The Farthest-Point Geodesic Voronoi Diagram of Points on the Boundary of a Simple Polygon*

Eunjin Oh[1], Luis Barba[2], and Hee-Kap Ahn[3]

1 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
jin9082@postech.ac.kr

2 **Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium; and School of Computer Science, Carleton University, Ottawa, Canada**
lbarbafl@ulb.ac.be

3 **Department of Computer Science and Engineering, POSTECH, 77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea**
heekap@postech.ac.kr

Abstract

Given a set of sites (points) in a simple polygon, the farthest-point geodesic Voronoi diagram partitions the polygon into cells, at most one cell per site, such that every point in a cell has the same farthest site with respect to the geodesic metric. We present an $O((n+m)\log\log n)$-time algorithm to compute the farthest-point geodesic Voronoi diagram for m sites lying on the boundary of a simple n-gon.

1998 ACM Subject Classification I.3.5 Computational Geometry and Object Modeling

Keywords and phrases Geodesic distance, simple polygons, farthest-point Voronoi diagram

Digital Object Identifier 10.4230/LIPIcs.SoCG.2016.56

1 Introduction

Let P be a simple polygon with n vertices. Given two points x and y in P, the *geodesic path* $\pi(x,y)$ is the shortest path contained in P connecting x with y. Note that if the straight-line segment connecting x with y is contained in P, then $\pi(x,y)$ is a straight-line segment. Otherwise, $\pi(x,y)$ is a polygonal chain whose vertices (other than its endpoints) are reflex vertices of P. We refer the reader to [10] for more information on geodesic paths.

The *geodesic distance* between x and y, denoted by $d(x,y)$, is the sum of the Euclidean lengths of each segment in $\pi(x,y)$. Throughout this paper, when referring to the distance between two points in P, we mean the geodesic distance between them. To ease the description, we assume that each vertex of P has a unique farthest neighbor. This *general position* condition was also assumed by Aronov et al. [3] and Ahn et al. [2] and can be obtained by applying a slight perturbation to the positions of the vertices [7].

Let S be a set of m *sites* (points) contained in P. Given a point $x \in P$, a (geodesic) *S-farthest neighbor* of x, is a site $\mathrm{N}(P,S,x)$ (or simply $\mathrm{N}(x)$) of S that maximizes the geodesic distance to x. Let $F_S : P \to \mathbb{R}$ be the function that maps each point $x \in P$ to the distance to a S-farthest neighbor of x (i.e., $F_S(x) = d(x, \mathrm{N}(x))$). A point x in P that minimizes $F_S(x)$ is called the *geodesic center* of S (in P).

* This was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the government of Korea.

We can decompose P into *Voronoi cells* such that for each site $s \in S$, $\mathsf{Cell}(s)$ is the set of points $x \in P$ such that $d(x, s)$ is strictly larger than $d(x, s')$ for any other site s' of S (some cells might be empty). The set $\text{int}(P) \setminus \cup_{s \in S} \mathsf{Cell}(s)$ defines the (farthest) *Voronoi tree* of S with root at the geodesic center of S and leaves on the boundary of P. Each edge of this diagram consists of a sequence of straight-lines and hyperbolic arcs [3].

The Voronoi tree together with the set of Voronoi cells defines the *farthest-point geodesic Voronoi diagram* of S (in P), denoted by $\mathsf{FVD}[S]$ (or simply FVD if S is clear from context). Thus, we indistinctively refer to FVD as a tree or as a set of Voronoi cells.

There are many similarities between the Euclidean farthest-point Voronoi diagram and the farthest-point geodesic Voronoi diagram (see [3] for further references). In the Euclidean case, a site has a nonempty Voronoi cell if and only if it is extreme, i.e., it lies on the boundary of the convex hull of the set of sites. Moreover, the clockwise sequence of Voronoi cells (at infinity) is the same as the clockwise sequence of sites along the boundary of the convex hull. With these properties, the Euclidean farthest-point Voronoi diagram can be computed in linear time if the convex hull of the sites is known [1].

In the geodesic case, a site with nonempty Voronoi cell lies on the boundary of the geodesic convex hull of the sites. The clockwise order of the Voronoi cells along the boundary of P is a subsequence of the clockwise order of sites along the boundary of the geodesic convex hull. However, the cell of an extreme site may be empty, roughly because the polygon is not large enough for the cell to appear. In addition, the complexity of the bisector between two sites can be linear to the complexity of the polygon.

Previous work. Since the early 1980s many classical geometric problems have been studied in the geodesic setting. The problem of computing the geodesic diameter of the vertices of a simple n-gon P (and its counterpart, the geodesic center) received a lot of attention from the computational geometry community. Chazelle [6] gave the first algorithm for computing the geodesic diameter. This algorithm runs in $O(n^2)$ time using linear space. Suri [13] reduced the complexity to $O(n \log n)$-time without increasing the space complexity. Finally, Hershberger and Suri [8] presented a fast matrix search technique, one application of which is a linear-time algorithm for computing the diameter of P. A key step in this process is the computation of the farthest neighbor of each vertex in P.

The first algorithm for computing the geodesic center was given by Asano and Toussaint [4], and runs in $O(n^4 \log n)$-time. This algorithm computes a super set of the vertices of $\mathsf{FVD}[V]$, where V is the set of vertices of P. In 1989, Pollack et al. [12] improved the running time to $O(n \log n)$ time. In a recent paper, Ahn et al. [2] settled the complexity of this problem by presenting a $\Theta(n)$-time algorithm to compute the geodesic center of a simple n-gon.

Since the geodesic center and diameter can both be computed from $\mathsf{FVD}[V]$ in linear time, the problem of computing farthest-point geodesic Voronoi diagrams is a strict generalization. For a set S of m points in P, Aronov et al. [3] presented an algorithm to compute $\mathsf{FVD}[S]$ in $O((n+m)\log(n+m))$ time. While a trivial lower bound of $\Omega(n + m \log m)$ is known for this general problem, there has been no progress closing this gap. In other words, it is not known whether or not the dependence on n, the complexity of P, is linear in the running time. In fact, this problem was explicitly posed by Mitchell [10, Chapter 27] in the Handbook of Computational Geometry.

Our result. In this paper, we present an $O((n+m)\log\log n)$-time algorithm to compute FVD of m points on the boundary of a simple n-gon. This is the first improvement on the computation of farthest-point geodesic Voronoi diagrams since 1993 [3]. Indeed, while we

consider sites lying on the boundary of the polygon only, our approach can also be extended to handle arbitrary sites in the polygon. Then the running time becomes $O(n \log\log n + m \log(n+m))$. The details can be found in the full version of this paper. Our result suggests that the computation time of Voronoi diagrams has only a close-to-linear dependence in the complexity of the polygon. We believe our results could be used as a stepping stone to solve the question posed by Mitchell [10, Chapter 27]. Due to lack of space, some of the proofs are omitted. All missing proofs can be found in the full version of this paper.

1.1 Outline

The algorithm consists of three phases. First, we compute the farthest-point geodesic Voronoi diagram restricted to the boundary of the polygon. Then we recursively decompose the interior of the polygon into smaller (non-Voronoi) cells until the complexity of each of them becomes constant. Finally, we explicitly compute the farthest-point geodesic Voronoi diagram in each of the cells and merge them to complete the description of the Voronoi diagram.

In order to compute the Voronoi diagram of S, we start by computing the restriction of $\mathsf{FVD}[S]$ to the boundary of P in linear time. The main tool used to speed up the algorithm is the matrix search technique introduced by Hershberger and Suri [8] which provides a "partial" description of $\mathsf{FVD}[S] \cap \partial P$ (i.e., the restriction of $\mathsf{FVD}[S]$ to the vertices of P.) To extend it to the entire boundary of P, we borrow some tools used by Ahn et al. [2]. This reduces the problem to the computation of upper envelopes of distance functions which can be completed in linear time.

Once $\mathsf{FVD}[S]$ restricted to ∂P is computed, we recursively split the polygon into cells by a closed polygonal path in time linear to the complexity of the cell. By recursively repeating this procedure on each resulting cell, we guarantee that after $O(\log\log n)$ rounds the boundary of each cell consists of a constant number of geodesic paths. In particular, we guarantee that each cell is a pseudo-triangle, a quadrilateral, or a simple polygon enclosed by a convex chain and a concave chain which we call a *lune-cell*.

While decomposing the polygon, we also compute the farthest-point geodesic Voronoi diagram of S restricted to the boundary of each cell. Each round can be completed in linear time which leads to an overall running time of $O((n+m)\log\log n)$.

Finally, we compute the farthest-point geodesic Voronoi diagram restricted to each cell in time linear to the complexity of the cell using the algorithm in [5].

2 Decomposing the boundary

Given a set A of points, let ∂A and $\mathrm{int}(A)$ denote the boundary and the interior of A, respectively. Let P be a simple n-gon and S be a set of m sites (points) contained in ∂P. Throughout most of this paper, we will make the assumption that S is the set of all vertices of P. This assumption is general enough as we show how to extend the result to the case when S is an arbitrary set of sites contained on the boundary of P in Section 6.

The following result was used by Ahn et al. [2] and is based on the matrix search technique developed by Hershberger and Suri [8].

▶ **Lemma 1** (Result from [8]). *We can compute the S-farthest neighbor of each vertex of P in $O(n)$ time.*

Using Lemma 1, we mark the vertices of P that are S-farthest neighbors of at least one vertex of P. Let M denote the set of marked vertices of P (clearly this set can be computed

in $O(n)$ time after applying Lemma 1). In other words, M contains all vertices of P whose Voronoi region contains at least one vertex of P.

For a marked vertex w of P, the vertices of P whose farthest neighbor is w appear contiguously along ∂P [3]. That is, given an edge uv such that $\textsc{n}(u) = \textsc{n}(v)$, we know that $\textsc{n}(x) = \textsc{n}(u) = \textsc{n}(v)$ for each point $x \in uv$. Therefore, after computing all these farthest neighbors, we effectively split ∂P into subchains, each associated with a different vertex of M (see [2] further for the first use of this technique).

Given two points x and y on ∂P, let $C[x, y]$ denote the portion of ∂P from x to y in clockwise order. We say that three (nonempty) disjoint sets A_1, A_2 and A_3 contained in ∂P are in *clockwise order* if $A_2 \subset C[a, c]$ for any $a \in A_1$ and any $c \in A_3$.

▶ **Lemma 2** ([3, Corollary 2.7.4]). *The order of sites with nonempty Voronoi cells along ∂P is the same as the order of Voronoi cells along ∂P.*

We call an edge ab of P a *transition edge* if $\textsc{n}(a) \neq \textsc{n}(b)$. Let ab be a transition edge of P such that b is the clockwise neighbor of a along ∂P. Recall that we have computed $\textsc{n}(a)$ and $\textsc{n}(b)$ and note that $a, b, \textsc{n}(a), \textsc{n}(b)$ are in clockwise order by Lemma 2. Let v be a vertex of P such that $\textsc{n}(a), v, \textsc{n}(b)$ are in clockwise order. If there is a point x on ∂P whose farthest neighbor is v, then x must lie on ab. In other words, the Voronoi cell $\mathsf{Cell}(v)$ restricted to ∂P is contained in ab and hence, there is no vertex u of P such that $\textsc{n}(u) = v$.

Since we know which vertex is the farthest neighbor of each non-transition edge of P, to complete the description of FVD restricted to ∂P it suffices to compute FVD restricted to transition edges. To this end, we need some tools introduced in the following sections.

2.1 The apexed triangles

An *apexed triangle* $\triangle = (a, b, c)$ with *apex* $\textsc{a}(\triangle) = a$ is a triangle contained in P with an associated distance function $g_\triangle(x)$ such that (1) $\textsc{a}(\triangle)$ is a vertex of P, (2) there is an edge of ∂P containing both b and c, and (3) there is a vertex $\textsc{d}(\triangle)$ of P, called the *definer* of $\triangle$, such that

$$g_\triangle(x) = \begin{cases} \|x - \textsc{a}(\triangle)\| + d(\textsc{a}(\triangle), \textsc{d}(\triangle)) = d(x, \textsc{d}(\triangle)) & \text{if } x \in \triangle \\ -\infty & \text{if } x \notin \triangle, \end{cases}$$

where $\|x - y\|$ denote the Euclidean distance between x and y.

Intuitively, $\triangle$ bounds a constant complexity region where the geodesic distance function from $\textsc{d}(\triangle)$ can be obtained by looking only at the distance from $\textsc{a}(\triangle)$. We call the side of an apexed triangle $\triangle$ opposite to the apex the *bottom side* of $\triangle$. Note that the bottom side of $\triangle$ is contained in an edge of P.

The concept of the apexed triangle was introduced by Ahn et al. [2]. After computing the farthest S-neighbor of each vertex, they show how to compute a linear number of apexed triangles in linear time with the following property: for each point $p \in P$, there exists an apexed triangle $\triangle$ such that $p \in \triangle$ and $\textsc{d}(\triangle) = \textsc{n}(p)$. By the definition of the apexed triangle, we have $d(p, \textsc{n}(p)) = g_\triangle(p)$. To summarize the results presented by Ahn et al. [2], we need some definitions. Given a chain C contained in ∂P with endpoints u and v, the *funnel* of a site s to C, denoted by $\gamma_s(C)$, is the weakly simple polygon contained in P bounded by C, $\pi(u, s)$ and $\pi(s, v)$.

▶ **Lemma 3** (Summary of [2]). *Given a simple n-gon P with vertex set S, we can compute a set of $O(n)$ apexed triangles in $O(n)$ time with the property that for any site $s \in S$, the union of all apexed triangles with definer s is a funnel γ_s such that $\mathsf{Cell}(s) \subset \gamma_s$.*

In other words, Lemma 3 states that for each site s of S, the set of apexed triangles with definer s forms a connected component. In particular, the union of their bottom sides is a connected chain along ∂P. Moreover, these apexed triangles are interior disjoint.

2.2 The refined farthest-point geodesic Voronoi diagram

We consider a refined version of FVD which we call the *refined farthest-point geodesic Voronoi diagram* defined as follows: for each site $s \in S$, the Voronoi cell $\mathsf{Cell}(s)$ of FVD is subdivided by the apexed triangles with definer s. That is, for each apexed triangle $\triangle$ with definer s, we define a *refined cell* $\mathsf{rCell}(\triangle) = \mathrm{int}(\triangle) \cap \mathsf{Cell}(s)$. Since any two apexed triangles $\triangle_1$ and $\triangle_2$ with the same definer are interior disjoint, we know that $\mathsf{rCell}(\triangle_1)$ and $\mathsf{rCell}(\triangle_2)$ are also interior disjoint. We denote the set $\mathrm{int}(P) \setminus \cup_{\triangle} \mathsf{rCell}(\triangle)$ by rFVD. Then, rFVD forms a tree consisting of arcs and vertices. Notice that each arc of rFVD is a connected subset of either the bisector of two sites or a side of an apexed triangle. Since the number of the apexed triangles is $O(n)$, the complexity of rFVD is still linear.

▶ **Lemma 4.** *For a point x in $\mathsf{rCell}(\triangle)$ for an apexed triangle $\triangle$, the line segment connecting x and y is contained in $\mathsf{rCell}(\triangle)$, where y is the point on the bottom side of $\triangle$ hit by the ray from $\mathrm{A}(\triangle)$ towards x. Moreover, $\mathsf{rCell}(\triangle)$ is connected.*

3 Computing the farthest-point geodesic Voronoi diagram restricted to the boundary of the polygon

We compute all apexed triangles satisfying the condition in Lemma 3 in $O(n)$ time [2]. Recall that the apexed triangles with the same definer are interior disjoint and have their bottom sides on ∂P whose union forms a connected chain. Moreover, their union is a funnel by Lemma 3. Thus, the apexed triangles with the same definer can be sorted along ∂P.

▶ **Lemma 5.** *Let s be a site in S and let $\tau_s \neq \emptyset$ be the set of all apexed triangles with definer s. We can sort the apexed triangles in τ_s along ∂P with respect to their bottom sides in $O(|\tau_s|)$ time.*

3.1 Computing rFVD restricted to a transition edge

Let uv be a transition edge of P such that u is the clockwise neighbor of v. Without loss of generality, we assume that uv is horizontal and u lies to the left of v. Recall that if there is a site s with $\mathsf{Cell}(s) \cap uv \neq \phi$, then s lies in $C[\mathrm{N}(v), \mathrm{N}(u)]$. Thus, to compute $\mathsf{rFVD} \cap uv$, it is sufficient to consider the apexed triangles with definers in $C[\mathrm{N}(v), \mathrm{N}(u)]$. Let A be the set of apexed triangles with definers in $C[\mathrm{N}(v), \mathrm{N}(u)]$.

In this section, we give a procedure to compute $\mathsf{rFVD} \cap uv$ in $O(|A|)$ time using the sorted lists of the apexed triangles with definers in $C[\mathrm{N}(v), \mathrm{N}(u)]$. Once it is done for all transition edges, we have the refined farthest-point geodesic Voronoi diagram restricted to ∂P. Let $s_1 = \mathrm{N}(u), s_2, \ldots, s_\ell = \mathrm{N}(v)$ be the sites lying on $C[\mathrm{N}(v), \mathrm{N}(u)]$ in counterclockwise order along ∂P.

3.1.1 An upper envelope and rFVD

Consider any t functions $f_1, \ldots, f_t$ with $f_j : D \to \mathbb{R} \cup \{-\infty\}$ for $1 \leq j \leq t$, where D is any point set. We define the *upper envelope* of $f_1, \ldots, f_t$ as the function $f : D \to \mathbb{R} \cup \{-\infty\}$ such that $f(x) = \max_{1 \leq j \leq t} f_j(x)$. Moreover, we say that a function f_j *appears* on the upper envelope if $f_j(x) = f(x) \in \mathbb{R}$ at some point x.

In this subsection, we restrict the domain of the distance functions $g_\triangle$ to uv. By definition, the upper envelope of $g_\triangle$ for all apexed triangles $\triangle \in A$ on uv coincides with $\mathsf{rFVD} \cap uv$ in its projection on uv. We consider the sites one by one in order and compute the upper envelope of $g_\triangle$ for all apexed triangles $\triangle \in A$ on uv as follows.

While the upper envelope of $g_\triangle$ for all apexed triangles $\triangle \in A$ is continuous on uv, the upper envelope of $g_{\triangle'}$ of all apexed triangles $\triangle'$ with definers from s_1 up to s_k on uv (we simply say the upper envelope for sites from s_1 to s_k) might be discontinuous at some point on uv for $1 \leq k < \ell$. Let w be the leftmost point where the upper envelope for sites from s_1 to s_k is discontinuous. Then we define $U(s_k)$ as the function such that $U(s_k)(x)$ is the value of the upper envelope for sites from s_1 to s_k at x for a point x lying to the left of w, and $U(s_k)(x) = -\infty$ for a point x lying to the right of w. By definition, $U(\textsc{n}(v))$ is the upper envelope of the distance functions of all apexed triangles in A. Note that $\mathsf{rCell}(\triangle) \cap uv = \phi$ for some apexed triangle $\triangle \in A$. Thus the distance function of an apexed triangle might not appear on $U(s_k)$ on uv. Let $\tau_U(s_k)$ be the list of the apexed triangles whose distance functions appear on $U(s_k)$ sorted along uv from u with respect to their bottom sides. Note that if $\textsc{d}(\triangle_i) \neq \textsc{d}(\triangle_{i+1})$, the bisector of $\textsc{d}(\triangle_i)$ and $\textsc{d}(\triangle_{i+1})$ crosses the intersection of the bottom sides of $\triangle_i$ and $\triangle_{i+1}$ for two consecutive apexed triangles $\triangle_i$ and $\triangle_{i+1}$ of $\tau_U(s_k)$.

3.1.2 A procedure for computing $U(s_\ell)$

Suppose that we have already computed $U(s_{k-1})$ and $\tau_U(s_{k-1})$ for some index $2 \leq k \leq \ell$. We show how to compute $U(s_k)$ and $\tau_U(s_k)$ from $U(s_{k-1})$ and $\tau_U(s_{k-1})$ in the following. We use two auxiliary lists U' and τ'_U which are initially set to $U(s_{k-1})$ and $\tau_U(s_{k-1})$. We update U' and τ'_U until they finally become $U(s_k)$ and $\tau_U(s_k)$, respectively.

Let τ_k be the list of the apexed triangles with definer s_k sorted along ∂P with respect to their bottom sides. For any apexed triangle $\triangle$, we denote the list of the apexed triangles in τ_k overlapping with $\triangle$ in their bottom sides by $\tau_O(\triangle)$. Also, we denote the lists of the apexed triangles in $\tau_k \setminus \tau_O(\triangle)$ lying left to $\triangle$ and lying right to $\triangle$ with respect to their bottom sides by $\tau_L(\triangle)$ and $\tau_R(\triangle)$, respectively.

Let $\triangle_a$ denote the last element (the rightmost apexed triangle) of τ'_U. With respect to $\triangle_a$, we partition τ_k into three disjoint sublists $\tau_L(\triangle_a)$, $\tau_O(\triangle_a)$ and $\tau_R(\triangle_a)$. We can compute these sublists in $O(|\tau_k|)$ time.

Case 1: Some apexed triangles in τ_k overlap with $\triangle_a$. If $\tau_O(\triangle_a) \neq \phi$, let $\triangle$ be the leftmost apexed triangle in $\tau_O(\triangle_a)$. We compare the distance functions $g_\triangle$ and $g_{\triangle_a}$ on $\triangle_a \cap \triangle \cap uv$. That is, we compare $d(x, s_k)$ and $d(x, \textsc{d}(\triangle_a))$ for $x \in \triangle_a \cap \triangle \cap uv$.

1. If there is a point on $\triangle_a \cap \triangle \cap uv$ that is equidistant from s_k and $\textsc{d}(\triangle_a)$, $g_\triangle$ appears on $U(s_k)$. Moreover, the distance functions of the apexed triangles in $\tau_R(\triangle)$ also appear on $U(s_k)$, and no apexed triangle in $\tau_L(\triangle)$ appears on $U(s_k)$ by Lemma 2. Thus we append $\triangle$ and the apexed triangles in $\tau_R(\triangle)$ at the end of τ'_U. We also update U' accordingly. Then, τ'_U and U' are $\tau_U(s_k)$ and $U(s_k)$, respectively.
2. If $d(x, \textsc{d}(\triangle_a)) > d(x, s_k)$ for all points $x \in \triangle_a \cap \triangle \cap uv$, then $\triangle$ and its distance function do not appear on $\tau_U(s_k)$ and $U(s_k)$, respectively, by Lemma 2. Thus we do nothing and scan the apexed triangles in $\tau_O(\triangle_a) \cup \tau_R(\triangle_a)$, except $\triangle$, from left to right until we find an apexed triangle $\triangle'$ such that there is a point on $\triangle_a \cap \triangle' \cap uv$ which is equidistant from $\textsc{d}(\triangle_a)$ and s_k. Then we apply the procedure in (1) with $\triangle'$ instead of $\triangle$. If there is no such apexed triangle, we have $U(s_k) = U'$ and $\tau_U(s_k) = \tau'_U$.

3. Otherwise, we have $d(x, s_k) > d(x, \mathrm{D}(\triangle_a))$ for all points $x \in \triangle_a \cap \triangle \cap uv$. Then the distance function of $\triangle_a$ does not appear on $U(s_k)$. Thus, we remove $\triangle_a$ and its distance function from τ'_U and U', respectively. We consider the apexed triangles in $\tau_L(\triangle_a)$ from right to left. For an apexed triangle $\triangle' \in \tau_L(\triangle_a)$, we do the following. Since τ'_U is updated, we update $\triangle_a$ to the last element of τ'_U. Afterwards, we check whether $d(x, s_k) \geq d(x, \mathrm{D}(\triangle_a))$ for all points $x \in \triangle_a \cap \triangle' \cap uv$ if $\triangle'$ overlaps with $\triangle_a$. If so, we remove $\triangle_a$ from τ'_U and update $\triangle_a$. We do this until we find an apexed triangle $\triangle' \in \tau_L(\triangle_a)$ such that this test fails. Then, there is a point on $\triangle' \cap \triangle_a \cap uv$ which is equidistant from $\mathrm{D}(\triangle_a)$ and s_k. After we reach such an apexed triangle $\triangle'$, we apply the procedure in (1) with $\triangle'$ instead of $\triangle$.

Case 2: No apexed triangle in τ_k overlaps with $\triangle_a$. If $\tau_O(\triangle_a) = \phi$, we cannot compare the distance function of any apexed triangle in τ_k with the distance function of $\triangle_a$ directly, so we need a different method to handle this.

There are two possible subcases: either $\tau_L(\triangle_a) = \phi$ or $\tau_R(\triangle_a) = \phi$. Note that these are the only possible subcases since the union of the apexed triangles with the same definer is connected. For the former subcase, the upper envelope of sites from s_1 to s_k is discontinuous at the right endpoint of the bottom side of $\triangle_a$. Thus $g_\triangle$ does not appear on $U(s_k)$ for any apexed triangle $\triangle \in \tau_k$. Thus $U(s_k) = U'$ and $\tau_U(s_k) = \tau'_U$.

For the latter subcase, at most one of s_k and $\mathrm{D}(\triangle_a)$ has its Voronoi cell in $\mathsf{FVD}[S_k]$, where $S_k = \{s_1, \ldots, s_k\}$, by Lemma 2. We can find a site (s_k or $\mathrm{D}(\triangle_a)$) which does not have its Voronoi cell in $\mathsf{FVD}[S_k]$ in $O(1)$ time. Due to lack of space, we omit the description of this procedure. It can be found in the full version of this paper.

If s_k does not have its Voronoi cell in $\mathsf{FVD}[S_k]$, then $U(s_k) = U'$ and $\tau_U(s_k) = \tau'_U$. If $\mathrm{D}(\triangle_a)$ does not have its Voronoi cell in $\mathsf{FVD}[S_k]$, we remove all apexed triangles with definer $\mathrm{D}(\triangle_a)$ from τ'_U and their distance functions from U'. Since such apexed triangles lie at the end of τ'_U consecutively, it takes the time proportional to the number of the apexed triangles. Afterwards, we do this until the last element of τ_k and the last element of τ'_U overlap in their bottom sides. When the two elements overlap, we apply the procedure of Case 1.

In total, the running time is bounded by $O(|A|)$.

▶ **Theorem 6.** *The farthest-point geodesic Voronoi diagram of the vertices of a simple n-gon P restricted to the boundary of P can be computed in $O(n)$ time.*

4 Decomposing the polygon into smaller cells

Until now, we have computed $\mathsf{rFVD} \cap \partial P$ of size $O(n)$. We add the points in $\mathsf{rFVD} \cap \partial P$ to the vertex set of P, and apply the algorithm to compute the apexed triangles with respect to the vertex set of P again [2]. Note that now there is no transition edge. Thus all apexed triangles are disjoint in their bottom sides. We have the set of the apexed triangles sorted along ∂P with respect to their bottom sides.

A subset A of P is *geodesically convex* if $\pi(x, y) \subseteq A$ for any $x, y \in A$. We define a *t-path-cell* for some $t \in \mathbb{N}$ as a simple polygon contained in P with all vertices on ∂P which is geodesically convex and has at most t convex vertices.

In the following, for a cell C, $|\partial C|$ denotes the number of edges of C. For a curve γ, $|\mathsf{rFVD} \cap \gamma|$ denotes the number of the refined cells intersecting γ.

Sketch of the algorithm. We subdivide P into t-path-cells recursively for some $t \in \mathbb{N}$ until each cell becomes a base cell. There are three types of base cells. The first type is

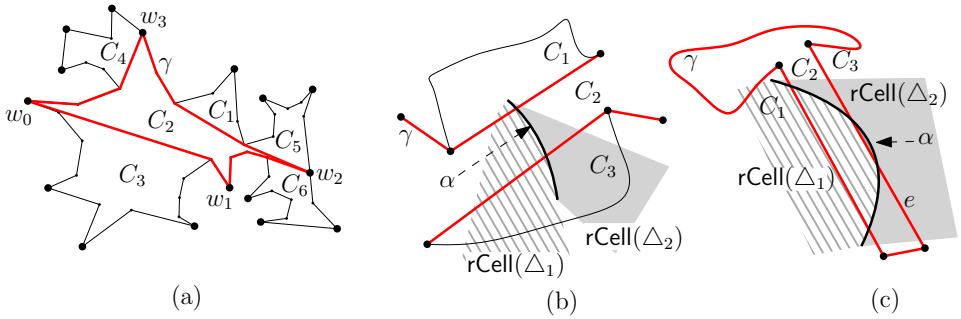

Figure 1 (a) The region bounded by the black curve is a 16-path-cell. All convex vertices are marked with black disks. The region is subdivided into six 5-path-cells by the red thick curve consisting of $\pi(w_0, w_1), \pi(w_1, w_2), \pi(w_2, w_3)$ and $\pi(w_3, w_0)$. (b) The arc α of rFVD intersects C_1, C_2, C_3 and crosses C_2. (c) The arc α of rFVD intersects C_1, C_2, C_3 and crosses C_2. Note that α does not cross C_3.

a quadrilateral crossed by exactly one arc of rFVD through two opposite sides, which we call an *arc-quadrilateral*. The second type is a 3-path-cell. Note that a 3-path-cell is a pseudo-triangle. The third type is a region of P whose boundary consists of one convex chain and one concave chain, which we call a *lune-cell*. Note that a convex polygon is a lune-cell whose concave chain is just a vertex of the polygon.

Let $\{t_k\}$ be the sequence such that $t_1 = n$ and $t_k = \lfloor\sqrt{t_{k-1}}\rfloor + 1$. Initially, P itself is a t_1-path-cell. Assume that the kth iteration is completed. We show how to subdivide each t_k-path-cell with $t_k > 3$ into t_{k+1}-path-cells and base cells in the $(k+1)$th iteration in Section 4.1. Note that a base cell is not subdivided further.

While subdividing the polygon into cells, we compute $\mathsf{rFVD} \cap \partial C$ for each cell C (of any kind) in time linear on $|\partial C|$ and $|\mathsf{rFVD} \cap \partial C|$. In Section 5, we show how to compute $\mathsf{rFVD} \cap T$ for each base cell T in $O(|\mathsf{rFVD} \cap \partial T|)$ time once $\mathsf{rFVD} \cap \partial T$ is computed.

Note that $t_k \leq 3$ with $k = c \log\log n$ for some constant $1 < c$. Moreover, in the kth iteration, P is subdivided into t_k-path-cells and base cells. Thus, in $O(\log\log n)$ iterations, every t-path-cell gets subdivided into base cells. We will show that each iteration takes $O(n)$ time in Section 4.1, which implies that the overall running time for the computation in this section is $O(n \log\log n)$. We will also show that the total complexity of rFVD restricted to the boundaries of all cells in the kth iteration is $O(kn)$ for any $k \in \mathbb{N}$. See Lemma 10.

4.1 Subdividing a t-path-cell into smaller cells

If a t_k-path-cell C is a pseudo-triangle or a lune-cell, C is a base cell and we do not subdivide it further. Otherwise, we subdivide it using the algorithm in this section.

The subdivision consists of three phases. In Phase 1, we subdivide each t_k-path-cell into t_{k+1}-path-cells by a curve connecting at most t_{k+1} vertices of the t_k-path-cell. In Phase 2, we subdivide each t_{k+1}-path-cell further along an arc of rFVD crossing the cell if there is such an arc. In Phase 3, we subdivide cells created in Phase 2 into t_{k+1}-path-cells and lune-cells.

4.1.1 Phase 1. Subdivision by a curve connecting at most t_{k+1} vertices

Let C be a t_k-path-cell computed in the kth iteration. Recall that C consists of at most t_k convex vertices and is simple. Let β be the largest integer satisfying that $\beta\lfloor\sqrt{t_k}\rfloor$ is less than the number of the convex vertices of C. Then we have $\beta \leq \lfloor\sqrt{t_k}\rfloor + 1 = t_{k+1}$.

We choose $\beta+1$ vertices $w_0, w_1, \ldots, w_\beta$ from the convex vertices of C as follows. We choose a convex vertex of C and denote it by w_0. Then we choose the $j\lfloor\sqrt{t_k}\rfloor$th convex vertex of C from w_0 in clockwise order and denote it by w_j for all $j = 1, \ldots, \beta$. We set $w_{\beta+1} = w_0$. Then we construct the closed curve γ_C (or simply γ when C is clear from context) consisting of the geodesic paths $\pi(w_0, w_1), \pi(w_1, w_2), \ldots, (w_\beta, w_0)$. See Figure 1(a). In other words, the closed curve γ_C is the boundary of the geodesic convex hull of $w_0, \ldots, w_\beta$. Note that γ does not cross itself. Moreover, γ is contained in C since C is geodesically convex.

We compute γ in time linear to the number of edges of C as follows. The algorithm computing geodesic paths in [11] takes k source-destination pairs as input, where both sources and destinations are on the boundary of a simple polygon. It returns the geodesic path between the source and the destination for each input pair. For all pairs, computing the geodesic paths takes $O(N+k)$ time in total if k shortest paths do not cross (but possibly overlap) one another, where N is the complexity of the polygon. In our case, the pairs (w_j, w_{j+1}) for $j = 0, \ldots, \beta$ are $\beta+1$ input source-destination pairs. Since the geodesic paths for all input pairs do not cross one another, γ can be computed in $O(\beta + |\partial C|) = O(|\partial C|)$ time. Then we compute $\mathsf{rFVD} \cap \gamma$ in $O(|\mathsf{rFVD} \cap \partial C| + |\partial C|)$ time using $\mathsf{rFVD} \cap \partial C$ which has already been computed in the kth iteration. We will describe this procedure in Section 4.2.

The curve γ subdivides C into t_{k+1}-path-cells. The set $C \setminus \gamma$ consists of at least $\beta+2$ connected components. Note that the closure of each connected component is a t_{k+1}-path-cell. Moreover, the union of the closures of all connected components is exactly the closure of C since C is simple. These components define the *subdivision of C induced by γ*.

4.1.2 Phase 2. Subdivision along an arc of rFVD

After subdividing C into t_{k+1}-path-cells $C_1, \ldots, C_\delta$ ($\delta \geq \beta+2$) by the curve γ_C, an arc α of rFVD may cross C_j for some $1 \leq j \leq \delta$. In Phase 2, for each arc α crossing C_j, we isolate the subarc $\alpha \cap C_j$ from C_j by creating a new cell which we call an arc-quadrilateral. For an arc-quadrilateral $\square$ created by an arc α, we have $\mathsf{rFVD} \cap \square = \alpha \cap C_j$.

To bound the number of arc-quadrilaterals created in each iteration and the running time for Phase 2, we need the following technical lemma.

▶ **Lemma 7.** *For a geodesic convex polygon C with t convex vertices ($t \in \mathbb{N}$), let γ be a simple closed curve connecting at most t convex vertices of C such that every two consecutive vertices in clockwise order are connected by a geodesic path. Then, for each arc α of* rFVD *with $\alpha \cap C \neq \phi$, α intersects at most three cells in the subdivision of C by γ.*

Since C is geodesically convex, α intersects at most two edges of a cell C in Phase 1, which can be proved in a way similar to the proof of Lemma 7. This implies that $\alpha \cap C_j$ consists of at most two connected components. We say an arc α of rFVD *crosses* a cell C' if exactly two edges of C' intersect α. For example, in Figure 1(c), α crosses C_2 while α does not cross C_3 because there is only one edge of C_3 intersecting α.

First, we find an arc α of rFVD that crosses C_j. Since the points in $\mathsf{rFVD} \cap \partial C_j$ along ∂C_j have already been computed, we can scan them in clockwise order. For all arcs, it can be done in $O(|\mathsf{rFVD} \cap C_j|)$ time by the following lemma.

▶ **Lemma 8.** *All arcs α of* rFVD *crossing C_j can be found in $O(|\mathsf{rFVD} \cap \partial C_j|)$ time. Moreover, for all such α, the pairs $(\triangle_1, \triangle_2)$ of apexed triangles such that $\alpha \cap C_j = \{x \in C_j : g_{\triangle_1}(x) = g_{\triangle_2}(x) > 0\}$ can be found in the same time in total.*

Recall that $\alpha \cap C_j$ consists of at most two connected components. For the case that it consists exactly two connected components, we consider each connected component separately. Thus we show the case that $\alpha \cap C_j$ is connected.

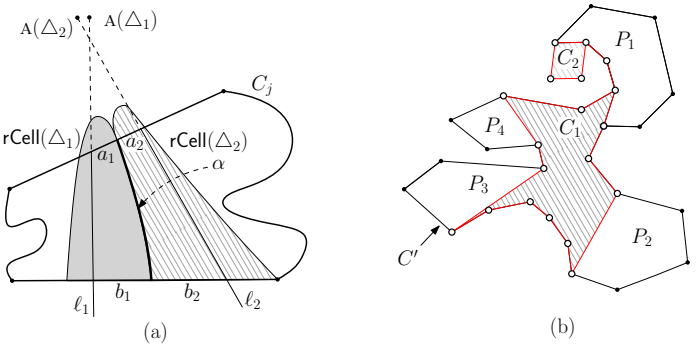

Figure 2 (a) The arc α of rFVD crosses C_j. Thus we isolate α by creating the arc-quadrilateral bounded by ℓ_1, ℓ_2 and ∂C_j. (b) The vertices marked with empty disks are vertices of P while the others are vertices of arc-quadrilaterals lying in $\mathrm{int}(P)$. We subdivide the cell into two t-path-cell C_1, C_2 and four lune-cells $P_1, \ldots, P_4$.

For an arc α crossing C_j, we subdivide C_j further into two cells with t' convex vertices for $t' \leq t_{k+1}$ and one arc-quadrilateral by adding two line segments bounding α such that no arc other than α intersects the arc-quadrilateral. Let $(\triangle_1, \triangle_2)$ be the pair of apexed triangles defining α. Let a_1, b_1 (and a_2, b_2) be the two connected components of $\mathsf{rCell}(\triangle_1) \cap \partial C_j$ (and $\mathsf{rCell}(\triangle_2) \cap \partial C_j$) incident to α such that a_1, a_2 are adjacent to each other and b_1, b_2 are adjacent to each other. See Figure 2(a).

Without loss of generality, we assume that a_1 is closer than b_1 to $\mathrm{A}(\triangle_1)$. Let x be any point on a_1. Then the farthest neighbor of x is the definer of $\triangle_1$. We consider the line ℓ_1 passing through x and the apex of $\triangle_1$. Then the intersection between C_j and ℓ_1 is contained in the closure of $\mathsf{rCell}(\triangle_1)$ by Lemma 4. Similarly, we find the line ℓ_2 passing through the apex of $\triangle_2$ and a point on a_2.

We subdivide C_j into two cells with at most t_{k+1} convex vertices and one arc-quadrilateral by two lines ℓ_1 and ℓ_2. The quadrilateral bounded by the two lines and ∂C_j is an arc-quadrilateral since α is the only arc of rFVD that intersects the quadrilateral. We do this for all arcs crossing some C_j. Note that no arc crosses the resulting cells other than arc-quadrilaterals by the construction. Then the resulting cells with at most t_{k+1} convex vertices and arc-quadrilaterals are the cells in the subdivision of C in the second phase.

4.1.3 Phase 3. Subdivision by a geodesic convex hull

Note that some cell C' with t' convex vertices for $3 < t' \leq t_{k+1}$ created in Phase 2 might be neither a t'-path-cell nor a base cell. Such a cell has some vertices in $\mathrm{int}(P)$ and some vertices of P as its vertices. In Phase 3, we subdivide such cells further into t'-path-cells and base cells.

To subdivide C' into t_{k+1}-path-cells and base cells, we first compute the geodesic convex hull CH of the vertices of C' which are vertices of P in time linear to the number of edges in C' using the algorithm for computing k-pair shortest paths in [11]. Consider the connected components of $C' \setminus \partial\mathsf{CH}$. There are two types of the connected components. A connected component of the first type is enclosed by a simple closed curve which is part of $\partial\mathsf{CH}$. For example, C_1 and C_2 in Figure 2(b) belong to this type. A connected component of the second type is enclosed by a subchain of $\partial\mathsf{CH}$ from u to w in clockwise order and a subchain of $\partial C'$ from w to u in counterclockwise order for some $u, w \in \partial P$. For example, P_i in Figure 2(b) belongs to the second type for $i = 1, \ldots, 4$.

By the construction, a connected component belonging to the first type has all its vertices on ∂P. Moreover, it has at most t' convex vertices since C' has t' convex vertices. Therefore, the closure of a connected component of $C' \setminus \partial\mathsf{CH}$ belonging to the first type is a t'-path-cell.

Every vertex of C' lying in $\mathrm{int}(P)$ is convex with respect to C' by the construction of C'. Thus, for a connected component P' belonging to the second type, the part of $\partial P'$ from $\partial C'$ is a convex chain with respect to P'. Moreover, the part of $\partial P'$ from $\partial\mathsf{CH}$ is the geodesic path between two points, thus it is a concave chain with respect to P'. Therefore, the closure of a connected component belonging to the second type is a lune-cell.

Since C' is a simple polygon, the union of the closures of all connected components of $C' \setminus \mathsf{CH}$ is exactly the closure of C'. The closures of all connected components of the first and the second types are t_{k+1}-path-cells and lune-cells created in the last phase of the $(k+1)$th iteration, respectively. We compute the t_{k+1}-path-cells and the lune-cells subdivided by $\partial\mathsf{CH}$. Then, we compute $\mathsf{rFVD} \cap \partial\mathsf{CH}$ using the procedure in Section 4.2. The resulting t_{k+1}-path-cells and base cells are the final decomposition of C of the $(k+1)$th iteration.

4.1.4 Analysis of the complexity

We first bound the complexity of the refined farthest-point geodesic Voronoi diagram restricted to the boundary of the cells in each iteration. The following technical lemma is used to bound the complexity.

▶ **Lemma 9.** *An arc α of* rFVD *intersects at most nine t_k-path-cells and $O(k)$ base cells at the end of the kth iteration. Moreover, there are at most three t_k-path-cells that α intersects but does not cross at the end of the kth iteration.*

Now we are ready to bound the complexities of the cells and rFVD restricted to the cells in each iteration. Then we finally prove that the running time of the algorithm in this section is $O(n \log\log n)$.

▶ **Lemma 10.** *At the end of the kth iteration, we have $\sum_{C:\text{a } t_k\text{-path-cell}} |\mathsf{rFVD} \cap \partial C| = O(n)$, $\sum_{C:\text{a } t_k\text{-path-cell}} |\partial C| = O(n)$, $\sum_{T:\text{a base cell}} |\mathsf{rFVD} \cap \partial T| = O(kn)$, and $\sum_{T:\text{a base cell}} |\partial T| = O(kn)$.*

Proof. Let α be an arc of rFVD. The first and the third complexity bounds hold by Lemma 9 and the fact that the number of the arcs of rFVD is $O(n)$.

The second complexity bound holds since the set of all edges of the t_k-path-cells is a subset of the chords in some triangulation of P. Note that any triangulation of P has $O(n)$ chords. Moreover, each chord is incident to at most two t_k-path-cells.

For the last complexity bound, the number of edges of base cells whose endpoints are vertices of P is $O(n)$ since they are chords in some triangulation of P. Thus we count the number of edges of base cells which are not incident to vertices of P. In Phase 1, we do not create any such edge. In Phase 2, we create at most $O(1)$ such edges whenever we create one arc-quadrilateral. All edges created in Phase 3 have their endpoints from the vertex set of P. Therefore, the total number of the edges of all base cells is asymptotically bounded by the number of arc-quadrilaterals, which is $O(kn)$. ◀

▶ **Corollary 11.** *In $O(\log\log n)$ iterations, P is subdivided into $O(n \log\log n)$ base cells.*

▶ **Lemma 12.** *The subdivision in each iteration can be done in $O(n)$ time.*

4.2 Computing rFVD restricted to the boundary of a t-path-cell

Recall that the bottom sides of all apexed triangles are interior-disjoint. Moreover, the union of them is ∂P. In this section, we describe a procedure to compute $\mathsf{rFVD} \cap \gamma$ in $O(|\mathsf{rFVD} \cap \partial C| + |\partial C|)$ time once $\mathsf{rFVD} \cap \partial C$ is computed. Recall that γ is a closed curve connecting consecutive points of every t_{k+1}th convex vertices of C in clockwise order.

If $\mathsf{rCell}(\triangle) \cap \gamma \neq \phi$ for an apexed triangle $\triangle$, then we have $\mathsf{rCell}(\triangle) \cap \partial C \neq \phi$. Thus, we consider only the apexed triangles $\triangle$ with $\mathsf{rCell}(\triangle) \cap \partial C \neq \phi$. Let $\mathcal{L}$ be the list of all such apexed triangles sorted along ∂P with respect to their bottom sides.

Consider a line segment ab contained in P. Without loss of generality, we assume that ab is horizontal and a lies to the left of b. Let $\triangle_a$ and $\triangle_b$ be the apexed triangles which maximize $g_{\triangle_a}(a)$ and $g_{\triangle_b}(b)$, respectively. If there is a tie by more than one apexed triangles, we choose an arbitrary one of them. With the two apexed triangles, we define two sorted lists $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$. Let $\mathcal{L}_{ab}$ be the sorted list of the apexed triangles in $\mathcal{L}$ which intersect ab and whose bottom sides lie from the bottom side of $\triangle_a$ to the bottom side of $\triangle_b$ in clockwise order along ∂P. Similarly, let $\mathcal{L}_{ba}$ be the sorted list of the apexed triangles in $\mathcal{L}$ which intersect ab and whose bottom sides lie from the bottom side of $\triangle_b$ to the bottom side of $\triangle_a$ in clockwise order along ∂P.

The following lemma together with Section 4.2.1 gives a procedure to compute $\mathsf{rFVD} \cap ab$. The procedure is similar to the procedure for computing $\mathsf{rFVD} \cap \partial P$ in Section 3.1.

▶ **Lemma 13.** *Let C be a geodesic convex polygon and a, b be two points with $ab \subset C$. Given the two sorted lists $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$, $\mathsf{rFVD} \cap ab$ can be computed in $O(|\mathcal{L}_{ab}| + |\mathcal{L}_{ba}|)$ time.*

Since an apexed triangle intersects at most two edges of γ, we can compute $\mathsf{rFVD} \cap \gamma$ in $O(|\mathcal{L}|) = O(|\mathsf{rFVD} \cap \partial C|)$ time once we have $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$ for all edges ab of γ.

4.2.1 Computing $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$ for all edges ab of γ

In this section, we show how to compute $\mathcal{L}_{ab}$ and $\mathcal{L}_{ba}$ for all edges ab of γ in $O(|\mathcal{L}| + |\partial C|)$ time. Recall that all endpoints of the geodesic paths bounding the t-path-cell C lie in ∂P. Let ab be an edge of γ, where b is the clockwise neighbor of a. The edge ab is a chord of P and divides P into two subpolygons such that $\gamma \setminus ab$ is contained in one of the subpolygons. Let $P_1(ab)$ be the subpolygon containing $\gamma \setminus ab$ and $P_2(ab)$ be the other subpolygon. For an apexed triangle in $\mathcal{L}_{ab}$, its bottom side lies in $\partial P_2(ab)$ and its apex lies in $\partial P_1(ab)$. On the other hand, for an apexed triangle in $\mathcal{L}_{ba}$, its bottom side lies in $\partial P_1(ab)$ and its apex lies in $\partial P_2(ab)$. Moreover, if its apex lies in $P_j(ab)$, then so does its definer for $j = 1, 2$. By the construction, $P_2(ab)$ and $P_2(e')$ are disjoint in their interior for any edge $e' \in \gamma \setminus \{ab\}$.

We compute $\mathcal{L}_{ab}$ for all edges ab in γ as follows. Initially, $\mathcal{L}_{ab}$ for all edges ab are set to ϕ. We update the list by scanning the apexed triangles in $\mathcal{L}$ from the first to the end. When we handle an apexed triangle $\triangle \in \mathcal{L}$, we first find the edge ab of γ such that $P_2(ab)$ contains the bottom side of $\triangle$ and check whether $\triangle \cap ab = \phi$. If it is nonempty, we append $\triangle$ to $\mathcal{L}_{ab}$. Otherwise, we do nothing. Then we handle the apexed triangle next to $\triangle$. For $\mathcal{L}_{ba}$, we do analogously, except that we find the edge ab of γ such that $P_2(ab)$ contains the definer of $\triangle$.

Note that any three apexed triangles $\triangle_1, \triangle_2, \triangle_3 \in \mathcal{L}$ appear on $\mathcal{L}$ in the order of their definers (and their bottom sides) appearing on ∂P. Thus to find the edge ab of γ such that $P_2(ab)$ contains the definer (or the bottom side) of $\triangle$, it is sufficient to check at most two edges; the edge e' such that $P_2(e')$ contains the bottom side of the apexed triangle previous to $\triangle$ in $\mathcal{L}$ and the clockwise neighbor of e'. Therefore, this procedure takes in $O(|\mathcal{L}|)$ time.

The following lemmas summarize this section.

▶ **Lemma 14.** *Let C be a t-path-cell and γ be a simple closed curve connecting at most t convex vertices of C lying on ∂P such that two consecutive vertices in clockwise order are connected by a geodesic path. Once $\mathsf{rFVD} \cap \partial C$ is computed, $\mathsf{rFVD} \cap \gamma$ can be computed in $O(|\mathsf{rFVD} \cap \partial C| + |\partial C|)$ time.*

▶ **Lemma 15.** *Each iteration takes $O(n)$ time and the algorithm in this section terminates in $O(\log \log n)$ iterations. Thus the running time of the algorithm in this section is $O(n \log \log n)$.*

5 Computing rFVD in the interior of a base cell

In this section, we consider a base cell T which is a lune-cell or a pseudo-triangle. Assume that $\mathsf{rFVD} \cap \partial T$ has already been computed. We extend $\mathsf{rFVD} \cap \partial T$ into the interior of the cell T in $O(|\mathsf{rFVD} \cap \partial T|)$ time.

To make the description easier, we first make two assumptions: (1) for any apexed triangle $\triangle$, $\mathsf{rCell}(\triangle) \cap \partial T$ is connected and contains the bottom side of $\triangle$, and (2) T is a lune-cell. In the full version of this paper, we show how to avoid the assumptions by subdividing each base cell and trimming each apexed triangle.

5.1 Definition for a new distance function

Without loss of generality, we assume that the bottom side of T is horizontal. We bound the domain by introducing a box B containing T. To apply the algorithm for computing the abstract Voronoi diagram in [5, 9], we need to define a new distance function $f_\triangle : B \to \mathbb{R}$ since $g_\triangle$ is not continuous. Imagine that we partition B into five regions with respect to an apexed triangle $\triangle$. We will define $f_\triangle$ as a function consisting of at most five algebraic functions each of whose domains corresponds to a partitioned region in B.

Consider five line segments ℓ_1, ℓ_2, ℓ_3, ℓ_4 and ℓ_5 such that their common endpoint is $\mathrm{A}(\triangle)$ and the other endpoints lie on ∂B. The line segments ℓ_1 and ℓ_2 contain the left and the right corners of $\triangle$, respectively. The line segments ℓ_3 and ℓ_5 are orthogonal to ℓ_2 and ℓ_1, respectively. The line segment ℓ_4 is contained in the line bisecting the angle of $\triangle$ at $\mathrm{A}(\triangle)$ but it does not intersect $\mathrm{int}(\triangle)$.

Then B is partitioned by these five line segments into five regions. We denote the region bounded by ℓ_1 and ℓ_2 which contains $\triangle$ by $G_{\mathrm{in}}(\triangle)$. Note that $\mathrm{D}(\triangle) \notin G_{\mathrm{in}}(\triangle)$ if $\mathrm{D}(\triangle) \neq \mathrm{A}(\triangle)$. The remaining four regions are denoted by $G_{\mathrm{Lside}}(\triangle)$, $G_{\mathrm{Ltop}}(\triangle)$, $G_{\mathrm{Rtop}}(\triangle)$, and $G_{\mathrm{Rside}}(\triangle)$ in the clockwise order from $G_{\mathrm{in}}(\triangle)$ along ∂B.

For a point $x \in G_{\mathrm{Lside}}(\triangle) \cup G_{\mathrm{Ltop}}(\triangle)$, let $\hat{x}_\triangle$ be the orthogonal projection of x on the line containing ℓ_1. Similarly, for a point $x \in G_{\mathrm{Rside}}(\triangle) \cup G_{\mathrm{Rtop}}(\triangle) \setminus \ell_4$, let $\hat{x}_\triangle$ be the orthogonal projection of x on the line containing ℓ_2. For a point $x \in G_{\mathrm{in}}(\triangle)$, we set $\hat{x}_\triangle = x$.

We define a new distance function $f_\triangle : B \to \mathbb{R}$ for each apexed triangle $\triangle$ with $\mathsf{rCell}(\triangle) \cap \partial T \neq \phi$ as follows.

$$f_\triangle(x) = \begin{cases} d(\mathrm{A}(\triangle), \mathrm{D}(\triangle)) - \|\hat{x}_\triangle - \mathrm{A}(\triangle)\| & \text{if } x \in G_{\mathrm{Ltop}}(\triangle) \cup G_{\mathrm{Rtop}}(\triangle), \\ d(\mathrm{A}(\triangle), \mathrm{D}(\triangle)) + \|\hat{x}_\triangle - \mathrm{A}(\triangle)\| & \text{otherwise.} \end{cases}$$

Note that $f_\triangle$ is continuous on B. Each contour curve, that is a set of points with the same function value, consists of two line segments and at most one circular arc.

Here, we assume that there is no pair $(\triangle_1, \triangle_2)$ of apexed triangles such that two sides, one from $\triangle_1$ and the other from $\triangle_2$, are parallel. If there exists such a pair, the contour curves for two apexed triangles may overlap. In the full version, we show how to avoid the assumption by slightly perturbing the distance function.

5.2 An algorithm for computing rFVD $\cap\, T$

To compute the farthest-point geodesic Voronoi diagram restricted to T, we apply the algorithms in [5, 9] with this new distance function, which computes the abstract Voronoi diagram in a domain where each site has a unique cell touching the boundary of the domain. While the algorithms in [5, 9] compute the abstract nearest-point Voronoi diagram, they can be used to compute the farthest-point Voronoi diagram. These algorithms are generalizations of the linear-time algorithm in [1], which computes the farthest-point and the nearest-point Voronoi diagram of points in convex position.

In the abstract Voronoi diagram, no explicit sites or distance functions are given. Instead, for any pair of sites s and s', the open domains $D(s, s')$ and $D(s', s)$ are given. Let A be the set of all apexed triangles with $\mathsf{rFVD} \cap \partial T$. In our problem, we regard the apexed triangles in A as the sites and B as the domain for the abstract Voronoi diagram. For two apexed triangles $\triangle_1$ and $\triangle_2$ in A, we define the open domain $D(\triangle_1, \triangle_2)$ as the set $\{x \in B : f_{\triangle_1}(x) > f_{\triangle_2}(x)\}$. We denote the abstract Voronoi diagram for the apexed triangles by aFVD and the cell of $\triangle$ on aFVD by $\mathsf{aCell}(\triangle)$.

Here, we need to show that the distance function we define in Section 5.1 satisfies the followings for any subset A' of A. A proof can be found in the full version of this paper.

1. For any two apexed triangles $\triangle_1$ and $\triangle_2$ in A, the set $\{x \in B : f_{\triangle_1}(x) = f_{\triangle_2}(x)\}$ is a curve with endpoints on ∂B. The curve consists of $O(1)$ pieces of algebraic curves.
2. Each apexed triangle $\triangle$ in A' has exactly one connected and nonempty cell in the abstract Voronoi diagram of A'.
3. Each point in B belongs to the closure of an abstract Voronoi cell.
4. The abstract Voronoi diagrams of A and A' form a tree and a forest, respectively.

Thus, we can compute aFVD using the algorithms in [5, 9]. The abstract Voronoi diagram restricted to T is exactly the refined farthest-point geodesic Voronoi diagram restricted to T. Note that we already have the abstract Voronoi diagram restricted to ∂T which coincides with the refined farthest-point geodesic Voronoi diagram restricted to ∂T. After computing aFVD on B, we traverse aFVD and extract aFVD lying inside T.

▶ **Lemma 16.** *Given a base cell T constructed by the subdivision algorithm in Section 4.1, $\mathsf{rFVD} \cap T$ can be computed in $O(|\mathsf{rFVD} \cap \partial T| + |\partial T|)$ time.*

▶ **Theorem 17.** *The farthest-point geodesic Voronoi diagram of the vertices of a simple n-gon can be computed in $O(n \log\log n)$ time.*

6 A set of sites on the boundary

In this section, we show that the results presented above are general enough to work when the set S is an arbitrary set of sites contained in the boundary of P.

Since S is a subset of sites contained in ∂P, we can assume without loss of generality that all sites of S are vertices of P by splitting the edges where they lie on. In this section, we decompose the boundary of P into chains of consecutive vertices that share the same S-farthest neighbor and edges of P whose endpoints have distinct S-farthest neighbors. The following lemma is a counterpart of Lemma 1. Lemma 1 is the only place where it was assumed that S is the set of vertices of P.

▶ **Lemma 18.** *Given a set S of m sites contained in ∂P, we can compute the S-farthest neighbor of each vertex of P in $O(n + m)$ time.*

Proof. Let $w : P \to \mathbb{R}$ be a real valued function on the vertices of P such that for each vertex v of P, $w(v) = D_P$ if $v \in S$, and $w(v) = 0$ otherwise, where D_P is any fixed constant larger than the diameter of P.

For each vertex $p \in P$, we want to identify $\text{N}(p)$. To this end, we define a new distance function $d^* : P \times P \to \mathbb{R}$ such that for any two points u and v of P, $d^*(u, v) = d(u, v) + w(u) + w(v)$. Using a result from Hershberger and Suri [8, Section 6.1 and 6.3], in $O(n + m)$ time we can compute the farthest neighbor of each vertex of P with respect to d^*.

By the definition of the function w, the maximum distance from any vertex of P is achieved at a site of S. Therefore, the farthest neighbor from a vertex v of P with respect to d^* is indeed the S-farthest neighbor, $\text{N}(v)$, of v. ◀

▶ **Theorem 19.** *The farthest-point geodesic Voronoi diagram of m points on the boundary of a simple n-gon can be computed in $O((n + m) \log \log n)$ time.*

References

1. Alok Aggarwal, Leonidas J Guibas, James Saxe, and Peter W Shor. A linear-time algorithm for computing the Voronoi diagram of a convex polygon. *Discrete & Computational Geometry*, 4(6):591–604, 1989.
2. Hee-Kap Ahn, Luis Barba, Prosenjit Bose, Jean-Lou De Carufel, Matias Korman, and Eunjin Oh. A linear-time algorithm for the geodesic center of a simple polygon. In *Proceedings of the 31st Symposium on Compututaional Geometry, SoCG*, pages 209–223, 2015.
3. Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, 1993.
4. T. Asano and G.T. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.
5. Cecilia Bohler, Rolf Klein, and Chih-Hung Liu. Forest-like abstract Voronoi diagrams in linear time. In *Proceedings of the 26th Canadian Conference on Computational Geometry, CCCG*, pages 133–141, 2014.
6. B Chazelle. A theorem on polygon cutting with applications. In *Proceedings 23rd Annual Symposium on Foundations of Computer Science, FOCS*, pages 339–349, 1982.
7. Herbert Edelsbrunner and Ernst Peter Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.
8. John Hershberger and Subhash Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.
9. Rolf Klein and Andrzej Lingas. Hamiltonian abstract Voronoi diagrams in linear time. In *Proceedings of the 5th International Symposium on Algorithms and Computation ISAAC*, pages 11–19, 1994.
10. J. S. B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.
11. Evanthia Papadopoulou. k-pairs non-crossing shortest paths in a simple polygon. *International Journal of Computational Geometry and Applications*, 9(6):533–552, 1999.
12. Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989.
13. Subhash Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39(2):220–235, 1989.