3.6, the shortest path map $SPM(s)$ can be built in additional $O(n)$ time.

3.9 Reducing the Space to $O(n)$

The preceding provides an algorithm for computing $SPM(s)$ in $O(n+h \log h)$ time and $O(n+h \log h)$ space. In this section, we reduce the space to $O(n)$, using the technique given in prior work [48].

The reason that the preceding algorithm needs $O(n+h \log h)$ space is twofold. First, it uses fully persistent binary trees (with the path-copying method) to represent wavefronts $W(e)$. Because there are $O(h)$ bisector events in the wavefront expansion algorithm and each event costs $O(\log n)$ additional space on a persistent tree, the total space of the algorithm is $O(n + h \log n)$. Second, to construct $SPM(s)$ after the wavefront expansion algorithm, the wavefronts $W(e)$ of all transparent edges e of $\mathcal{S}'$ are needed, which are maintained in those persistent trees. We resolve these two issues in the following way.

3.9.1 Reducing the Space of the Wavefront Expansion Algorithm. We still use persistent trees to represent wavefronts. However, as there are $O(h)$ bisector events in total in the algorithm, we divide the algorithm into $O(\log h)$ phases so that each phase has no more than $h/\log h$ events. The total additional space for processing the events using persistent trees in each phase is $O(h)$. At the end of each phase, we "reset" the space of the algorithm by only storing a "snapshot" of the algorithm (and discarding all other used space) so that (1) the snapshot contains sufficient information for the subsequent algorithm to proceed as usual and (2) the total space of the snapshot is $O(h)$.

Specifically, we make the following changes to the wavefront propagation procedure, which is to compute the wavefronts $W(e, g)$ for all edges $g \in output(e)$ using the wavefront $W(e)$. We now maintain a counter $count$ to record the number of bisector events that have been processed so far since the last space reset; $count = 0$ initially. Consider a wavefront propagation procedure on the wavefront $W(e)$ of a transparent edge e. The algorithm will compute $W(e, g)$ for all edges $g \in output(e)$, by propagating $W(e)$. We apply the same algorithm as before. For each bisector event, we first do the same as before. Then, we increment $count$ by 1. If $count < h/\log h$, we proceed as before (i.e., process the next event). Otherwise, we have reached the end of the current phase and will start a new phase. To do so, we first reset $count = 0$ and then reset the space by constructing and storing a snapshot of the algorithm (other space occupied by the algorithm is discarded), as follows:

(1) Let g refer to the edge of $output(e)$ whose $W(e, g)$ is currently being computed in the algorithm. We store the tree that is currently being used to compute $W(e, g)$ right after the preceding event. To do so, we can make a new tree by copying the newest version of the current persistent tree the algorithm is operating on. The size of the tree is bounded by $O(h)$. We will use this tree to "resume" computing $W(e, g)$ in the subsequent algorithm.
(2) For each $g' \in output(e)\backslash\{g\}$ whose $W(e, g')$ has been computed, we store the tree for $W(e, g')$. We will use the tree to compute the wavefronts $W(g')$ of g' in the subsequent algorithm.
(3) We store the tree for the wavefront $W(e)$. Note that the tree may have many versions due to processing the events and we only keep its original version for $W(e)$. Hence, the size of

clockwise around v after u', we trace out the edges of F in the same way as earlier, until we reach a bisector edge that extends to the infinity, at which moment all edges of F are constructed.

the tree is $O(h)$. This tree will be used in the subsequent algorithm to compute $W(e, g')$ for those edges $g' \in output(e) \setminus \{g\}$ whose $W(e, g')$ have not been computed yet.

(4) We check every transparent edge e' of $\mathcal{S}'$ with $e' \neq e$. If e' has been processed (i.e., the wavefront propagation procedure has been called on $W(e')$) and there is an edge $g' \in output(e')$ that has *not* been processed, we know that $W(e', g')$ has been computed and is available; we store the tree for $W(e', g')$. We will use the tree to compute the wavefronts $W(g')$ of g' in the subsequent algorithm.

We refer to the wavefronts stored in the algorithm as the *snapshot*; intuitively, the snapshot contains all wavelets in the forefront of the wavelet expansion. By analysis similar to that in previous work [48], we can show that the snapshot contains sufficient information for the subsequent algorithm to proceed as usual and the total space of the snapshot is $O(h)$. For completeness, we provide the details in the following. Before doing so, we give a remark about the wavefront merging procedure. The preceding discusses our changes to the wavefront propagation procedure. For the wavefront merging procedure, which is to construct $W(e)$ from $W(f, e)$ for the edges $f \in input(e)$, notice that we do not need the old versions of $W(f, e)$ anymore after $W(e)$ is constructed. Therefore, it is not necessary to use the path-copying method to process each event in the procedure. Hence, the total space needed in the wavefront merging procedure in the entire algorithm is $O(n)$.

The following lemma shows that if we discard all persistent trees currently used in the algorithm and instead store the snapshot, then the algorithm will proceed without any issues. The proof is literally the same as that in prior work [48] because it only relies on the high-level algorithm scheme not the implementation details.

Lemma 3.25. *The snapshot stores sufficient information for the subsequent algorithm to proceed as usual.*

Proof. Let ξ be the moment of the algorithm when the snapshot construction algorithm starts. Let ξ' be any moment of the algorithm after the snapshot is constructed. We show in the following that if the algorithm needs any information that was computed before ξ to perform certain operation at ξ', then that information is guaranteed to be stored at the snapshot. This will prove the lemma.

At ξ, the transparent edges of $\mathcal{S}'$ excluding e can be classified into two categories: those that have already been processed at ξ and those that have not been processed. Let E_1 and E_2 denote the sets of the edges in these two categories, respectively. The edge e is special in the sense that it is currently being processed at ξ. Hence, it does not belong to either set.

At ξ', depending on whether the algorithm is in the wavefront merging procedure or in the wavefront propagation procedure, there are two cases, which are presented next.

The Algorithm Is in the Wavefront Merging Procedure. Suppose the algorithm is in the wavefront merging procedure at ξ', which is to compute $W(e')$ for some edge e' by merging all wavefronts $W(f', e')$ for $f' \in input(e')$. Because at ξ' we need some information that was computed before ξ, that information must be the wavefront $W(f', e')$ for some edge $f' \in input(e')$. Depending on whether $f' = e$, there are two subcases:

- If $f' = e$, then since $W(e, e')$ was computed before ξ, $W(e, e')$ is stored in the snapshot by Step (2) of the snapshot construction algorithm.
- If $f' \neq e$, then since $W(f', e')$ was computed before ξ, the edge f' must have been processed at ξ. Because the algorithm is computing $W(e')$ at ξ', e' has not been processed at ξ. Therefore, $W(f', e')$ is stored in the snapshot by Step (4) of the snapshot construction algorithm.

Hence, in either subcase, the information needed by the algorithm at ξ' is stored at the snapshot.

The Algorithm Is in the Wavefront Propagation Procedure. Suppose the algorithm is in the wavefront propagation procedure at ξ', which is to process a transparent edge e'—that is, compute $W(e', g')$ for some $g' \in output(e')$. Because at ξ' the algorithm needs some information that was computed before ξ to compute $W(e', g')$, $W(e')$ must have been computed before ξ (since $W(e', g')$ relies on $W(e')$).

We claim that e' must be e. Indeed, if $e' \in E_1$, then e' has been processed before ξ and thus the wavefront propagation procedure cannot happen to e' at ξ', a contradiction. If $e' \in E_2$, then e' has not been processed at ξ. According to our algorithm, for any transparent edge e'', $W(e'')$ is computed during the wavefront merging procedure for e'', which is immediately followed by the wavefront propagation procedure to process e''. Since at ξ the algorithm is in the wavefront propagation procedure to process e, the wavefront merging procedure for e' must have not been invoked at ξ, and thus $W(e')$ must have not been computed at ξ. This contradicts with the fact that $W(e')$ has been computed at ξ. Therefore, e' must be e.

Depending on whether g' is g, there are two subcases:

- If $g' = g$, then the tree at the moment ξ during the propagation for computing $W(e, g)$ from $W(e)$ is stored in the snapshot by Step (1) of the snapshot construction algorithm, and thus we can use the tree to "resume" computing $W(e, g)$ at ξ'.
- If $g' \neq g$, then to compute $W(e, g')$ at ξ', we need the wavefront $W(e)$, which is stored in the snapshot by Step (3) of the snapshot construction algorithm.

Hence, in either subcase, the information needed by the algorithm at ξ' is stored at the snapshot. The lemma thus follows. □

Bounding the space of the snapshot is more challenging, which is established in the following lemma. As the proof is lengthy and technical, we devote the next section to it.

Lemma 3.26. *The total space of the snapshot is $O(h)$.*

Since each phase has no more than $h/\log h$ wavefront propagation operations, the total extra space introduced by the persistent trees in each phase is $O(h)$. Due to the space-reset and Lemma 3.26, the total space of the algorithm is $O(n)$.

For the running time of our new algorithm, comparing with the original HS algorithm, we spend extra time on constructing the snapshot at the end of each phase. In light of Lemma 3.26, as $\mathcal{S}'$ has $O(h)$ transparent edges, each call of the snapshot construction algorithm takes $O(h)$ time. As there are $O(\log h)$ phases, the total time on constructing the snapshots in the entire algorithm is $O(h \log h)$. Hence, the running time of our new algorithm is still bounded by $O(n + h \log h)$.

3.9.2 Proof of Lemma 3.26. We now prove Lemma 3.26—that is, the space introduced by the four steps of the snapshot construction algorithm is $O(h)$.

For the first step, as each wavefront has $O(h)$ generators (and representing each generator takes $O(1)$ space), the size of the tree representing the wavefront is $O(h)$. For the second step, since $|output(e)| = O(1)$ and the size of each $W(e, g')$ is $O(h)$, the total space is $O(h)$. For the third step, since $|W(e)| = O(h)$, the space is $O(h)$. In the following, we focus on the fourth step.

Let Π denote the collection of pairs (e, g) whose wavefront $W(e, g)$ is stored in the fourth step of the snapshot construction algorithm. Our goal is to show that $\sum_{(e,g)\in\Pi} |W(e, g)| = O(h)$.

For an edge e, Π may have multiple pairs with e as the first element and the second elements of all these pairs are in $output(e)$; among all those pairs, we only keep the pair (e, g) such that $|W(e, g)|$ is the largest in Π and remove all other pairs from Π. Since $|output(e)| = O(1)$, it suffices to show that the total sum of $|W(e, g)|$ for all pairs (e, g) in the new Π is $O(h)$. Now in the new Π, no

two pairs have the same first element. However, for an edge g, it is possible that there are multiple pairs in Π whose second elements are all g and their first elements are all from $input(g)$; among all those pairs, we only keep the pair (e, g) such that $|W(e, g)|$ is the largest in Π and remove all other pairs from Π. Since $|input(g)| = O(1)$, it suffices to show that the total sum of $|W(e, g)|$ for all pairs (e, g) in the new Π is $O(h)$. Now in the new Π, no two pairs have the same first element and no two pairs have the same second element. As $\mathcal{S}'$ has $O(h)$ transparent edges, the size of Π is $O(h)$.

For each pair $(e, g) \in \Pi$, recall that $W(e, g)$ may have up to two artificial generators (i.e., the endpoints of e). Since $|\Pi| = O(h)$, there are a total of $O(h)$ artificial generators in $W(e, g)$ for all pairs $(e, g) \in \Pi$. In the following discussion, we ignore these artificial generators, and by slightly abusing the notation, we use $W(e, g)$ to refer to the remaining wavefront without the artificial generators. Hence, each generator of $W(e, g)$ is on an elementary chain. It suffices to prove $\sum_{(e,g)\in\Pi} |W(e, g)| = O(h)$.

For any three adjacent generators $\alpha_1 = (A_1, a_1)$, $\alpha_2 = (A_2, a_2)$, and $\alpha_3 = (A_3, a_3)$ in a wavefront, we call $(\alpha_1, \alpha_2, \alpha_3)$ an *adjacent-generator-triple*. For two generators $\alpha = (A, a)$ and $\alpha' = (A', a')$, we say that $\alpha \neq \alpha'$ if $A \neq A'$, $a \neq a'$, or the designated directions of A and A' are not the same in the case $A = A'$ and $a = a'$. Two adjacent-generator-triples $(\alpha_1, \alpha_2, \alpha_3)$ and $(\alpha'_1, \alpha'_2, \alpha'_3)$ are *distinct* if $\alpha_i \neq \alpha'_i$ from some $i \in \{1, 2, 3\}$. We have the following observation.

Observation 2. *The number of distinct adjacent-generator-triples in all wavefronts involved in the entire wavefront expansion step of the algorithm is $O(h)$.*

Proof. Initially, the algorithm starts a wavefront from s with only one generator s. Whenever a distinct adjacent-generator-triple is produced during the algorithm, either a bisector event happens or a new generator is created. In either case, a generator will be marked according to our generator marking rules. As the total number of marked generators during the algorithm is $O(h)$ by Lemma 3.15, the observation follows. □

Lemma 3.26 is almost an immediate consequence of the following lemma.

Lemma 3.27. *Any adjacent-generator-triple $(\alpha_1, \alpha_2, \alpha_3)$ can appear in the wavefront $W(e, g)$ for at most $O(1)$ pairs (e, g) of Π.*

Before proving Lemma 3.27, we prove Lemma 3.26 with the help of Lemma 3.27. Indeed, there are $O(h)$ distinct adjacent-generator-triples in the entire algorithm by Observation 2. Now that each such triple can only appear in a constant number of wavefronts $W(e, g)$ for all $(e, g) \in \Pi$, the total number of generators in all wavefronts $W(e, g)$ for all $(e, g) \in \Pi$ is bounded by $O(h)$. This leads to Lemma 3.26.

Proving Lemma 3.27. We prove Lemma 3.27 in the rest of this section. Assume to the contrary that an adjacent-generator-triple $(\alpha_1, \alpha_2, \alpha_3)$ appears in the wavefront $W(e, g)$ for more than $O(1)$ pairs (e, g) of Π;[7] let Π' denote the set of all such pairs.

Consider a pair $(e, g) \in \Pi'$. Recall that $W(e, g)$ is obtained by propagating $W(e)$ from e to g through the well-covering region $\mathcal{U}(g)$ if $e \in input(g)$ and through $\mathcal{U}(e)$ otherwise. If one of the generator initial vertices a_1, a_2, and a_3 is in $\mathcal{U}(g) \cup \mathcal{U}(e)$, then we call (e, g) a *special pair*. The properties of the subdivision $\mathcal{S}'$ guarantee that each cell of $\mathcal{S}'$ is in $\mathcal{U}(f)$ for at most $O(1)$ transparent edges f of $\mathcal{S}'$, implying that each vertex a_i, $1 \leq i \leq 3$, is in $\mathcal{U}(f)$ for at most $O(1)$ transparent edges f of $\mathcal{S}'$, and thus Π' has at most $O(1)$ special pairs. We remove all special pairs from Π', after which Π' still has more than $O(1)$ pairs and for each pair $(e, g) \in \Pi'$ the initial

[7] By "more than $O(1)$ pairs," we intend to say "more than c pairs for a constant c to be fixed later." To simplify the discussion, we use "more than $O(1)$ pairs" instead with the understanding that such a constant c can be fixed.

vertices of the three generators α_1, α_2, and α_3 are all outside $\mathcal{U}(g) \cup \mathcal{U}(e)$. Therefore, α_1, α_2, and α_3 are also generators in $W(e)$ in this order (although they may not be adjacent in $W(e)$). Because $(\alpha_1, \alpha_2, \alpha_3)$ is an adjacent-generator-triple in $W(e, g)$, $B(\alpha_1, \alpha_2)$ (respectively, $B(\alpha_2, \alpha_3)$) intersects g. In addition, note that α_1, α_2, and α_3 are on the same side of the supporting line of e; since they are generators of both $W(e)$ and $W(e, g)$, the bisector $B(\alpha_1, \alpha_2)$ (respectively, $B(\alpha_2, \alpha_3)$) intersects e (although the intersection may not appear in $W(e)$, i.e., the intersection may be claimed by a different generator in $W(e)$). Further, α_2 is closer to the intersection of $B(\alpha_1, \alpha_2)$ (respectively, $B(\alpha_2, \alpha_3)$) with e than with g. Let (e_1, g_1) be the pair in Π' such that the intersection of $B(\alpha_1, \alpha_2)$ with its first element is closest to α_2 among the intersections of $B(\alpha_1, \alpha_2)$ with the first elements of all pairs of Π'.

By the property of $\mathcal{S}'$, $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$ contains $O(1)$ transparent edges. Because $|\Pi'|$ is not $O(1)$, Π' must have a pair, denoted by (e_2, g_2), such that e_2 is outside $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$. Recall that $W(e_1, g_1)$ is obtained by propagating $W(e_1)$ from e_1 to g_1 inside $\mathcal{U}(g_1)$ or $\mathcal{U}(e_1)$. Hence, if we move along the bisector $B(\alpha_1, \alpha_2)$ from its intersection with e_1 to its intersection with g_1, all encountered edges of $\mathcal{S}'$ are in $\mathcal{U}(g_1) \cup \mathcal{U}(e_1)$. Therefore, by the definitions of e_1 and e_2, $B(\alpha_1, \alpha_2)$ intersects e_1, g_1, e_2, and g_2 in this order following their distances from α_2 (Figure 30).

By definition (i.e., the fourth step of our snapshot construction algorithm), e_2 has been processed but g_1 has not. According to the wavefront expansion algorithm, $covertime(e_2) \leq covertime(g_1)$. In the following, we will obtain $covertime(e_2) > covertime(g_1)$, which leads to a contradiction.

Let b be the intersection between the bisector $B(\alpha_1, \alpha_2)$ and g_2 (e.g., see Figure 30). Let $\pi'(b, a_2)$ be the path from b to a_2 following the tangent from b to A_2 and then following A_2 to a_2 (e.g., the dashed segment in Figure 30). Since α_1 and α_2 are two adjacent generators in $W(e_2, g_2)$, $\pi'(b, a_2)$ is in the free space $\mathcal{F}$. As $W(e_2, g_2)$ is obtained from $W(e_2)$ and α_2 is also a generator in $W(e_2)$, $\pi'(b, a_2)$ must intersect e_2, say, at a point q (e.g., see Figure 30). Further, we have the following observation.

Observation 3. *$\pi'(b, a_2)$ intersects g_1 (e.g., see Figure 30).*

Proof. Since $(\alpha_1, \alpha_2, \alpha_3)$ is an adjacent-generator-triple in both wavefronts $W(e_1, g_1)$ and $W(e_2, g_2)$, each of the bisectors $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$ intersects both g_1 and g_2. Let z_1 and z_2 be the intersections of g_1 with $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$, respectively. Let b' be the intersection of g_2 and $B(\alpha_2, \alpha_3)$. Then, $\overline{z_1 z_2}$, which is a subsegment of g_1, is claimed by α_2 in $W(e_1, g_1)$. Similarly, $\overline{bb'}$, which is a subsegment of g_2, is claimed by α_2 in $W(e_2, g_2)$. Recall that α_2 is closer to z_1 than to b. Hence, for any point $p \in \overline{bb'}$, $\pi'(p, \alpha_2)$ must intersect $\overline{z_1 z_2}$, where $\pi(p, \alpha_2)$ is defined in a way similar to $\pi'(b, \alpha_2)$. This leads to the observation, for $\overline{z_1 z_2} \subseteq g_1$ and $b \in \overline{bb'}$. □

In light of the preceding observation, let z be an intersection point of $\pi'(b, a_2)$ and g_1 (e.g., see Figure 30). For any two points p and p' of $\pi'(b, a_2)$, we use $\pi'(p, p')$ to refer to the subpath of $\pi'(b, a_2)$ between p and p'. Using the properties of the well-covering regions of $\mathcal{S}'$, we have the following observation.

Observation 4. $|\pi'(q, z)| \geq 2 \cdot |g_1|$.

Proof. Since $q \in e_2$ is outside the well-covering region $\mathcal{U}(g_1)$ of g_1, $z \in g_1$, and $\pi'(q, z)$ is in the free space $\mathcal{F}$, $\pi'(q, z)$ must cross the boundary of $\mathcal{U}(g_1)$, say, at a point p. By the property of the conforming subdivision $\mathcal{S}'$, $|\pi'(p, z)| \geq 2 \cdot |g_1|$. As $|\pi'(q, z)| \geq |\pi'(p, z)|$, the observation follows. □

The following lemma, which is a consequence of Observation 4, leads to a contradiction, for $covertime(g_1) \geq covertime(e_2)$, and thus Lemma 3.27 is proved.

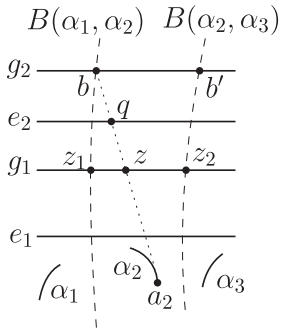

Fig. 30. Illustrating the two bisectors $B(\alpha_1, \alpha_2)$ and $B(\alpha_2, \alpha_3)$ as well as the four transparent edges e_1, g_1, e_2, and g_2 (each of these edges may also be vertical).

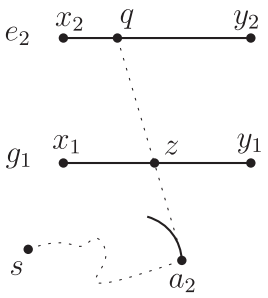

Fig. 31. Illustrating the proof of Lemma 3.28.

Lemma 3.28. $covertime(g_1) < covertime(e_2)$.

Proof. Let x_1 and y_1 be the two endpoints of g_1, respectively. Let x_2 and y_2 be the two endpoints of e_2, respectively. Refer to Figure 31. As $\pi'(z, a_2) \subseteq \pi'(b, a_2)$ is in the free space $\mathcal{F}$, we have the following for $covertime(g_1)$:

$$\begin{aligned} covertime(g_1) &= \min\{d(s, x_1), d(s, y_1)\} + |g_1| \\ &\leq d(s, a_2) + |\pi'(a_2, z)| + |g_1|/2 + |g_1| \\ &= d(s, a_2) + |\pi'(a_2, z)| + 3|g_1|/2. \end{aligned}$$

However, because α_2 claims b in $W(e_2, g_2)$ (as explained in the proof of Observation 3), α_2 is also a generator in $W(e_2)$, and $\pi'(b, a_2)$ intersects e_2 at q, α_2 must claim q in $W(e_2)$. Hence, $\min\{d(s, x_2), d(s, y_2)\} + |e_2| \geq d(s, a_2) + |\pi'(a_2, q)|$ must hold, since otherwise the artificial wavelets at the endpoints of e_2 would have prevented α_2 from claiming q. Therefore, we have the following for $covertime(e_2)$:

$$\begin{aligned} covertime(e_2) &= \min\{d(s, x_2), d(s, y_2)\} + |e_2| \\ &\geq d(s, a_2) + |\pi'(a_2, q)| \\ &= d(s, a_2) + |\pi'(a_2, z)| + |\pi'(z, q)| \\ &\geq d(s, a_2) + |\pi'(a_2, z)| + 2 \cdot |g_1|. \end{aligned}$$

The last inequality is due to Observation 4. As $|g_1| > 0$, we obtain $covertime(g_1) < covertime(e_2)$. □

3.9.3 Reducing the Space of Constructing SPM′(s). For the second issue of constructing $SPM'(s)$, our original algorithm relies on the wavefronts $W(e)$ for all transparent edges e, which are maintained by persistent trees. Due to the space-reset, our new algorithm does not maintain the wavefronts anymore, and thus we need to somehow restore these wavefronts to construct $SPM'(s)$. To this end, a key observation is that by marking a total of $O(h)$ additional wavelet generators, it is possible to restore all historical wavefronts that are needed for constructing $SPM'(s)$. In this way, $SPM'(s)$ can be constructed in $O(n + h \log h)$ time and $O(n)$ space.

More specifically, our algorithm considers each cell c of $\mathcal{S}'$ individually. For each cell c, the algorithm has two steps. First, compute the active regions of c. Second, for each active region R, compute the vertices of $SPM'(s)$ in R. For both steps, our algorithm utilizes the wavefronts $W(e)$

of the transparent edges e on the boundary of c. Due to the space-reset, the wavefronts $W(e)$ are not available anymore in our new algorithm. We use the following strategy to resolve the issue.

First, to compute the active regions in c, we need to know the bisectors $B(\alpha, \alpha')$ defined by an unmarked generator α and a marked generator α' in the wavefronts $W(e)$ of the transparent edges e of c. Consider such a bisector $B(\alpha, \alpha')$. Observe that α and α' must be from the same wavefront $W(e)$ of a transparent edge e of c. Indeed, assume to the contrary that α is from $W(e_1)$ and α' is from $W(e_2)$ of two different transparent edges e_1 and e_2 of c. Then, because α and α' are adjacent generators along the boundary of c, the wavelet of α in $W(e_1)$ must claim an endpoint of e_1 and the wavelet of α' in $W(e_2)$ must claim an endpoint of e_2. Hence, both α and α' are marked by Rule 2(a) of the generator marking rules. But this incurs a contradiction, as α is unmarked. Therefore, α and α' must be from the same wavefront $W(e)$ and they are actually neighboring in $W(e)$.

Based on the preceding observation, we slightly modify our wavefront expansion algorithm as follows. In addition to our original marking rules, we add the following new rule: If a generator α' in a wavefront $W(e)$ is marked for a cell by the original rules during the algorithm, we also mark both neighboring generators of α' in $W(e)$. We call the generators marked by the new rule *newly marked* generators, whereas the generators marked by the original rules are called *originally marked* generators. If a generator is both newly marked and originally marked, we consider it as originally marked. As each generator has two neighbors in a wavefront, the total number of marked generators is still $O(h)$. The newly marked generators and the originally marked generators are sufficient for computing all active regions of each cell c as before. Indeed, the active regions are decomposition of c by the bisectors of adjacent generators with one originally marked and the other newly marked in $W(e)$ of the transparent edges e of c.

Second, to compute the vertices of $SPM'(s)$ in each active region R of c, we need to restore the wavefronts $W(e)$ of the transparent edges e on the boundary of R. To this end, the algorithm in the preceding first step determines a list $L(e)$ of originally marked generators that claim e. It turns out that $L(e)$ is exactly $W(e)$, as shown next.

Observation 5. $L(e) = W(e)$.

Proof. Since each generator of $L(e)$ is marked for e and c, $L(e)$ is a subset of $W(e)$. However, because R is an active region, R is claimed by originally marked generators only. Since each generator of $W(e)$ claims at least one point of e and $e \subseteq R$, all generators are originally marked. Therefore, all generators of $W(e)$ are in $L(e)$. The observation thus follows. □

In light of the preceding observation, the wavefronts $W(e)$ for all transparent edges e of all active regions R of c can be restored once all active regions of c are computed in the first step. Subsequently, the same algorithm as before can be applied to compute the vertices of $SPM'(s)$ in R. When computing the partition $Vor(e)$ of R, we propagate $W(e)$ using the wavefront propagation procedure, which uses persistent trees to represent $W(e)$. Since here we do not need to keep the old versions of $W(e)$ anymore, we can use an ordinary balanced binary search tree (without the path-copying method) to represent $W(e)$. In this way, processing each bisector event only introduces $O(1)$ additional space.

In summary, $SPM'(s)$ can now be constructed in $O(n + h \log h)$ time and $O(n)$ space.

4 THE GENERAL CASE

In this section, we extend our algorithm for the convex case in Section 3 to the general case where obstacles of $\mathcal{P}$ may not be convex. To this end, we resort to an extended corridor structure of $\mathcal{P}$, which has been used to solve various problems in polygonal domains [4, 7–9, 29–31, 39]. The structure decomposes the free space $\mathcal{F}$ into three types of regions: an *ocean* $\mathcal{M}$, $O(h)$ *canals*, and $O(n)$ *bays*. The details are given in the following.

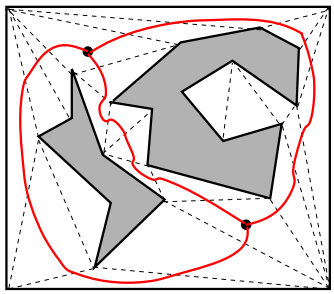

Fig. 32. Illustrating a triangulation of $\mathcal{P}$ with two obstacles. There are two junction triangles indicated by the large dots inside them, connected by three solid (red) curves. Removing the two junction triangles results in three corridors.

4.1 The Extended Corridor Structure

Let $Tri(\mathcal{P})$ denote an arbitrary triangulation of $\mathcal{P}$. Let $G(\mathcal{P})$ be the (planar) dual graph of $Tri(\mathcal{P})$—that is, each node of $G(\mathcal{F})$ corresponds to a triangle in $Tri(\mathcal{P})$ and each edge connects two nodes of $G(\mathcal{P})$ corresponding to two triangles sharing a triangulation diagonal of $Tri(\mathcal{P})$. We compute a *corridor graph* G as follows. First, repeatedly remove every degree-1 node from $G(\mathcal{P})$ until no such node remains. Second, repeatedly remove every degree-2 node from $G(\mathcal{P})$ and replace its two incident edges by a single edge until no such node remains. The resulting graph is G (Figure 32), which has $O(h)$ faces, nodes, and edges [31]. Each node of G corresponds to a triangle in $Tri(\mathcal{P})$, which is called a *junction triangle* (e.g., see Figure 32). The removal of all junction triangles results in $O(h)$ *corridors* (defined in the following), and each corridor corresponds to an edge of G.

The boundary of a corridor C consists of four parts (Figure 33): (1) a boundary portion of an obstacle, from an obstacle vertex a to an obstacle vertex b; (2) a triangulation diagonal of a junction triangle from b to an obstacle vertex e; (3) a boundary portion of an obstacle from e to an obstacle vertex f; and (4) a diagonal of a junction triangle from f to a. The two diagonals $\overline{be}$ and $\overline{af}$ are called the *doors* of C, and the other two parts of the boundary of C are the two *sides* of C. Note that C is a simple polygon. A point is in the *interior* of C if it is in C excluding the two doors. Let $\pi_C(a, b)$ (respectively, $\pi_C(e, f)$) be the shortest path from a to b (respectively, e to f) in C. The region H_C bounded by $\pi_C(a, b)$, $\pi_C(e, f)$, $\overline{be}$, and $\overline{fa}$ is called an *hourglass*, which is *open* if $\pi_C(a, b) \cap \pi_C(e, f) = \emptyset$ and *closed* otherwise (see Figure 33). If H_C is open, then both $\pi_C(a, b)$ and $\pi_C(e, f)$ are convex chains and called the *sides* of H_C; otherwise, H_C consists of two "funnels" and a path $\pi_C = \pi_C(a, b) \cap \pi_C(e, f)$ joining the two apices of the funnels, called the *corridor path* of C. Each side of a funnel is also convex.

Let $\mathcal{M}$ be the union of all $O(h)$ junction triangles, open hourglasses, and funnels. We call $\mathcal{M}$ the *ocean*, whose boundary $\partial\mathcal{M}$ consists of $O(h)$ convex chains that are sides of open hourglasses and funnels. The other space of $\mathcal{P}$ (i.e., $\mathcal{P} \setminus \mathcal{M}$) is further partitioned into two types of regions—*bays* and *canals*—defined as follows. Consider the hourglass H_C of a corridor C.

If H_C is open (see Figure 33), then H_C has two sides. Let S_1 be a side of H_C. The obstacle vertices on S_1 all lie on the same side of C. Let c and d be any two consecutive vertices on S_1 such that $\overline{cd}$ is not an obstacle edge of $\mathcal{P}$ (e.g., see Figure 33, left). The region enclosed by $\overline{cd}$ and the boundary portion of C between c and d is called a *bay*, denoted by $bay(\overline{cd})$. We call $\overline{cd}$ the *gate* of $bay(\overline{cd})$.

If H_C is closed, let x and y be the two apices of the two funnels. Consider two consecutive vertices c and d on a side of a funnel such that $\overline{cd}$ is not an obstacle edge of $\mathcal{P}$. If c and d are on the same side of the corridor C, then $\overline{cd}$ also defines a bay. Otherwise, either c or d is a funnel apex, say, $c = x$, and we call $\overline{xd}$ a *canal gate* at $x = c$ (e.g., see Figure 33, right). Similarly, there is also a

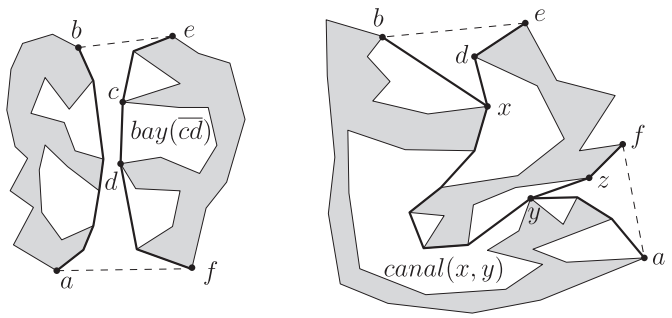

Fig. 33. Illustrating an open hourglass (left) and a closed one (right) with a corridor path connecting the apices x and y of the two funnels. The dashed segments are diagonals. The paths $\pi_C(a, b)$ and $\pi_C(e, f)$ are marked by thick solid curves. A bay $bay(\overline{cd})$ with gate $\overline{cd}$ (left) and a canal $canal(x, y)$ with gates $\overline{xd}$ and $\overline{yz}$ (right) are also shown.

canal gate at the other funnel apex y, say $\overline{yz}$. The region of C between the two canal gates $\overline{xd}$ and $\overline{yz}$ is the *canal* of H_C, denoted by $canal(x, y)$.

Each bay or canal is a simple polygon. All bays and canals together constitute the space $\mathcal{P} \setminus \mathcal{M}$. Each vertex of $\partial\mathcal{M}$ is a vertex of $\mathcal{P}$ and each edge of $\partial\mathcal{M}$ is either an edge of $\mathcal{P}$ or a gate of a bay/canal. Gates are common boundaries between $\mathcal{M}$ and bays/canals. After $\mathcal{P}$ is triangulated, $\mathcal{M}$ and all bays and canals can be obtained in $O(n)$ time [31].

The reason that the extended corridor structure can help find a shortest path is the following. Suppose we want to find a shortest s-t path for two points s and t. We consider s and t as two special obstacles and build the extended corridor structure. If a shortest s-t path $\pi(s, t)$ contains a point in the interior of a corridor C, then $\pi(s, t)$ must cross both doors of C and stay in the hourglasses of C, and further, if the hourglass is closed, then its corridor path must be contained in $\pi(s, t)$. In fact, $\pi(s, t)$ must be in the union of the ocean $\mathcal{M}$ and all corridor paths [31].

In light of the preceding properties, we propose the following algorithm. Let s be a given source point. By considering s as a special obstacle of $\mathcal{P}$, we construct the extended corridor structure of $\mathcal{P}$. Consider any query point t, which may be in the ocean $\mathcal{M}$, a bay $bay(\overline{cd})$, or a canal $canal(x, y)$:

- If $t \in \mathcal{M}$, then the union of $\mathcal{M}$ and all corridor paths contains a shortest s-t path. To handle this case, we will build a shortest path map $SPM(\mathcal{M})$ in $\mathcal{M}$ with respect to the union of $\mathcal{M}$ and all corridor paths. In fact, $SPM(\mathcal{M})$ is exactly the portion of $SPM(s)$ in $\mathcal{M}$ (i.e., $SPM(s) \cap \mathcal{M}$). To build $SPM(\mathcal{M})$, a key observation is that the boundary $\partial\mathcal{M}$ consists of $O(h)$ convex chains. Therefore, we can easily adapt our previous algorithm for the convex case. However, the algorithm needs to be modified so that the corridor paths should be taken into consideration. Intuitively, corridor paths provide certain kind of "shortcuts" for wavefronts to propagate.
- If t is in a bay $bay(\overline{cd})$, then any shortest s-t path must cross its gate $\overline{cd}$. To handle this case, we will extend $SPM(\mathcal{M})$ into $bay(\overline{cd})$ through the gate $\overline{cd}$ to construct the shortest path map in $bay(\overline{cd})$—that is, the portion of $SPM(s)$ in $bay(\overline{cd})$, $SPM(s) \cap bay(\overline{cd})$.
- If t is in a canal $canal(x, y)$, then any shortest s-t path must cross one of the two gates of the canal. To handle this case, we will extend $SPM(\mathcal{M})$ into $canal(x, y)$ through the two gates to construct the shortest path map in $canal(x, y)$—that is, the portion of $SPM(s)$ in $canal(x, y)$, $SPM(s) \cap canal(x, y)$.

In the following, we first describe our algorithm for constructing $SPM(\mathcal{M})$ in Section 4.2. We then expand $SPM(\mathcal{M})$ into all bays in Section 4.3 and expand $SPM(\mathcal{M})$ into all canals in Section 4.4. The algorithm for the canal case utilizes the bay case algorithm as a subroutine.

4.2 Constructing the Shortest Path Map in the Ocean $SPM(\mathcal{M})$

As the boundary of $\mathcal{M}$ consists of $O(h)$ convex chains, we can apply and slightly modify our algorithm for the convex case. To do so, for each convex chain of $\partial\mathcal{M}$, we define its rectilinear extreme vertices in the same way as before. Let $\mathcal{V}$ be the set of the rectilinear extreme vertices of all convex chains. Hence, $|\mathcal{V}| = O(h)$. In addition, to incorporate the corridor paths into the algorithm, we include the endpoints of each corridor path in $\mathcal{V}$. As there are $O(h)$ corridor paths, the size of $\mathcal{V}$ is still bounded by $O(h)$. Note that each corridor path endpoint is also an endpoint of a convex chain of $\partial\mathcal{M}$. We construct the conforming subdivision $\mathcal{S}$ based on the points of $\mathcal{V}$ and then insert the convex chains of $\mathcal{M}$ into $\mathcal{S}$ to obtain $\mathcal{S}'$. The algorithm is essentially the same as before. In addition, we make the following changes to $\mathcal{S}'$, which is mainly for incorporating the corridor paths into our wavefront expansion algorithm, as will be clear later.

Let v be an endpoint of a corridor path π. Since v is in $\mathcal{V}$, v is incident to $O(1)$ transparent edges in $\mathcal{S}'$. For each such transparent edge e, if $|\pi| < 2 \cdot |e|$, then we divide e into two subedges such that the length of the one incident to v is equal to $|\pi|/2$; for each subedge, we set its well-covering region the same as $\mathcal{U}(e)$. Note that this does not affect the properties of $\mathcal{S}'$. In particular, each transparent edge e is still well-covered. This change guarantees the following property: for each corridor path π, $|\pi| \geq 2 \cdot |e'|$ holds, where e' is any transparent edge of $\mathcal{S}'$ incident to either endpoint of π. For reference purposes, we refer to it as the *corridor path length property*.

Next we apply the wavefront expansion algorithm. Here we need to incorporate the corridor paths into the algorithm. Intuitively, each corridor path provides a "shortcut" for the wavefront—that is, if a wavelet hits an endpoint of a corridor path, then the wavelet will come out of the corridor path from its other endpoint but with a delay of distance equal to the length of the corridor path. More details are given in the following.

Since all corridor path endpoints are in $\mathcal{V}$, they are vertices of transparent edges of $\mathcal{S}'$. Consider an endpoint v of a corridor path π. Let u be the other endpoint of π. Recall that the wavefront propagation procedure for $W(e)$ is to propagate $W(e)$ to compute $W(e, g)$ for all edges $g \in output(e)$. In addition to the previous algorithm for the procedure, we also propagate $W(e)$ through the corridor path π to u. This is done as follows. Recall that when e is processed, since v is an endpoint of e, the weighted distance of v through $W(e)$ is equal to $d(s, v)$. Hence, the wavefront $W(e)$ hits u through π at time $d(s, v) + |\pi|$. We then update $covertime(g) = \min\{covertime(g), d(s, v) + |\pi| + |g|\}$, for each transparent edge g incident to u. We also set the wavefront $W(e, g)$ consisting of the only wavelet with u as the generator with weight equal to $d(s, v) + |\pi|$. Since there are $O(1)$ transparent edges g incident to u, the preceding additional step takes $O(1)$ time, which does not change the time complexity of the overall algorithm asymptotically. The corridor path length property assures that if $W(e)$ contributes to a wavefront $W(g)$ at g, then e must be processed earlier than g. This guarantees the correctness of the algorithm.

In this way, we can first construct a decomposition $SPM'(\mathcal{M})$ of $\mathcal{M}$ in $O(n+h \log h)$ time and $O(n)$ space, where $SPM'(\mathcal{M})$ is defined similarly as $SPM'(s)$ in Section 3. Then, by a similar algorithm as that for Lemma 3.6, $SPM(\mathcal{M})$ can be obtained in additional $O(n)$ time.

4.3 Expanding $SPM(\mathcal{M})$ into All Bays

We now expand $SPM(\mathcal{M})$ into all bays in $O(n + h \log h)$ time and $O(n)$ space. In fact, we expand $SPM'(\mathcal{M})$ to the bays. We process each bay individually. Consider a bay $bay(\overline{cd})$ with gate $\overline{cd}$. Without loss of generality, we assume that $\overline{cd}$ is horizontal, c is to the left of d, and $bay(\overline{cd})$ is locally above $\overline{cd}$.

Let $v_1, v_2, \ldots, v_m$ be the vertices of $SPM'(\mathcal{M})$ on $\overline{cd}$ ordered from left to right (Figure 34). Let $c = v_0$ and $d = v_{m+1}$. Hence, each $\overline{v_i v_{i+1}}$ is claimed by a generator $\alpha_i = (A_i, a_i)$ for all $i = 0, 1, \ldots, m$.

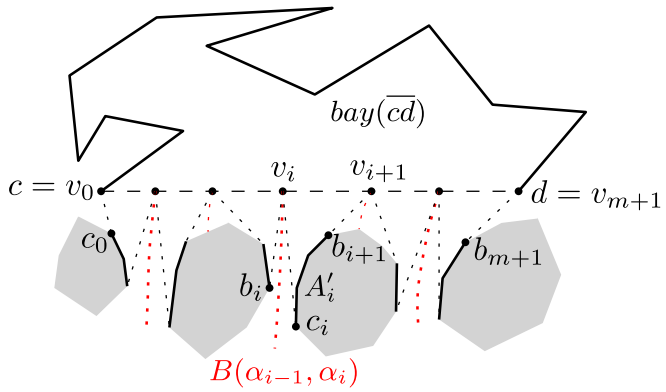

Fig. 34. Illustrating $bay(\overline{cd})$ and the generators. The thick segments on obstacles are A'_i, $i = 0, 1, \ldots, m$.

Let b_i and c_i be the tangent points on A_{i-1} and A_i from v_i, respectively, for each $i = 1, 2, \ldots, m$ (e.g., see Figure 34). For v_0, only c_0 is defined; for v_{m+1}, only b_{m+1} is defined. Observe that for any point $p \in \overline{v_i v_{i+1}}$, which is claimed by α_i, its tangent point on A_i must be on the portion of A_i between c_i and b_{i+1} and we use A'_i to denote that portion. So with respect to $bay(\overline{cd})$, we use $\alpha'_i = (A'_i, a'_i)$ to refer to the generator, where a'_i refers to the one of c_i and b_{i+1} that is closer to a_i. Hence, for any point $t \in bay(\overline{cd})$, any shortest path $\pi(s, t)$ from s to t must be via one of the generators α'_i for $i = 0, 1, \ldots, m$. Consider the region R bounded by A'_i for all $i \in [0, m]$, the tangents from v_i to their generators for all $i \in [0, m+1]$, and the boundary of the bay excluding its gate. Notice that R is a simple polygon. For any point $t \in bay(\overline{cd})$, the preceding observation implies any shortest s-t path $\pi(s, t)$ is the concatenation of a shortest path from s to a generator initial vertex a'_i and the shortest path from a'_i to t in R.

According to the preceding discussion, expanding $SPM'(\mathcal{M})$ into $bay(\overline{cd})$ is equivalent to the following weighted geodesic Voronoi diagram problem in a simple polygon: Partition R into cells with respect to the point sites $a'_0, a'_1, \ldots, a'_m$ (with weights equal to their geodesic distances to s) such that all points in the same cell have the same closest site. Let n_b be the number of vertices in $bay(\overline{cd})$ (the subscript "b" represents "bay"). Let n_g be the total number of obstacle vertices in A'_i for all $i \in [0, m]$ (the subscript "g" represents "generator"). Note that v_i for all $i = 1, \ldots, m$ are also vertices of R. Hence, the number of vertices of R is $n_b + n_g + m$. The preceding problem can be solved in $O(m \log m + n_b + n_g)$ time by the techniques of Oh [40]. Indeed, given m' point sites in a simple polygon P' of n' vertices, Oh [40] gave an algorithm that can compute the geodesic Voronoi diagram of the sites in P' in $O(n' + m' \log m')$ time and $O(n' + m')$ space. Although the point sites in Oh's problem do not have weights, our problem is essentially an intermediate step of Oh's algorithm because all weighted point sites in our problem are on one side of $\overline{cd}$. Therefore, we can run Oh's algorithm from "the middle" and solve our problem in $O(m \log m + n_b + n_g)$ and $O(n_b + n_g)$ space. In fact, our problem is a special case of Oh's problem because there are no sites in $bay(\overline{cd})$. For this reason, we propose our own algorithm to solve this special case and the algorithm is much simpler than Oh's algorithm; our algorithm also runs in $O(m \log m + n_b + n_g)$ and $O(n_b + n_g + m)$ space. This also makes our work more self-contained.

Before presenting the algorithm, we analyze the total time for processing all bays. Since $SPM'(\mathcal{M})$ has $O(h)$ vertices, the total sum of m for all bays is $O(h)$. The total sum of n_b for all bays is at most n. Notice that the obstacle edges on A'_i are disjoint for different bays, and thus the

total sum of n_g for all bays is $O(n)$. Hence, expanding $SPM'(\mathcal{M})$ to all bays takes $O(n + h \log h)$ time and $O(n)$ space in total.

The preceding actually only considers the case where the gate $\overline{cd}$ contains at least one vertex of $SPM'(\mathcal{M})$. It is possible that no vertex of $SPM'(\mathcal{M})$ is on $\overline{cd}$, in which case the entire gate is claimed by one generator α of $SPM'(\mathcal{M})$. We can still define the region R in the same way. But now, R has only one weighted site and thus the geodesic Voronoi diagram problem becomes computing a shortest path map in the simple polygon R for a single source point. This problem can be solved in $O(n_b + n_g)$ time [20]; note that $m = 0$ in this case. Hence, the total time for processing all bays in this special case is $O(n)$.

4.3.1 Solving the Special Weighted Geodesic Voronoi Diagram Problem. We present an algorithm for the preceding special case of the weighted geodesic Voronoi diagram problem, and the algorithm runs in $O(m \log m + n_b + n_g)$ and $O(n_b + n_g + m)$ space.

If we consider all generators α_i for $i = 0, 1, \ldots, m$ as a wavefront at $\overline{cd}$, denoted by $W(\overline{cd})$, then our algorithm is essentially to propagate $W(\overline{cd})$ inside $bay(\overline{cd})$. To this end, we first triangulate $bay(\overline{cd})$ and will use the triangulation to guide the wavefront propagation. Each time, we propagate the wavefront through a triangle. In the following, we first discuss how to propagate $W(\overline{cd})$ through the triangle $\triangle cda$ with $\overline{cd}$ as an edge and a as the third vertex. This is a somewhat special case, as $\triangle cda$ is the first triangle the wavefront will propagate through, but the general case is quite similar.

Recall that each convex chain A'_i is represented by an array, and the generator list $W(\overline{cd})$ is represented by a balanced binary search tree $T(W(\overline{cd}))$. We build a point location data structure on the triangulation of $bay(\overline{cd})$ in $O(n_b)$ time [16, 32] so that given any query point p, we can determine the triangle that contains p in $O(\log n_b)$ time.

An an initialization step, we compute the intersection of the adjacent bisectors $B(\alpha_{i-1}, \alpha_i)$ and $B(\alpha_i, \alpha_{i+1})$ for all $i = 1, 2, \ldots, m-1$. Each intersection can be computed in $O(\log n_g)$ time by the bisector-bisector intersection operation in Lemma 3.11. Computing all intersections takes $O(m \log n_g)$ time. For each intersection q, called a *bisector event*, we use the point location data structure to find the triangle of the triangulation that contains q and store q in the triangle.

Since all generators are outside $bay(\overline{cd})$, by Corollary 3.8, all bisectors are monotone with respect to the direction orthogonal to $\overline{cd}$. Our algorithm for propagating the wavefront through $\triangle cda$ is based on this property. We sort all bisector events in $\triangle cda$ according to their perpendicular distances to the supporting line of $\overline{cd}$. Then, we process these events in the same way as in our wavefront propagation procedure. Specifically, for each bisector event q of two bisectors $B(\alpha', \alpha)$ and $B(\alpha, \alpha'')$, we process it as follows. Let α'_1 be the other neighboring generator of α' than α, and let α''_1 be the other neighbor of α'' than α. We remove α from the generator list. In addition, we remove the intersection of $B(\alpha', \alpha)$ and $B(\alpha', \alpha'_1)$ from the triangle of $bay(\overline{cd})$ that contains it; we do the same for the intersection of $B(\alpha'', \alpha)$ and $B(\alpha'', \alpha''_1)$. Finally, we compute the intersection q' of $B(\alpha', \alpha'')$ with $B(\alpha'_1, \alpha')$, and we use the point location data structure to find the triangle of $bay(\overline{cd})$ that contains q' and store q' in the triangle. If $q' \in \triangle cda$, then we insert it into the bisector event sorted list of $\triangle cda$. We do the same for $B(\alpha', \alpha'')$ and $B(\alpha'', \alpha''_1)$.

After all events in $\triangle cda$ are processed, we split the current wavefront W at the vertex a. To this end, we first find the generator α^* of W that claims a. For this, we use the same algorithm as before—that is, binary search plus bisector tracing. So we need to maintain a tracing-point for each bisector as before (initially, we can set v_i as the tracing-point for $B(\alpha_{i-1}, \alpha_i)$, $i = 1, 2, \ldots, m$). The correctness of the preceding algorithm for finding the generator α^* relies on the property of the following lemma.

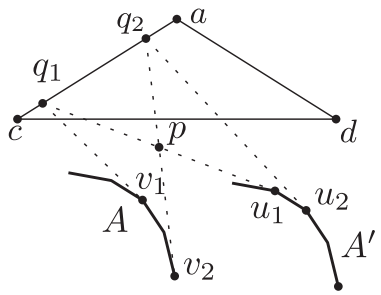

Fig. 35. Illustrating the proof of Lemma 4.1.

Lemma 4.1. *The bisector $B(\alpha, \alpha')$ intersects $\overline{ad} \cup \overline{ac}$ at most once for any two bisectors of α and α' of $W(\overline{cd})$.*

Proof. Note that we cannot apply the result of Lemma 3.7 since to do so we need to have $\overline{ad}$ and $\overline{ac}$ parallel to $\overline{cd}$. But the proof is somewhat similar to that for Lemma 3.7.

Assume to the contrary that $B(\alpha, \alpha')$ intersects $\overline{ad} \cup \overline{ac}$ at two points, q_1 and q_2. Let A and A' be the underlying arcs of α and α', respectively. Let v_1 and u_1 be the tangents points of q_1 on A and A', respectively (Figure 35). Let v_2 and u_2 be the tangents points of q_2 on A and A', respectively. Since both A and A' are on one side of $\overline{cd}$ while $\overline{ad} \cup \overline{ac}$ is on the other side, if we move a point q from q_1 to q_2 on $\overline{ad} \cup \overline{ac}$, the tangent from q to A will continuously change from $\overline{q_1v_1}$ to $\overline{q_2v_2}$ and the tangent from q to A' will continuously change from $\overline{q_1u_1}$ to $\overline{q_2u_2}$. Therefore, either $\overline{q_1u_1}$ intersects $\overline{q_2v_2}$ in their interiors or $\overline{q_1v_1}$ intersects $\overline{q_2u_2}$ in their interiors; without loss of generality, we assume that it is the former case. Let p be the intersection of $\overline{q_1u_1}$ and $\overline{q_2v_2}$ (e.g., see Figure 35). Since $q_1 \in B(\alpha, \alpha')$, points of $\overline{q_1u_1}$ other than q_1 have only one predecessor, which is α'. As $p \in \overline{q_1u_1}$ and $p \neq q_1$, p has only one predecessor α'. Similarly, since $q_2 \in B(\alpha, \alpha')$ and $p \in \overline{q_2v_2}$, α is also p's predecessor. We thus obtain a contradiction. □

After α^* is found, depending on whether α^* is the first or last generator of W, there are three cases:

(1) If α^* is not the first or last generator of W, then we split W into two wavefronts, one for $\overline{ca}$ and the other for $\overline{ad}$. To do so, we first split the binary tree $T(W)$ that represents the current wavefront W at α^*. Then, we do binary search on A^* to find the tangent point from a, where A^* is the underlying chain of α^*. We also split α^* into two at the tangent point of A^*—that is, split A^* into two chains that form two generators, one for $\overline{ac}$ and the other for $\overline{ad}$ (Figure 36). As A^* is represented by an array, splitting α^* can be performed in $O(1)$ time by resetting the end indices of the chains in the array. This finishes the propagation algorithm in $\triangle acd$. The preceding splits W into two wavefronts, one for $\overline{ac}$ and the other for $\overline{ad}$; we then propagate the wavefronts through $\overline{ac}$ and $\overline{ad}$ recursively.

(2) If α^* is the first generator of W, then α^* must be α_0—that is, the leftmost generator of $W(\overline{cd})$. In this case, we do not need to split W. But we still need to split the generator α_0 at the tangent point p_0 of α_0 from a. To find the tangent point p_0, however, this time we do not use binary search, as it is possible that we will need to do this for $\Omega(n_b)$ vertices of $bay(\overline{cd})$, which would take $\Omega(n_b \log n_g)$ time in total. Instead, we use the following approach. Recall that the vertex $c = v_0$ connects to α_0 by a tangent with tangent point c_0 (e.g., see Figure 34), and c_0 is an endpoint of A'_0. We traverse A'_0 from c_0 to b_0—that is, the other endpoint of A'_0;

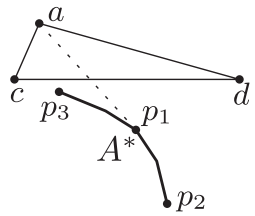

Fig. 36. Splitting the generator α^*. Assume that A^* is the convex chain from p_2 to p_3 with p_2 as the initial vertex of the generator. $\overline{ap_1}$ is tangent to A^* at p_1. After the split, the chain from p_3 to p_1 becomes a generator with p_1 as the initial vertex and the chain from p_1 to p_2 becomes another generator with p_2 as the initial vertex.

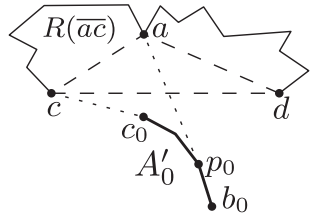

Fig. 37. Illustrating the case for propagating the one-generator wavefront to $R(\overline{ac})$.

for each vertex, we check whether it is the tangent from a. In this way, p_0 can be found in time linear in the number of vertices of A'_0 between c_0 and the tangent point p_0 (Figure 37). After that, we still split α_0 into two generators at p_0; one is the only generator for $\overline{ac}$ and the other becomes the first generator of the wavefront for $\overline{ad}$.

For $\overline{ad}$, we propagate its wavefront through $\overline{ad}$ recursively. For $\overline{ac}$, to propagate its wavefront through $\overline{ac}$, since the wavefront has only one generator, we can simply apply the linear time shortest path map algorithm for simple polygons [20]; we refer to this as the *one-generator case*. Indeed, $\overline{ac}$ partitions $bay(\overline{cd})$ into two subpolygons and let $R(\overline{ac})$ denote the one that does not contain $\triangle cda$ (e.g., see Figure 37). Hence, all points of $R(\overline{ac})$ are claimed by the only generator for $\overline{ac}$, whose underlying chain A' is the subchain of A'_0 from c_0 to p_0 and whose initial vertex is p_0. Consider the region $R'(\overline{ac})$ bounded by $\overline{cc_0}$, A', $\overline{p_0a}$, and the boundary of $R(\overline{ac})$ excluding $\overline{ac}$. It is a simple polygon with a single weighted source p_0. Therefore, our problem is equivalent to computing the shortest path map in $R'(\overline{ac})$ with respect to the source point p_0, which can be done in $O(|R(\overline{ac})| + |A'|)$ time [20], where $|R(\overline{ac})|$ and $|A'|$ are the numbers of vertices of $R(\overline{ac})$ and $|A'|$, respectively.

(3) If α^* is the last generator of W, the algorithm is symmetric to the preceding second case.

The preceding describes our algorithm for propagating the wavefront $W(\overline{cd})$ through the first triangle $\overline{cda}$. Next, we discuss the general case where we propagate a wavefront W of more than one generator through an arbitrary triangle of the triangulation of $bay(\overline{cd})$. For the sake of notational convenience, we consider the problem of propagating the wavefront $W(\overline{ad})$ at $\overline{ad}$ through $\overline{ad}$ into the region $R(\overline{ad})$, where $R(\overline{ad})$ is the one of the two subpolygons of $bay(\overline{cd})$ partitioned by $\overline{ad}$ that does not contain $\triangle cda$. Let $\triangle adb$ be the triangle in $R(\overline{ad})$ with $\overline{ad}$ with as an edge—that is, b is the third vertex of the triangle. The algorithm for propagating $W(\overline{ad})$ through $\triangle adb$ is quite similar to the preceding. We first sort all bisector events in $\triangle adb$, following their perpendicular

distances to the supporting line of $\overline{ad}$. Then we process these events in the same way as before. After all events are processed, the wavefront is finally split at the vertex b.

The preceding algorithm is for the case where $\overline{ad}$ is a triangulation diagonal. If $\overline{ad}$ is an obstacle edge, then the wavefront $W(\overline{ad})$ stops at $\overline{ad}$. Notice that each bisector of the wavefront $W(\overline{ad})$ must intersect $\overline{ad}$. For each bisector of the wavefront, starting from its current tracing-point, we trace it out until the current traced hyperbolic-arc intersects $\overline{ad}$.

The algorithm stops once all triangles are processed as previously.

Time Analysis. We now analyze the time complexity. As discussed before, the initial step for computing the intersections of adjacent bisectors of the wavefront $W(\overline{cd})$ and locating the triangles containing them together takes $O(m \log(n_g + n_b))$ time. During the entire algorithm, each traced bisector hyperbolic-arc belongs to the shortest path map in $bay(\overline{cd})$ (i.e., the portion of $SPM(s)$ in $bay(\overline{cd})$), whose size is $O(n_b + n_g + m)$. Hence, the total time on the bisector tracing in the entire algorithm is $O(n_b + n_g + m)$. For the one-generator case where only one generator is available for $\overline{ac}$, the time for processing the subpolygon $R(\overline{ac})$ is $O(|R(\overline{ac})| + |A'|)$. Notice that all such subpolygons $R(\overline{ac})$ in the one-generator case are interior disjoint. Hence, the total sum of their sizes is $O(n_b)$. In addition, all such generator underlying chains A' are also interior disjoint, and thus the total sum of their sizes is $O(n_g)$. Therefore, the overall time for processing the one-generator case subpolygons is $O(n_g + n_b)$.

For the general case of processing a triangle $\triangle$ of the triangulation, the total time for processing all events is $O(m'' \log(n_g + n_b))$,[8] where m'' is the number of events in $\triangle$, both valid and invalid. Each event is computed either in the initial step or after a generator is deleted. The total number of bisector events in the former case in the entire algorithm is at most $m - 1$. The total number in the latter case in the entire algorithm is no more than the number of generators that are deleted in the algorithm, which is at most m because once a generator is deleted it will never appear in any wavefront again. Hence, the total time for processing events in the entire algorithm is $O(m \log(n_g + n_b))$. Once all events in $\triangle$ are processed, we need to find the generator α^* of the current wavefront W that claims the third vertex b of the triangle, by binary search plus bisector tracing. The time is $O(\log m)$ plus the time for tracing the bisector hyperbolic-arcs. Hence, excluding the time for tracing the bisector hyperbolic-arcs, which has been counted previously, the total time for this operation in the entire algorithm is $O(m' \log m)$, where m' is the number of triangles the algorithm processed for the case where α^* is not the first or last generator. We will show later in Lemma 4.2 that $m' = O(m)$.

After α^* is found, depending on whether α^* is the first or last generator of W, there are three cases. If α^* is not the first or last generator of W, then we find the tangent from b to α^* by binary search in $O(\log n_g)$ time and split both W and α; otherwise, we find the tangent by a linear scan on α^* and only need to split α^*. Splitting W takes $O(\log m)$ time while splitting a generator only takes $O(1)$ time as discussed before. Therefore, the total time for splitting generators in the entire algorithm is $O(n_b)$, as there are $O(n_b)$ triangles in the triangulation. If α^* is either the first or last generator of W, then a one-generator case subproblem will be produced and the time of the linear scan for finding the tangent is $O(|A'|)$, where A' is the subchain of α^* that belongs to the one-generator case subproblem. As discussed earlier, all such A' in all one-generator case subproblems are interior disjoint, and thus the total time on the linear scan in the entire algorithm is $O(n_g)$. Therefore, the total time for finding the tangent point and splitting W is $O(m' \log n_g + n_b)$ as $m \leq n_g$.

[8] More specifically, sorting all events takes $O(m'' \log m'')$ time and processing each event takes $O(\log(n_g + n_b))$ time.

Lemma 4.2. $m' \leq m - 1$.

Proof. Initially $W = W(\overline{cd})$, which consists of m generators. Each split operation splits a wavefront W at a generator α^* into two wavefronts, each of which has at least two generators (such that both wavefronts contain α^*). More specifically, if a wavefront of size k is split, then one wavefront has k' generators and the other has $k - k' + 1$ generators with $k' \geq 2$ and $k - k' + 1 \geq 2$. The value m' is equal to the number of all split operations in the algorithm.

We use a tree structure T to characterize the split operations. The root of T corresponds to the initial generator sequence $W(\overline{cd})$. Each split on a wavefront W corresponds to an internal node of T with two children corresponding to the two subsequences of W after the split. Hence, m' is equal to the number of internal nodes of T. In the worst case, T has m leaves, each corresponding to two generators α_i and α_{i+1} for $i = 0, 1, \ldots, m$. Since each internal node of T has two children and T has at most m leaves, the number of internal nodes of T is at most $m - 1$. Therefore, $m' \leq m - 1$. □

With the preceding lemma, the total time of the algorithm is $O(m \log(n_g + n_b) + n_g + n_b)$, which is $O(m \log m + n_g + n_b)$ by similar analysis as Observation 1. The space is $O(n_g + n_b + m)$, as no persistent data structures are used.

4.4 Expanding $SPM'(\mathcal{M})$ into All Canals

Consider a canal $canal(x, y)$ with two gates $\overline{xd}$ and $\overline{yz}$. The goal is to expand the map $SPM'(\mathcal{M})$ into $canal(x, y)$ through the two gates to obtain the shortest path map in the canal, denoted by $SPM(canal(x, y))$.

The high-level scheme of the algorithm is similar in spirit to that for the L_1 problem [9]. The algorithm has three main steps. First, we expand $SPM'(\mathcal{M})$ into $canal(x, y)$ through the gate $\overline{xd}$, by applying our algorithm for bays. Let $SPM_1(canal(x, y))$ denote the map of $canal(x, y)$ obtained by the algorithm. Second, we expand $SPM(\mathcal{M})$ into $canal(x, y)$ through the gate $\overline{yz}$ by a similar algorithm as earlier; let $SPM_2(canal(x, y))$ denote the map of $canal(x, y)$ obtained by the algorithm. Third, we merge the two maps $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$ to obtain $SPM(canal(x, y))$. This is done by using the merge step from the standard divide-and-conquer Voronoi diagram algorithm to compute the region closer to the generators at $\overline{xd}$ than those at $\overline{yz}$. We will provide more details for this step in the following, and we will show that this step can be done in time linear in the total size of the two maps $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$. Before doing so, we analyze the complexities of the algorithm.

The first step takes $O(n + h \log h)$ time for all canals. So does the second step. The third step takes linear time in the total size of $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$. Since the total size of the two maps over all canals is $O(n)$, the total time of the third step for all canals is $O(n)$. In summary, the time for computing the shortest path maps in all canals is $O(n + h \log h)$ and the space is $O(n)$.

In the following, we provide more details for the third step of the algorithm.

Recall that x and y are the two endpoints of the corridor path π in $canal(x, y)$. It is possible that the shortest s-x path $\pi(s, x)$ contains y or the shortest s-y path $\pi(s, y)$ contains x. To determine that, we can simply check whether $d(s, x) + |\pi| = d(s, y)$ and whether $d(s, y) + |\pi| = d(s, x)$. Note that both $d(s, x)$ and $d(s, y)$ are available once $SPM(\mathcal{M})$ is computed.

We first consider the case where neither $\pi(s, x)$ contains y nor $\pi(s, y)$ contains x. In this case, there must be a point p^* in π such that $d(s, x) + |\pi(x, p^*)| = d(s, y) + |\pi(y, p^*)|$, where $\pi(x, p^*)$ (respectively, $\pi(y, p^*)$) is the subpath of π between x (respectively, y) and p^*. We can easily find p^* in $O(|\pi|)$ time. To merge the two maps $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$ to obtain $SPM(canal(x, y))$, we find a dividing curve Γ in $canal(x, y)$ such that $W(\overline{xd})$ claims all points of

$canal(x, y)$ on one side of Γ while $W(\overline{yz})$ claims all points of $canal(x, y)$ on the other side of Γ, where $W(\overline{xd})$ is the set of generators of $SPM'(\mathcal{M})$ claiming $\overline{xd}$ (one may consider $W(\overline{xd})$ is a wavefront) and $W(\overline{yz})$ is the set of generators of $SPM'(\mathcal{M})$ claiming $\overline{yz}$. The curve Γ consists of all points in $canal(x, y)$ that have equal weighted distances to $W(\overline{xd})$ and $W(\overline{yz})$. Therefore, the point p^* must be on Γ. Starting from p^*, we can trace Γ out by walking simultaneously in the cells of the two maps $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$. The running time is thus linear in the total size of the two maps.

We then consider the case where either $\pi(s, x)$ contains y or $\pi(s, y)$ contains x. Without loss of generality, we assume the latter case. We first check whether $\pi(s, p)$ contains x for all points p on the gate $\overline{yz}$. To do so, according to the definitions of corridor paths and gates of canals, for any point $p \in \overline{yz}$, its shortest path to x in $canal(x, y)$ is the concatenation of the corridor path π and $\overline{yp}$. Hence, it suffices to check whether $d(s, z) = d(s, y) + |\overline{yz}|$:

- If yes, then all points of $\overline{yz}$ are claimed by the generators of $W(\overline{xd})$ (in fact, they are claimed by x because for any point $q \in \overline{xd}$ and any point $p \in \overline{yz}$, their shortest path in $canal(x, y)$ is the concatenation of $\overline{qx}$, π, and $\overline{yp}$). Hence, all points in $canal(x, y)$ are claimed by $W(\overline{xd})$ and $SPM(canal(x, y))$ is $SPM_1(canal(x, y))$.
- Otherwise, some points of $\overline{yz}$ are claimed by $W(\overline{xd})$ while others are claimed by $W(\overline{yz})$. As in the preceding case, we need to find a dividing curve Γ consisting of all points with equal weighted distances to $W(\overline{xd})$ and $W(\overline{yz})$. To this end, we again first find a point p^* in Γ. For this, since $\pi(s, y)$ contains x, y is claimed by $W(\overline{xd})$. However, since $d(s, z) \neq d(s, y) + |\overline{yz}|$, z is claimed by $W(\overline{yz})$. Therefore, $\overline{yz}$ must contains a point $p^* \in \Gamma$. Such a point p^* can be found by traversing $\overline{yz}$ simultaneously in the cells of both $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$. After p^* is found, we can again trace Γ out by walking simultaneously in the cells of $SPM_1(canal(x, y))$ and $SPM_2(canal(x, y))$. The running time is also linear in the total size of the two maps.

4.5 Wrapping Things Up

The following theorem summarizes our result.

Theorem 4.3. *Suppose $\mathcal{P}$ is a set of h pairwise disjoint polygonal obstacles with a total of n vertices in the plane and s is a source point. Assume that a triangulation of the free space is given. The shortest path map $SPM(s)$ with respect to s can be constructed in $O(n + h \log h)$ time and $O(n)$ space.*

Proof. Using the triangulation, we decompose the free space $\mathcal{F}$ into an ocean $\mathcal{M}$, canals, and bays in $O(n)$ time [31]. Then, the shortest path map $SPM(\mathcal{M})$ in the ocean $\mathcal{M}$ can be constructed in $O(n + h \log h)$ time and $O(n)$ space. Next, $SPM(\mathcal{M})$ can be expanded into all bays and canals in additional $O(n + h \log h)$ time and $O(n)$ space. The shortest path map $SPM(s)$ is thus obtained. □

The current best algorithms can compute a triangulation of the free space in $O(n \log n)$ time or in $O(n + h \log^{1+\epsilon} h)$ time for any small $\epsilon > 0$ [2].

After $SPM(s)$ is computed, by building a point location data structure [16, 32] on $SPM(s)$ in additional $O(n)$ time, given a query point t, the shortest path length from s to t can be computed in $O(\log n)$ time and a shortest s-t path can be produced in time linear in the number of edges of the path.

Corollary 4.4. *Suppose $\mathcal{P}$ is a set of h pairwise disjoint polygonal obstacles with a total of n vertices in the plane and s is a source point. Assume that a triangulation of the free space is given. A data structure of $O(n)$ space can be constructed in $O(n + h \log h)$ time and $O(n)$ space so that given*

any query point t, the shortest path length from s to t can be computed in $O(\log n)$ time and a shortest s-t path can be produced in time linear in the number of edges of the path.

REFERENCES

[1] S. W. Bae and H. Wang. 2019. L_1 shortest path queries in simple polygons. *Theoretical Computer Science* 790 (2019), 105–116.

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

[3] D. Z. Chen, J. Hershberger, and H. Wang. 2013. Computing shortest paths amid convex pseudodisks. *SIAM Journal on Computing* 42, 3 (2013), 1158–1184.

[4] D. Z. Chen, R. Inkulu, and H. Wang. 2016. Two-point L_1 shortest path queries in the plane. *Journal of Computational Geometry* 1 (2016), 473–519.

[5] D. Z. Chen, K. S. Klenk, and H.-Y. T. Tu. 2000. Shortest path queries among weighted obstacles in the rectilinear plane. *SIAM Journal on Computing* 29, 4 (2000), 1223–1246.

[6] D. Z. Chen and H. Wang. 2015. Computing shortest paths among curved obstacles in the plane. *ACM Transactions on Algorithms* 11 (2015), Article 26.

[7] D. Z. Chen and H. Wang. 2015. A new algorithm for computing visibility graphs of polygonal obstacles in the plane. *Journal of Computational Geometry* 6 (2015), 316–345.

[8] D. Z. Chen and H. Wang. 2017. Computing the visibility polygon of an island in a polygonal domain. *Algorithmica* 77 (2017), 40–64.

[9] D. Z. Chen and H. Wang. 2019. Computing L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica* 81 (2019), 2430–2483.

[10] Y.-J. Chiang and J. S. B. Mitchell. 1999. Two-point Euclidean shortest path queries in the plane. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'99)*. 215–224.

[11] K. L. Clarkson, R. Cole, and R. E. Tarjan. 1992. Randomized parallel algorithms for trapezoidal diagrams. *International Journal of Computational Geometry and Application* 2 (1992), 117–133.

[12] K. Clarkson, S. Kapoor, and P. Vaidya. 1987. Rectilinear shortest paths through polygonal obstacles in $O(n \log^2 n)$ time. In *Proceedings of the 3rd Annual Symposium on Computational Geometry (SoCG'87)*. 251–257.

[13] K. Clarkson, S. Kapoor, and P. Vaidya. 1988. Rectilinear shortest paths through polygonal obstacles in $O(n \log^{2/3} n)$ time. Manuscript.

[14] P. J. de Rezende, D. T. Lee, and Y. F. Wu. 1989. Rectilinear shortest paths in the presence of rectangular barriers. *Discrete and Computational Geometry* 4 (1989), 41–53.

[15] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. 2020. The Open Problems Project. Retrieved January 23, 2023 from https://topp.openproblem.net/.

[16] H. Edelsbrunner, L. Guibas, and J. Stolfi. 1986. Optimal point location in a monotone subdivision. *SIAM Journal on Computing* 15, 2 (1986), 317–340.

[17] S. D. Eriksson-Bique, J. Hershberger, V. Polishchuk, B. Speckmann, S. Suri, T. Talvitie, K. Verbeek, and H. Yıldız. 2015. Geometric k shortest paths. In *Proceedings of the 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*. 1616–1625.

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

[19] L. J. Guibas and J. Hershberger. 1989. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences* 39, 2 (1989), 126–152.

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

[21] L. Guibas, J. Hershberger, and J. Snoeyink. 1991. Compact interval trees: A data structure for convex hulls. *International Journal of Computational Geometry and Applications* 1, 1 (1991), 1–22.

[22] J. Hershberger. 1991. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters* 38, 5 (1991), 231–235.

[23] J. Hershberger, V. Polishchuk, B. Speckmann, and T. Talvitie. 2014. Geometric kth shortest paths. In *Proceedings of the 30th Annual Symposium on Computational Geometry (SoCG'14)*. Article 96, 2 pages. http://www.computational-geometry.org/SoCG-videos/socg14video/ksp/applet/index.html.

[24] J. Hershberger and J. Snoeyink. 1994. Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications* 4 (1994), 63–97.

[25] J. Hershberger and S. Suri. 1999. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing* 28 (1999), 2215–2256.

[26] J. Hershberger, S. Suri, and H. Yıldız. 2013. A near-optimal algorithm for shortest paths among curved obstacles in the plane. In *Proceedings of the 29th Annual Symposium on Computational Geometry (SoCG'13)*. 359–368.
[27] S. Hertel and K. Mehlhorn. 1985. Fast triangulation of the plane with respect to simple polygons. *Information and Control* 64 (1985), 52–76.
[28] R. Inkulu, S. Kapoor, and S. N. Maheshwari. 2010. A near optimal algorithm for finding Euclidean shortest path in polygonal domain. *arXiv:1011.6481v1* (2010).
[29] S. Kapoor and S. N. Maheshwari. 1988. Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles. In *Proceedings of the 4th Annual ACM Symposium on Computational Geometry (SoCG'88)*. 172–182.
[30] S. Kapoor and S. N. Maheshwari. 2000. Efficiently constructing the visibility graph of a simple polygon with obstacles. *SIAM Journal on Computing* 30, 3 (2000), 847–871.
[31] S. Kapoor, S. N. Maheshwari, and J. S. B. Mitchell. 1997. An efficient algorithm for Euclidean shortest paths among polygonal obstacles in the plane. *Discrete and Computational Geometry* 18 (1997), 377–383.
[32] D. Kirkpatrick. 1983. Optimal search in planar subdivisions. *SIAM Journal on Computing* 12 (1983), 28–35.
[33] D. Kirkpatrick and J. Snoeyink. 1995. Tentative prune-and-search for computing fixed-points with applications to geometric computation. *Fundamenta Informaticae* 22 (1995), 353–370.
[34] D. T. Lee and F. P. Preparata. 1984. Euclidean shortest paths in the presence of rectilinear barriers. *Networks* 14 (1984), 393–410.
[35] J. S. B. Mitchell. 1989. An optimal algorithm for shortest rectilinear paths among obstacles. In *Abstracts of* the *1st Canadian Conference on Computational Geometry*.
[36] J. S. B. Mitchell. 1991. A new algorithm for shortest paths among obstacles in the plane. *Annals of Mathematics and Artificial Intelligence* 3 (1991), 83–105.
[37] J. S. B. Mitchell. 1992. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica* 8 (1992), 55–88.
[38] J. S. B. Mitchell. 1996. Shortest paths among obstacles in the plane. *International Journal of Computational Geometry and Applications* 6 (1996), 309–332.
[39] J. S. B. Mitchell and S. Suri. 1995. Separation and approximation of polyhedral objects. *Computational Geometry: Theory and Applications* 5 (1995), 95–114.
[40] E. Oh. 2019. Optimal algorithm for geodesic nearest-point Voronoi diagrams in simple polygons. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'19)*. 391–409.
[41] M. Overmars and J. van Leeuwen. 1981. Maintenance of configurations in the plane. *Journal of Computer and System Sciences* 23 (1981), 166–204.
[42] H. Rohnert. 1986. Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters* 23 (1986), 71–76.
[43] N. Sarnak and R. E. Tarjan. 1986. Planar point location using persistent search trees. *Communications of the ACM* 29 (1986), 669–679.
[44] M. Sharir and A. Schorr. 1986. On shortest paths in polyhedral spaces. *SIAM Journal on Computing* 15 (1986), 193–215.
[45] J. A. Storer and J. H. Reif. 1994. Shortest paths in the plane with polygonal obstacles. *Journal of the ACM* 41 (1994), 982–1012.
[46] H. Wang. 2019. Quickest visibility queries in polygonal domains. *Discrete and Computational Geometry* 62 (2019), 374–432.
[47] H. Wang. 2020. A divide-and-conquer algorithm for two-point L_1 shortest path queries in polygonal domains. *Journal of Computational Geometry* 11 (2020), 235–282.
[48] H. Wang. 2021. Shortest paths among obstacles in the plane revisited. In *Proceedings of the 32nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'21)*. 810–821.

Received 30 June 2021; revised 21 October 2022; accepted 10 January 2023