# Voronoi Diagrams for a Moderate-Sized Point-Set in a Simple Polygon*

**Eunjin Oh[1] and Hee-Kap Ahn[2]**

1 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
jin9082@postech.ac.kr
2 **Department of Computer Science and Engineering, POSTECH, Pohang, Korea**
heekap@postech.ac.kr

## Abstract

Given a set of sites in a simple polygon, a geodesic Voronoi diagram partitions the polygon into regions based on distances to sites under the geodesic metric. We present algorithms for computing the geodesic nearest-point, higher-order and farthest-point Voronoi diagrams of $m$ point sites in a simple $n$-gon, which improve the best known ones for $m \leq n/\operatorname{polylog} n$. Moreover, the algorithms for the nearest-point and farthest-point Voronoi diagrams are optimal for $m \leq n/\operatorname{polylog} n$. This partially answers a question posed by Mitchell in the Handbook of Computational Geometry.

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

**Keywords and phrases** Simple polygons, Voronoi diagrams, geodesic distance

**Digital Object Identifier** 10.4230/LIPIcs.SoCG.2017.52

## 1 Introduction

The *geodesic distance* between any two points $x$ and $y$ contained in a simple polygon is the length of the shortest path contained in the polygon connecting $x$ and $y$. A geodesic Voronoi diagram of a set $S$ of $m$ sites contained in a simple polygon $P$ partitions $P$ into regions based on distances to sites of $S$ under the geodesic metric. The *geodesic nearest-point Voronoi diagram* of $S$ partitions $P$ into cells, exactly one cell per site, such that every point in a cell has the same nearest site of $S$ under the geodesic metric. The higher-order Voronoi diagram, also known as the order-$k$ Voronoi diagram, is a generalization of the nearest-point Voronoi diagram. For an integer $k$ with $1 \leq k \leq m-1$, the *geodesic order-$k$ Voronoi diagram* of $S$ partitions $P$ into cells, at most one cell per $k$-tuple of sites, such that every point in a cell has the same $k$ nearest sites under the geodesic metric. Thus, the geodesic order-1 Voronoi diagram is the geodesic nearest-point Voronoi diagram. The geodesic order-$(m-1)$ Voronoi diagram is called the geodesic farthest-point Voronoi diagram. Hence, the *geodesic farthest-point Voronoi diagram* of $S$ partitions $P$ into cells, at most one cell per site, such that every point in a cell has the same farthest site under the geodesic metric.

In this paper, we study the problem of computing the geodesic nearest-point, higher-order and farthest-point Voronoi diagrams of a set $S$ of $m$ points contained in a simple $n$-gon $P$. Each edge of a geodesic Voronoi diagram is either a hyperbolic arc or a line segment consisting of points equidistant from two sites [2, 3, 11]. The boundary between any two neighboring cells of a geodesic Voronoi diagram is a chain of $O(n)$ edges. Each end vertex

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





of the boundary is of degree 1 or 3 under the assumption that no point in the polygon is equidistant from four distinct sites while every other vertex is of degree 2. There are $O(k(m-k))$ degree-3 vertices in the geodesic order-$k$ Voronoi diagram of $S$ [11]. Every degree-3 vertex is equidistant from three sites and is a point where three Voronoi cells meet. The number of degree-2 vertices is $\Theta(n)$ for the geodesic nearest-point Voronoi diagram and the geodesic farthest-point Voronoi diagram [2, 3]. For the geodesic order-$k$ Voronoi diagram, it is known that the number of degree-2 vertices is $O(kn)$ [11], but this bound is not tight.

The first nontrivial algorithm for computing the geodesic nearest-point Voronoi diagram was given by Aronov in 1989 [2]. Their algorithm takes $O((n+m)\log^2(n+m))$ time. Later, Papadopoulou and Lee [15] improved the running time to $O((n+m)\log(n+m))$. However, there has been no progress since then while the best known lower bound of the running time remains to be $\Omega(n+m\log m)$. In fact, Mitchell posed a question whether this gap can be resolved in the Handbook of Computational Geometry [13, Chapter 27].

For the geodesic order-$k$ Voronoi diagram, the first nontrivial algorithm was given by Liu and Lee [11] in 2013. Their algorithm works for a polygonal domain with holes and takes $O(k^2(n+m)\log(n+m))$ time. Thus, this algorithm also works for a simple polygon. They presented an asymptotically tight combinatorial complexity of the geodesic order-$k$ Voronoi diagram for a polygonal domain with holes, which is $\Theta(k(m-k)+kn)$. However, it is not tight for a simple polygon: the geodesic order-$(m-1)$ Voronoi diagram of $m$ points in a simple $n$-gon has complexity $\Theta(n+m)$ [3]. There is no bound better than the one by Liu and Lee known for the complexity of the geodesic order-$k$ Voronoi diagram in a simple polygon.

For the geodesic farthest-point Voronoi diagram, the first nontrivial algorithm was given by Aronov et al. [3] in 1993, which takes $O((n+m)\log(n+m))$ time. While the best known lower bound is $\Omega(n+m\log m)$, there has been no progress until Oh et al. [14] presented a faster $O((n+m)\log\log n)$-time algorithm for the special case that all sites are on the boundary of the polygon in 2016. They also claimed that their algorithm can be extended to compute the geodesic farthest-point Voronoi diagram for any $m$ points contained in a simple $n$-gon in $O(n\log\log n+m\log(n+m))$ time.[1]

**Our results.** Our main contributions are the algorithms for computing the nearest-point, higher-order and farthest-point Voronoi diagrams of $m$ sites in a simple $n$-gon, which improve the best known ones for $m \leq n/\operatorname{polylog} n$. To be specific, we present

- an $O(n+m\log m\log^2 n)$-time algorithm for the nearest-point Voronoi diagram,
- an $O(k^2 m\log m\log^2 n+\min\{nk, n(m-k)\})$-time algorithm for the order-$k$ Voronoi diagram, and
- an $O(n+m\log m+m\log^2 n)$-time algorithm for the farthest-point Voronoi diagram.

Moreover, our algorithms reduce the gaps of the running times towards the lower bounds. Our algorithm for the geodesic nearest-point Voronoi diagram is optimal for $m \leq n/\log^3 n$. Since the algorithm by Papadopoulou and Lee is optimal for $m \geq n$, our algorithm together with the one by Papadopoulou and Lee gives the optimal running time for computing the diagram, except for the case that $n/\log^3 n < m < n$.

Similarly, our algorithm for the geodesic farthest-point Voronoi diagram is optimal for $m \leq n/\log^2 n$. Since the algorithm by Aronov et al. [3] is optimal for $m \geq n$, our algorithm together with the one by Aronov et al. gives the optimal running time for computing the diagram, except for the case that $n/\log^2 n < m < n$. This answers the question posed by

[1] Details will be found in the journal version of their paper.

Mitchell on the geodesic nearest-point and farthest-point Voronoi diagrams, except for the short intervals of $n/\operatorname{polylog} n < m < n$ stated above.

For the geodesic order-$k$ Voronoi diagram, we analyze an asymptotically tight combinatorial complexity of the diagram for a simple polygon, which is $\Theta(k(m-k)+\min\{nk, n(m-k)\})$.

Other contributions of this paper are the algorithms for computing the topological structures of the geodesic nearest-point, order-$k$ and farthest-point Voronoi diagrams which take $O(m \log m \log^2 n)$, $O(k^2 m \log m \log^2 n)$ and $O(m \log m \log^2 n)$ time, respectively. These algorithms allow us to obtain a dynamic data structure for answering nearest or farthest point queries. In this problem, we are given a static simple $n$-gon $P$ and a dynamic point set $S \subseteq P$. We are allowed to insert points to $S$ and delete points from $S$. After processing updates, we are to find the point of $S$ nearest (or farthest) from a query point efficiently. This data structure requires $O(\sqrt{m}\log(n+m))$ query time and $O(\sqrt{m}\log m \log^2 n)$ update time, where $m$ is the number of points in $S$ at the moment.

## 1.1 Outline

Our algorithms for computing the geodesic nearest-point, higher-order and farthest-point Voronoi diagrams are based on a *polygon-sweep paradigm*. For the geodesic nearest-point and higher-order Voronoi diagrams, we fix a point $o$ on the boundary of the polygon and move another point $x$ from $o$ in clockwise direction along the boundary of the polygon. While $x$ moves along the boundary, we compute the Voronoi diagram of sites contained in the subpolygon bounded by the shortest path between $o$ and $x$ and the part of the boundary of $P$ from $o$ to $x$ in clockwise order. For the geodesic farthest-point Voronoi diagram, we sweep the polygon with a curve consisting of points equidistant from the geodesic center of the sites. The curve moves from the boundary towards the geodesic center. During the sweep, we gradually compute the diagram restricted to the region we have swept.

To achieve algorithms faster than the best known ones for $m \leq n/\operatorname{polylog} n$, we first compute the topological structure of a diagram instead of computing the diagram itself directly. The topological structure, which will be defined later, represents the adjacency of the Voronoi cells and has complexity smaller than the one of the complete Voronoi diagram. Once we have the topological structure, we can compute the complete Voronoi diagram in $O(T_1 + T_2 \log n)$ time, where $T_1$ is the combinatorial complexity of the complete Voronoi diagram and $T_2$ is the combinatorial complexity of the topological structure of the diagram.

We define four types of events where the topological structure changes. To handle each event, we compute a point equidistant from three points under the geodesic metric. There is no algorithm known for computing a point equidistant from three points efficiently, except an $O(n)$-time trivial algorithm. We present an $O(\log^2 n)$-time algorithm assuming that the data structure of Guibas and Hershberger [9] is constructed for $P$. This algorithm allows us to handle each event in $O(\operatorname{polylog}\{n, m\})$ time.

One application of an algorithm for computing the topological structure is a data structure for nearest (or farthest) point queries for a dynamic point set. To obtain this data structure, we apply the framework given by Bentley and Saxe [4]. We observe that we can find the Voronoi cell of the diagram containing a query point in $O(\log(n+m))$ time once we have the topological structure of the diagram.

# 2 Preliminaries

Let $P$ be a simple $n$-gon and $S$ be a set of $m$ points in $P$. For ease of description, we use $\mathsf{VD}[S]$, $k$-$\mathsf{VD}[S]$ and $\mathsf{FVD}[S]$ (or simply $\mathsf{VD}$, $k$-$\mathsf{VD}$ and $\mathsf{FVD}$ if they are understood in the

context) to denote the geodesic nearest-point, order-$k$ and farthest-point Voronoi diagrams of $S$ in $P$, respectively. We assume the *general position condition* that no vertex of $P$ is equidistant from two distinct sites of $S$ and no point of $P$ is equidistant from four distinct sites of $S$. This was also assumed in previous work [3, 11, 15] on geodesic Voronoi diagrams. This condition can be obtained by applying a slight perturbation of the positions of sites [6].

Consider any three points $x, y$ and $z$ in $P$. We use $\pi(x, y)$ to denote the shortest path (geodesic path) between $x$ and $y$ contained in $P$, and $d(x, y)$ to denote the geodesic distance between $x$ and $y$. Two geodesic paths $\pi(x, y)$ and $\pi(x, z)$ do not cross each other, but may overlap with each other. We call a point $x'$ the *junction* of $\pi(x, y)$ and $\pi(x, z)$ if $\pi(x, x')$ is the maximal common path of $\pi(x, y)$ and $\pi(x, z)$. Refer to Figure 1(a).

Given a point $p \in P$ and a closed set $A \subseteq P$, we slightly abuse the notation $\pi(p, A)$ to denote the shortest path contained in $P$ connecting $p$ and a point in $A$. Similarly, we abuse the notation $d(p, A)$ to denote the length of $\pi(p, A)$. It holds that $d(p, A) \leq d(p, q) + d(q, A)$ for any two points $p, q \in P$ and any closed set $A \subseteq P$.

We say a set $A \subseteq P$ is *geodesically convex* if $\pi(x, y) \subseteq A$ for any two points $x$ and $y$ in $A$. The *geodesic convex hull* of $S$ is the intersection of all geodesic convex sets containing $S$. The geodesic convex hull of a set of $m$ points can be computed in $O(n + m \log(n + m))$ time [9].

The *geodesic center* of a simple polygon $P$ is the point $c$ that minimizes $\max_{p \in P} d(c, p)$. The center is unique [16] and can be computed in $O(n)$ time [1]. Similarly, the geodesic center of $S$ can be defined as the point $c$ that minimizes $\max_{s \in S} d(c, s)$. It is known that the geodesic center of a set $S$ of $m$ points in $P$ coincides with the geodesic center of the geodesic convex hull of $S$ [3]. Therefore, we can compute the center of $S$ by computing the geodesic convex hull of $S$ and its center. This takes $O(n + m \log(n + m))$ time in total.

Due to lack of space, some of the proofs are omitted. All missing proofs can be found in the full version of this paper.

## 3 Computing the Geodesic Center of Points in a Simple Polygon

We first present an $O(\log^2 n)$-time algorithm for computing the geodesic center of three points contained in $P$, assuming that we have the data structure of Guibas and Hershberger [9, 10]. This algorithm will be used as a subprocedure for computing the geodesic Voronoi diagrams.

### 3.1 Computing the Geodesic Center of Three Points

Let $p_1, p_2$ and $p_3$ be three points in $P$, and let $c$ be the geodesic center of them. The geodesic convex hull of $p_1, p_2, p_3$ is bounded by $\pi(p_1, p_2), \pi(p_2, p_3)$, and $\pi(p_3, p_1)$. The geodesic convex hull may have complexity $\Omega(n)$, but its interior is bounded by at most three concave chains. This allows us to compute the geodesic center of it efficiently.

We first construct the data structure of Guibas and Hershberger [9, 10] for $P$ that allows us to compute the geodesic distance between any two points in $O(\log n)$ time. To compute $c$, we compute the shortest paths $\pi(p_1, p_2), \pi(p_2, p_3)$, and $\pi(p_3, p_1)$. Each shortest path has a linear size, but we can compute them in $O(\log n)$ time using the data structure of Guibas and Hershberger. Then we find a convex $t$-gon with $t \leq 6$ containing $c$ such that the geodesic path $\pi(x, p_i)$ has the same combinatorial structure for any point $x$ in the $t$-gon for each $i = 1, 2, 3$. To find such a convex $t$-gon, we apply two-level binary search. Then we can compute $c$ directly in constant time inside the $t$-gon.

**The data structure given by Guibas and Hershberger.** Guibas and Hershberger [9, 10] gave a data structure of linear size that enables us to compute the geodesic distance between

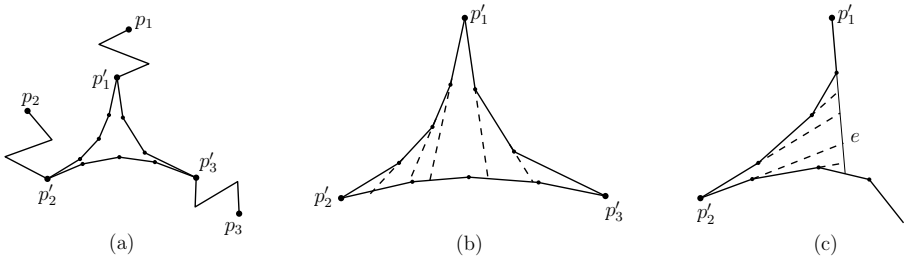



**Figure 1** (a) $p'_i$ is the junction of $\pi(p_i, p_j)$ and $\pi(p_i, p_k)$ for three distinct indices $i, j$ and $k$ in $\{1, 2, 3\}$. (b) The subdivision of $\triangle$ with respect to $p'_1$. (c) The subdivision of $e$ with respect to $p'_2$.

any two query points lying inside $P$ in $O(\log n)$ time. We call this structure the *shortest path data structure*. They showed that this data structure can be constructed in $O(n)$ time.

In the preprocessing, they compute a number of shortest paths such that for any two points $p$ and $q$ in $P$, the shortest path $\pi(p, q)$ consists of $O(\log n)$ subchains of precomputed shortest paths and $O(\log n)$ additional edges. In the query algorithm, they find such subchains and edges connecting them in $O(\log n)$ time. Then the query algorithm returns the shortest path between two query points represented as a binary tree of height $O(\log n)$ [10]. Therefore, we can apply binary search on the vertices of the shortest path between any two points.

**Computing the geodesic center of three points: two-level binary search.** Let $\triangle$ be the geodesic convex hull of $p_1, p_2$ and $p_3$. The geodesic center $c$ of the three points is the geodesic center of $\triangle$ [3], thus is contained in $\triangle$. If the center lies on the boundary of $\triangle$, we can compute it in $O(\log n)$ time since it is the midpoint of two points. So, we assume that the center lies in the interior of $\triangle$. Let $p'_i$ be the junction of $\pi(p_i, p_j)$ and $\pi(p_i, p_k)$ for three distinct indices $i, j$ and $k$ in $\{1, 2, 3\}$. See Figure 1(a).

We use the following lemmas to apply two-level binary search.

▶ **Lemma 1** ([9]). *We can compute the junctions $p'_1, p'_2$ and $p'_3$ in $O(\log n)$ time.*

▶ **Lemma 2** ([5]). *Given a point $p \in \triangle$ and a direction, we can find the first intersection point of the boundary of $\triangle$ with the ray from $p$ in the direction in $O(\log n)$ time.*

**The first level.** Imagine that we subdivide $\triangle$ into $O(n)$ cells with respect to $p'_1$ by extending the edges of $\pi(p'_1, p'_2) \cup \pi(p'_1, p'_3)$ towards $\pi(p'_2, p'_3)$. See Figure 1(b). The extensions of the edges can be sorted in the order of their endpoints appearing along $\pi(p'_2, p'_3)$. Consider the subdivision of $\triangle$ by the extensions, and assume that we can determine which side of a given extension in $\triangle$ contains $c$ in $T(n)$ time. Then we can compute the cell of the subdivision containing $c$ in $O(T(n) \log n)$ time by applying binary search on the extensions. Note that any point $x$ in the same cell has the same combinatorial structure of $\pi(x, p_1)$ (and $\pi(x, p'_1)$).

We also do this for $p'_2$ and $p'_3$. Then we have three cells whose intersection contains $c$. Let $D$ be the intersection of these three cells. We can find $D$ in constant time by the the fact that $D$ is a convex polygon with at most six edges from extensions of the cells.

We do not subdivide $\triangle$ explicitly. Because we have $\pi(p'_1, p'_2)$ and $\pi(p'_1, p'_3)$ in binary trees of height $O(\log n)$, we can apply binary search on the extensions of the edges of the geodesic paths without subdividing $\triangle$ explicitly. In this case, during the binary search, we compute the extension of a given edge of $\pi(p'_1, p'_2) \cup \pi(p'_1, p'_3)$ using Lemma 2 in $O(\log n)$ time.

There is a vertex $p$ on the boundary of $\triangle$ such that for any point $x$ contained in $D$ we have $d(p_1, x) = d(p_1, p) + \|p - x\|$, where $\|p - x\|$ is the Euclidean distance between $p$ and $x$.

Moreover, we already have $p$ from the computation of the cell containing $c$ in the subdivision with respect to $p'_1$. The same holds for $p_2$ and $p_3$. Therefore, we can compute the point $c$ that minimizes the maximum of $d(c, p_1), d(c, p_2)$ and $d(c, p_3)$ in constant time inside $D$.

Therefore, we have the following lemma.

▶ **Lemma 3.** *Assuming that we can determine which side of an extension in $\triangle$ contains $c$ in $T(n)$ time, we can compute the geodesic center $c$ in $O((T(n) + \log n) \log n)$ time.*

**The second level.** In the second level binary search, we determine which side of an extension $e$ in $\triangle$ contains $c$. Without loss of generality, we assume that $e$ comes from the subdivision with respect to $p'_1$. Then $\pi(p_1, x)$ has the same combinatorial structure for any point $x \in e$.

This subproblem was also considered in previous works on computing the geodesic center of a simple polygon [1, 16]. They first compute the point $c_e$ in $e$ that minimizes $\max_{p \in P} d(p, c_e)$, that is, the geodesic center of the polygon restricted to $e$. Based on $c_e$ and its farthest point, Pollack et al. [16] presented a way to decide which side of $e$ contains the geodesic center of the polygon in constant time. However, to compute $c_e$, they spend $O(n)$ time.

In our problem, we can do this in logarithmic time using the fact that the interior of $\triangle$ is bounded by at most three concave chains. By this fact, there are two possible cases: $c_e$ is an endpoint of $e$, or $c_e$ is equidistant from $p_1$ and $p_i$ for $i = 2$ or 3. We compute the point on $e$ equidistant from $p_1$ and $p_2$, and the point on $e$ equidistant from $p_1$ and $p_3$. Then we find the point $c_e$ among the two points and the two endpoints of $e$. In the following, we show how to compute the point on $e$ equidistant from $p_1$ and $p_2$ if it exists. The point on $e$ equidistant from $p_1$ and $p_3$ can be computed analogously.

Observe that $e$ can be subdivided into $O(n)$ disjoint line segments by the extensions of the edges of $\pi(p'_2, v_1) \cup \pi(p'_2, v_2)$ towards $e$, where $v_1$ and $v_2$ are endpoints of $e$. See Figure 1(c). For any point $x$ in the same line segment, $\pi(p_2, x)$ has the same combinatorial structure.

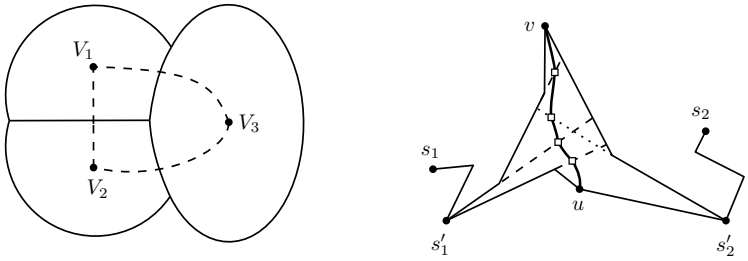
There is at most one point on $e$ equidistant from $p_1$ and $p_2$. (For a proof, see the full paper.) Thus we can apply binary search on the line segments in the subdivision of $e$. As we did before, we do not subdivide $e$ explicitly. Instead, we use the binary trees representing $\pi(p'_2, v_1)$ and $\pi(p'_2, v_2)$. For a point $x$ in $e$, by comparing $d(p_1, x)$ and $d(p_2, x)$, we can determine which part of $x$ on $e$ contains the point equidistant from $p_1$ and $p_2$ in constant time. In this case, we can compute the extension from an edge towards $e$ in constant time since $e$ is a line segment. Thus, we complete the binary search in $O(\log n)$ time.

Therefore, we can compute $c_e$ in $O(\log n)$ time and determine which side of $e$ in $\triangle$ contains $c$ in the same time using the method of Pollack et al [16]. The following lemma summarizes this section.

▶ **Lemma 4.** *Given any three points $p_1, p_2$ and $p_3$ contained in a simple $n$-gon $P$, the geodesic center of $p_1, p_2$ and $p_3$ can be computed in $O(\log^2 n)$ time after the shortest path data structure for $P$ is constructed in linear time.*

Similarly, we can compute a point equidistant from any three points in $P$ (or, two points and a line segment). They are used as subprocedures for computing the Voronoi diagrams. Note that the geodesic center of three points may not be equidistant from all of them. Moreover, there may be an infinite number of points equidistant from the three points (or, two points and a line segment). In this case, we compute the one closest to them.

▶ **Lemma 5.** *Given any three points contained in a simple $n$-gon, we can compute the closest equidistant point from them under the geodesic metric in $O(\log^2 n)$ time if it exists.*




(a) (b)

**Figure 2** (a) The adjacency graph of a Voronoi diagram. (b) The number of edges in the common boundary of two adjacent Voronoi cells is bounded by the total complexity of $\pi(s_1', v)$, $\pi(s_1', u)$, $\pi(s_2', v)$ and $\pi(s_2', u)$, where $u$ and $v$ are endpoints of the common boundary.

▶ **Lemma 6.** *Given any two points and any line segment contained in a simple $n$-gon, we can compute the closest equidistant point from them under the geodesic metric in $O(\log^2 n)$ time if it exists.*

Combining the result in this subsection with the algorithms for computing the center of points in the plane [12] and computing the geodesic center of a simple polygon [16], we can compute the geodesic center of $m$ points contained in $P$. (For details, see the full version.)

▶ **Theorem 7.** *The geodesic center of $m$ points contained in a simple polygon with $n$ vertices can be computed in $O(m \log m \log^2 n)$ time after the shortest path data structure for the simple polygon is constructed.*

## 4 Topological Structures of Voronoi Diagrams

In this section, we define the topological structure of Voronoi diagrams and show how to compute the complete Voronoi diagrams from their topological structures. The topological structure of a Voronoi diagram represents the adjacency of their Voronoi cells.

The common boundary of any two adjacent Voronoi cells is connected for the nearest-point and farthest-point Voronoi diagrams of point sites in a simple polygon [2, 3]. Similarly, it can be shown that this also holds for the higher-order Voronoi diagram of point sites in a simple polygon.

The topological structure of $k$-VD is defined as follows for $1 \leq k \leq m-1$. Imagine that we apply vertex suppression for every degree-2 vertex of the Voronoi diagram while preserving the topology of the Voronoi diagram. Vertex suppression of a vertex $v$ of degree 2 is the operation of removing $v$ and adding an edge connecting the two neighbors of $v$. We call the dual of this graph the *adjacency graph* of the Voronoi diagram. See Figure 2(a). It represents the topological structure of the Voronoi diagram. The adjacency graph is a planar graph with complexity $O(k(m-k))$, because the number of degree-1 and degree-3 vertices of $k$-VD is $O(k(m-k))$ [11].

Assume that we have the adjacency graph of the Voronoi diagram together with the exact positions of the degree-1 and degree-3 vertices of the diagram. Consider two Voronoi cells $V_1$ and $V_2$ which are adjacent to each other. Each Voronoi cell is defined by $k$ sites, but any two adjacent Voronoi cells share $k-1$ sites. Let $s_1$ and $s_2$ be the two sites defining $V_1$ and $V_2$, respectively, which are not shared by them. There are two Voronoi vertices $v$ and $u$ defined by a triple $(s_1, s_2, s)$ and a triple $(s_1, s_2, s')$ of sites defining $v$ and $u$, respectively, for some

sites $s, s'$. Each Voronoi edge in the common boundary of $V_1$ and $V_2$ is a part of the bisector of $s_1$ and $s_2$ lying between $v$ and $u$. See Figure 2(b).

To compute the Voronoi edges in the common boundary of $V_1$ and $V_2$, we consider the geodesic paths $\pi(s'_i, v)$ and $\pi(s'_i, u)$ for $i = 1, 2$, where $s'_i$ is the junction of $\pi(s_i, v)$ and $\pi(s_i, u)$. Then for any vertex $x$ in $\pi(s'_1, v) \cup \pi(s'_1, u)$, there exists a point $q$ in the bisector of $s_1$ and $s_2$ lying between $v$ and $u$ such that $\pi(s_1, q)$ and $\pi(s_1, x)$ have the same combinatorial structure. The same holds for $s_2$. Thus, the number of edges in the common boundary is bounded by the total complexity of $\pi(s'_1, v) \cup \pi(s'_1, u)$ and $\pi(s'_2, v) \cup \pi(s'_2, u)$. Thus, we may compute the geodesic paths explicitly and consider every edge of the geodesic paths.

Therefore, we can compute the common boundary of two adjacent Voronoi cells in time linear to its complexity plus $O(\log n)$. This leads to $O(T_1 + T_2 \log n)$ time for computing the complete Voronoi diagram from its topological structure, where $T_1$ is the complexity of the complete Voronoi diagram and $T_2$ is the complexity of the adjacency graph.

▶ **Lemma 8.** *We can compute the complete Voronoi diagram of $m$ points in a simple polygon with $n$ vertices in $O(T_1 + T_2 \log n)$ time once its adjacency graph, and the degree-1 and degree-3 vertices at their places are given, where $T_1$ is the complexity of the complete Voronoi diagram and $T_2$ is the complexity of the adjacency graph.*

Therefore, in the following, we focus on computing the topological structure of VD, $k$-VD and FVD, that is, the adjacency graphs of them with degree-3 vertices at their places.

## 5 The Geodesic Nearest-Point Voronoi Diagram

Fortune [7] presented an $O(m \log m)$-time algorithm to compute the nearest-point Voronoi diagram of $m$ points in the plane by sweeping the plane with a horizontal line from top to bottom. During the sweep, they compute a part of the Voronoi diagram of sites lying above the horizontal line, which finally becomes the complete Voronoi diagram in the end of the sweep. To do this, they define two types of events and handle $O(m)$ events in total. Each event can be handled in $O(\log m)$ time, which leads to $O(m \log m)$ total running time.

In our case, we sweep the polygon with a geodesic path $\pi(o, x)$ for a fixed point $o$ on the boundary of $P$ and a point $x$ moving along the boundary of $P$ from $o$ in clockwise direction. The point $x$ is called the *sweep point*. If we compute all $O(n + m)$ degree-1, degree-2 and degree-3 vertices of the Voronoi diagram during the sweep, we may not achieve the running time better than $O((n + m) \log(n + m))$. The key to improve the running time is to compute the topological structure of the Voronoi diagram first which consists of the degree-1 and degree-3 vertices of the Voronoi diagram and the adjacency graph of Voronoi cells. Then we construct the complete Voronoi diagram, including degree-2 vertices, from its topological structure using Lemma 8.

Let $o$ be an arbitrary point on $\partial P$, where $\partial P$ denotes the boundary of $P$. Consider the sweep point $x$ that moves from $o$ along $\partial P$ in clockwise order. We use $P(x)$ to denote the subpolygon of $P$ bounded by $\pi(o, x)$ and the part of $\partial P$ from $o$ to $x$ in clockwise order. Note that $P(x)$ is weakly simple. See Figure 3. As $x$ moves along $\partial P$, $P(x)$ does not decrease. That is, $P(x_1) \subseteq P(x_2)$ for any two points $x_1$ and $x_2$ on $\partial P$ such that $x_1$ comes before $x_2$ from $o$ in clockwise order.

For a site $s \in P(x)$, let $R_s(x)$ be the region $\{p \in P(x) \mid d(p, s) \leq d(p, \pi(o, x))\}$. By definition, $R_s(x)$ does not decrease as $x$ moves along $\partial P$ in clockwise order.

▶ **Lemma 9.** *$R_s(x)$ is connected.*

**Proof.** To prove the lemma, we show that $\pi(p,s) \subseteq R_s(x)$ for any point $p \in R_s(x)$. This implies that $R_s(x)$ is connected because $s$ is contained in $R_s(x)$ if $s \in P(x)$ by definition. Let $p$ be a point in $R_s(x)$. Consider a point $r \in \pi(p,s)$. We have $d(r,s) = d(p,s) - d(p,r)$. Moreover, we have $d(p,\pi(o,x)) - d(p,r) \leq d(r,\pi(o,x))$. Since $p \in R_s(x)$, it holds that $d(p,s) \leq d(p,\pi(o,x))$. Thus, $d(r,s) \leq d(r,\pi(o,x))$. Therefore, $r$ is in $R_s(x)$, and therefore, $R_s(x)$ is connected. ◀

We say that a subset $A$ of $P$ is *weakly monotone* with respect to a geodesic path $\pi'$ if the intersection of $\pi(p,\pi')$ with $A$ is connected for any point $p \in A$.

▶ **Lemma 10.** *The boundary of $R_s(x)$ consists of one polygonal chain of $\partial P$ and a simple curve whose both endpoints lie on $\partial P(x)$ unless $s \in \pi(o,x)$. Moreover, the simple curve is weakly monotone with respect to $\pi(o,x)$.*

**Proof.** For any point $p \in R_s(x)$, the geodesic ray from $p$ in direction opposite to the edge of $\pi(p,\pi(o,x))$ incident to $p$ is contained in $R_s(x)$. This is because $d(r,\pi(o,x)) = d(p,\pi(o,x)) + d(p,r) \geq d(p,s) + d(p,r) \geq d(r,s)$ by triangle inequality for any point $r$ in the geodesic ray. The lemma follows from this fact and Lemma 9. ◀

Now we consider the union of $R_s(x)$ for every site $s$ in $P(x)$ and denote it by $R(x)$. The dashed region in Figure 3(a) is $R(x)$. Note that for any point $p \in P(x)$ and any site $s \in S$ lying outside of $P(x)$, it holds that $d(p,\pi(o,x)) < d(p,s)$. This implies that for any point $p \in R(x)$, the nearest site of $p$ is in $P(x)$. Therefore, to compute VD restricted to $R(x)$, we do not need to consider the sites lying outside of $P(x)$.

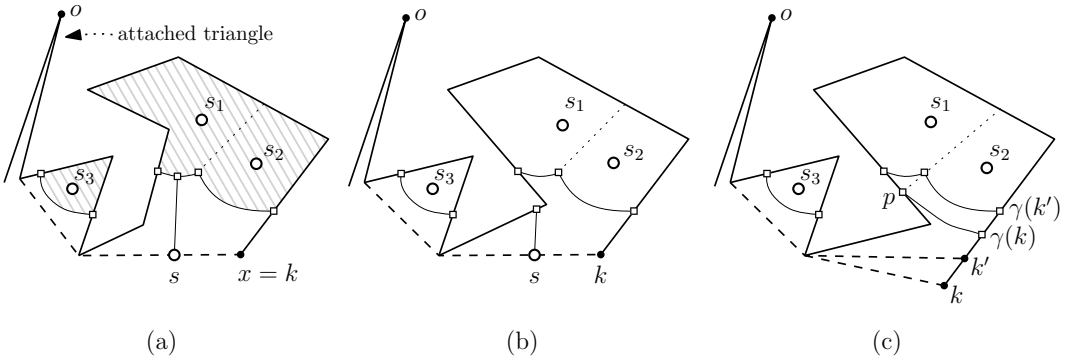
▶ **Corollary 11.** *The closure of $P(x) \setminus R(x)$ is weakly simple.*

▶ **Corollary 12.** *Each connected component of $R(x)$ consists of a polygonal chain of $\partial P$ and a connected simple curve with endpoints on $\partial P$ unless $\pi(o,x)$ contains some site. Moreover, the union of such curves is weakly monotone with respect to $\pi(o,x)$ at any time.*

We maintain the nearest-point Voronoi diagram of sites contained in $P(x)$ restricted to $R(x)$ while $x$ moves along $\partial P$. However, after the sweep, $R(x)$ does not coincide with $P$. This means that we cannot obtain VD in this way. To solve this problem, we attach a long and very thin triangle to $\partial P$ to make a bit larger simple polygon $P'$, as illustrated in Figure 3, so that once we finish the sweep (when the sweep point returns back to $o$) in $P'$ we have the complete Voronoi diagram in $P$. (The triangle has height of the diameter of $P$.)

**The beach line and the breakpoints.** There are $O(m)$ connected components of $R(x)$ each of whose boundaries consists of a polygonal chain of $\partial P$ and a connected simple curve. The *beach line* is defined to be the union of $O(m)$ simple curves of $R(x)$. It has properties similar to those of the beach line of the Euclidean Voronoi diagram. It consists of $O(n+m)$ hyperbolic or linear arcs. We do not maintain them explicitly because the complexity of the sequence is too large for our purpose. Instead, we maintain the combinatorial structure of the beach line using $O(m)$ space as follows.

A point on the beach line is called a *breakpoint* if it is equidistant from two distinct sites. Each breakpoint moves and traces out some Voronoi edge as the sweep point moves along $\partial P'$. We represent each breakpoint symbolically. That is, a breakpoint is represented as the pair of sites equidistant from it. We say that such a pair *defines* the breakpoint. Given the pair defining a breakpoint and the position of the sweep point, we can find the exact position of the breakpoint using Lemma 6 in $O(\log^2 n)$ time.




(a) (b) (c)

**Figure 3** (a) The site event defined by $s$ with key $k$. Two (degenerate) breakpoints appear in $B(k)$. (b) Two (degenerate) endpoints appear in $B(k)$. (c) The vanishing event defined by the breakpoint of $(s_1, s_2)$ with key $k$. The endpoint defined by $s_1$ and the breakpoint merge into the endpoint defined by $s_2$.

Additionally, we consider the endpoints of $O(m)$ connected curves of the beach line. We simply call them the *endpoints* of the beach line. For an endpoint $p$, there is the unique site $s$ with $d(p, s) = d(p, \pi(o, x))$. We say that $s$ *defines* $p$.

By Corollary 11 and Corollary 12, we can define the order of these breakpoints and these endpoints. Let $B(x) = \langle \beta_1, \ldots, \beta_{m'} \rangle$ be the sequence of the breakpoints and the endpoints of the beach line sorted in clockwise order along the boundary of $P(x) \setminus R(x)$ with $m' = O(m)$. We maintain $B(x)$ instead of all arcs on the beach line. As $x$ moves along $\partial P'$, the sequence $B(x)$ changes.

While maintaining $B(x)$, we compute the adjacency graph of $\mathsf{VD}$ together with the degree-1 and degree-3 vertices of $\mathsf{VD}$ restricted to $R(x)$. (Recall that a cell of $\mathsf{VD}$ restricted to $R(x)$ is associated with a site in $P(x)$.) Specifically, when a new breakpoint defined by a pair $(s, s')$ of sites is added to $B(x)$, we add an edge connecting the node for $s$ and the node for $s'$ into the adjacency graph.

**Events.** We have four types of events: site events, circle events, vanishing events, and merging events. Every event corresponds to a *key*, which is a point on the boundary of $P'$. We maintain the events with respect to their keys sorted in clockwise order from $o$ along the boundary. Given a sorted sequence of $\nu$ events for $\nu \in \mathbb{N}$, we can insert a new event in $O(\log \nu)$ time since we are given each point on the boundary of $P'$ together with the edge of $P'$ where it lies. The event occurs when the sweep point $x$ passes through the corresponding key. The sequence $B(x)$ changes only when $x$ passes through the key of an event.

The definitions of the first two events are similar to the ones in Fortune's algorithm. Each site $s$ in $S$ defines a *site event*. The key $k$ of the site event defined by $s$ is the point on $\partial P'$ closest to $o$ among the points $x'$ with $s \in \pi(o, x')$. When the sweep point passes through $k$, the site $s$ appears on the beach line. Moreover, new breakpoints defined by $s$ and some other site $s'$, or new endpoints defined by $s$ appear on $B(x)$. See Figure 3(a) and (b).

The other events are defined by a pair of consecutive points in $B(x)$ or a single breakpoint in $B(x)$. Before the sweep point reaches the key of an event, the points in the pair defining the event may become non-consecutive, or the breakpoint defining the event may disappear from $B(x)$ due to the changes of $B(\cdot)$. In this case, we say that the event is *invalid*. Otherwise, we say that the event is *valid*.

A pair $(\beta_1, \beta_2)$ of consecutive breakpoints in $B(x)$ defines a *circle event* if a point $c$ equidistant from $s_1, s_2$ and $s_3$ exists, where $(s_1, s_2)$ and $(s_2, s_3)$ are two pairs of sites defining

$\beta_1$ and $\beta_2$, respectively. The key $k$ of this circle event is the point on $\partial P'$ closest to $o$ among the points $x'$ with $d(c, \pi(o, x')) = d(c, s_1)(= d(c, s_2) = d(c, s_3))$. Assume that this event is valid when $x$ passes through $k$. At that time, $c$ appears on the beach line. Moreover, $\beta_1$ and $\beta_2$ disappear from the beach line, and a new breakpoint defined by $(s_1, s_3)$ appears on the beach line. (One may think that $\beta_1$ and $\beta_2$ merge into the breakpoint defined by $(s_1, s_3)$.)

Each breakpoint $\beta$ in $B(x)$ defines a *vanishing event*. Let $(s_1, s_2)$ be the pair of sites defining $\beta$. Consider two points on $\partial P'$ equidistant from $s_1$ and $s_2$. We observe that exactly one of the two points lies outside of $R(x)$. We denote it by $p$. See Figure 3(c). The key $k$ of the vanishing event is the point on $\partial P'$ closest to $o$ among the points $x'$ with $d(p, \pi(o, x')) = d(p, s_1)$. Assume that this event is valid when $x$ passes through $k$. Then, $\beta$ traces out a Voronoi edge and reaches $p$, which is a degree-1 Voronoi vertex. Moreover, $B(x)$ changes accordingly as follows. Right before $x$ reaches $k$, an endpoint defined by $s_1$ or $s_2$ is a neighbor of $\beta$ in $B(x)$. (Otherwise, the beach line is not weakly monotone.) Without loss of generality, we assume that an endpoint of $s_1$ is a neighbor of $\beta$. When $x$ reaches the key, this endpoint and $\beta$ disappear from $B(x)$, and an endpoint defined by $s_2$ appears in $B(x)$. (One may think that this endpoint and $\beta$ merge into the endpoint defined by $s_2$.)

A pair $(\beta_1, \beta_2)$ of consecutive endpoints in $B(x)$ defines a *merging event* if $\beta_1$ and $\beta_2$ are endpoints of different connected curves of the beach line. Let $s_1$ and $s_2$ be the sites defining $\beta_1$ and $\beta_2$, respectively. Without loss of generality, assume that $\beta_1$ comes before $\beta_2$ from $o$ in clockwise order. Let $p$ be the (unique) point equidistant from $s_1, s_2$ that comes after $\beta_1$ and before $\beta_2$ from $o$ along $\partial P'$ in clockwise order. The key $k$ of the merging event is the point on $\partial P'$ closest to $o$ among the points $x'$ with $d(p, \pi(o, x')) = d(p, s_1)$. Assume that this event is valid when $x$ passes through $k$. Then, as the sweep point $x$ moves along $\partial P'$, $\beta_1$ and $\beta_2$ are closer and finally meet each other at $p$. This means that the two connected curves, one containing $\beta_1$ and the other containing $\beta_2$, merge into one connected curve. At this time, $\beta_1$ and $\beta_2$ merge into a breakpoint defined by $(s_1, s_2)$.

## 5.1 An algorithm

Initially, $B(x) = B(o)$ is empty, so there is no circle, vanishing, or merging event. We compute the keys of all site events in advance. For each site, we compute the key corresponding to its event in $O(\log n)$ time [8].

As $B(x)$ changes, we obtain new pairs of consecutive breakpoints or endpoints, and new breakpoints. Then we compute the key of each event in $O(\log^2 n)$ time using the following lemmas and Lemma 5. In addition, as $B(x)$ changes, some event becomes invalid. Once an event becomes invalid, it does not become valid again. Thus we discard an event if it becomes invalid. Therefore, the number of events we have is $O(|B(x)|)$ at any time.

▶ **Lemma 13.** *Given a point $p$ in $P'$ and a distance $r \in \mathbb{R}$, the point on $\partial P'$ closest to $o$ among the points $x'$ with $d(p, \pi(o, x')) = r$ can be found in $O(\log^2 n)$ time.*

▶ **Lemma 14.** *For any two sites $s_1$ and $s_2$ in $P'$, we can compute the two points equidistant from $s_1$ and $s_2$ lying on the boundary of $P'$ in $O(\log^2 n)$ time.*

**Handling site events.** To handle a site event defined by a site $s$ with key $k$, we do the following. By definition, the site $s$ appears on the beach line when the sweep point $x$ passes through $k$. Thus, two breakpoints defined by $(s, s')$ for other sites $s'$ (or two endpoints defined by $s$) appear on $B(k)$. See Figure 3(a) and (b). We find the positions of them in $B(x)$ and update $B(x)$ by adding them using Lemma 15.

▶ **Lemma 15.** *We can obtain $B(k)$ from $B(k')$ in $O(\log^2 n \log |B(k')|)$ time, where $k'$ is the key previous to $k$.*

Adding two breakpoints or two endpoints to $B(x)$ makes a constant number of new pairs of consecutive breakpoints or endpoints, which define circle or merging events. This also makes a constant number of new vanishing events. We compute the key for each event in $O(\log^2 n)$ time and add the events to the event queue in $O(\log |B(x)|)$ time. Recall that the number of events we have is $O(|B(x)|)$ at any time. Therefore, each site event can be handled in $O(\log^2 n \log |B(x)|)$ time.

**Handling the other events.** Let $k$ be the key of a circle, vanishing, or merging event. For a circle event, two breakpoints defining the event disappear from the beach line, and a new breakpoint appears. For a vanishing event, the breakpoint defining the event and its neighboring endpoint disappear from the beach line, and a new endpoint appears. For a merging event, two endpoints defining the event are replaced with a new breakpoint. In any case, we can update $B(x)$ in $O(\log |B(x)|)$ time.

After updating $B(x)$, we have a constant number of new pairs of consecutive breakpoints or endpoints, and a constant number of new breakpoints. We compute the keys of events defined by them in $O(\log^2 n)$ time.

**Analysis.** During the sweep, we handle $O(m)$ events in total. Each event can be processed in $O(\log^2 n \log m)$ time since the length of $B(x)$ is $O(m)$ at any time. Thus we have the following lemma and theorem.

▶ **Lemma 16.** *After computing the shortest path data structure for $P'$ and the shortest path map for $P'$ from the point $o$, we can compute the topological structure of VD in $O(m \log m \log^2 n)$ time using $O(m)$ space (excluding the two data structures).*

▶ **Theorem 17.** *Given a set of $m$ point sites contained in a simple polygon with $n$ vertices, we can compute the geodesic nearest-point Voronoi diagram of the sites in $O(n + m \log m \log^2 n)$ time using $O(n + m)$ space.*

## 6 The Geodesic Higher-order Voronoi Diagram

In this section, we first present an asymptotically tight combinatorial complexity of the geodesic higher-order Voronoi diagram of points. Then we present an algorithm to compute the diagram by applying the polygon-sweep paradigm introduced in Section 5. In the plane, Zavershynskyi and Papadopoulou [17] presented a plane-sweep algorithm to compute the higher-order Voronoi diagram. We use their approach together with our approach in Section 5. In the following, we assume that $1 \leq k \leq m - 1$.

### 6.1 The Complexity of the Diagram inside a Simple Polygon

Liu and Lee [11] presented an asymptotically tight complexity of the geodesic higher-order Voronoi diagram of points in a polygonal domain with holes, which is $\Theta(k(m - k) + nk)$. However, it is not tight for a simple polygon.

Figure 4 shows an example that the order-$k$ Voronoi diagram has complexity of $\Omega(k(m - k) + \min\{nk, n(m - k)\})$. We construct a simple polygon $P$ and a set of sites with respect to a reference point $o$. Let $\ell_1$ be a sufficiently long horizontal line segment whose right endpoint is $o$ and $\ell_2$ be a sufficiently long line segment with a positive slope whose left endpoint is

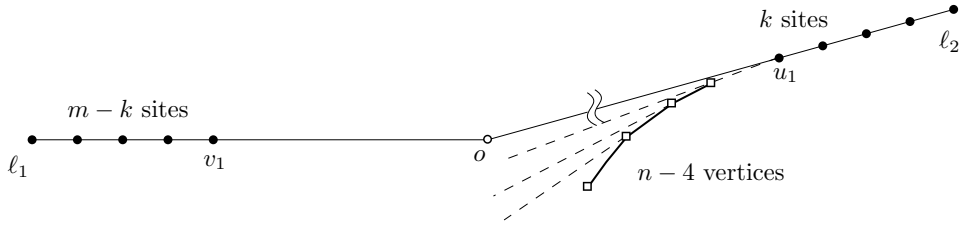



**Figure 4** An example that the order-$k$ Voronoi diagram has complexity of $\Omega(k(m-k) + \min\{nk, n(m-k)\})$.

$o$. ($\ell_1$ and $\ell_2$ are not parts of the simple polygon, but they are auxiliary line segments to locate the points.) We put $m-k$ sites on $\ell_1$ such that the sites are sufficiently close to each other. Similarly, we put $k$ sites on $\ell_2$ such that sites are sufficiently close to each other and $\|v_{m-k} - o\| \ll \|u_1 - o\|$, where $v_i$'s are sites on $\ell_1$ sorted along $\ell_1$ from $o$ and $u'_j$'s are sites on $\ell_2$ sorted along $\ell_2$ from $o$, for $i = 1, \ldots, m-k$ and $j = 1, \ldots, k$.

As a part of the boundary of $P$, we put a concave and $y$-monotone polygonal curve with $n-4$ vertices below $\ell_2$ such that the highest point of the curve is sufficiently close to $u_1$. Then we put another four vertices of $P$ sufficiently far from the sites such that $P$ contains all sites and there is no simple polygon containing more Voronoi vertices than $P$ contains.

In the full version, we show that $k$-VD of this example has complexity of $\Omega(k(m-k) + \min\{nk, n(m-k)\})$. Moreover, we show that $k$-VD for any simple polygon and any point set has complexity of $O(k(m-k) + \min\{nk, n(m-k)\})$.

▶ **Lemma 18.** *The geodesic order-$k$ Voronoi diagram of $m$ points inside a simple polygon with $n$ vertices has complexity of* $\Theta(k(m-k) + \min\{nk, n(m-k)\})$.

## 6.2 Computing the Topological Structure of the Diagram

Due to lack of space, we present only an outline of our algorithm. For details, refer to the full version. The algorithm is similar to the one in Section 5. Recall that for a site $s \in P(x)$, the boundary of $R_s(x)$ consists of one polygonal chain of $\partial P$ and a simple curve with endpoints on $\partial P$. We call the simple curve the *wave-curve* for $s$. The wave-curve for $s$ is monotone with respect to $\pi(o, x)$.

We say that a point $p \in P(x)$ lies *above* a curve if $\pi(p, \pi(o, x))$ intersects the curve. We use $L_i(x)$ to denote the region of $P(x)$ consisting of all points lying above at least $i$ wave-curves for sites contained in $P(x)$. The *$i$th-level* of the arrangement of wave-curves of sites contained in $P(x)$ is defined to be the boundary of $L_i(x)$ excluding $\partial P$. The $i$th-level for each $i$ is weakly monotone with respect to $\pi(o, x)$ and consists of at most $m$ simple curves with endpoints lying on $\partial P$.

$k$-VD$[S]$ restricted to $L_k(x)$ coincides with $k$-VD$[S \cap P(x)]$ restricted to $L_k(x)$. Therefore, we can obtain the topological structure of $k$-VD$[S]$ by maintaining the topological structure of the $k$th-level of the arrangement of wave-curves. In addition to this, we maintain the topological structures of the $i$th-levels of the arrangement for all $i < k$ to detect the changes to the topological structure of the $k$th-level.

There are four types of events; site events, circle events, vanishing events and merging events. The definitions of the event types are analogous to the ones in Section 5. We can handle the site and circle events in a way similar to the one for the Euclidean order-$k$ Voronoi diagram given by Zavershynskyi and Papadopoulou [17]. We can handle the other events in a way similar to the one in Section 5.

▶ **Theorem 19.** *Given a set of $m$ points contained in a simple polygon with $n$ vertices, we can compute the geodesic order-$k$ Voronoi of the points in $O(k^2 m \log m \log^2 n + \min\{nk, n(m-k)\})$ time using $O(n + km)$ space.*

## 7 The Geodesic Farthest-Point Voronoi Diagram

We present an outline of our algorithm for computing the topological structure of FVD. For details, refer to the full version. Aronov et al. [3] showed that FVD forms a tree whose root is the geodesic center $c$ of the sites. They first compute $c$ and the geodesic convex hull of the sites. Then they compute FVD restricted to the boundary of the polygon. Then they compute the edges of FVD towards the geodesic center by a *reverse geodesic sweep method*. We follow the framework of the algorithm by Aronov et al. [3], but we can achieve a faster algorithm for $m \leq n/\log^2 n$ by applying their approach to compute the topological structure of FVD.

In the reverse geodesic sweep, a *sweep line* is a simple curve consisting of points equidistant from $c$. Let $B$ be the (circular) sequence of the sites that have their Voronoi cells on the sweep line in clockwise order. As an exception, in the initial state, we set $B$ to be the sequence of the sites whose Voronoi cells appear on $\partial P$.

No site, vanishing, or merging event occurs during the sweep because FVD forms a tree. So we handle only circle events. Every triple of consecutive sites in $B$ defines a circle event. The key defined by a triple $(s_1, s_2, s_3)$ is $d(c, c')$ for the point $c'$ equidistant from $s_1, s_2$ and $s_3$. We can compute the key of each triple of consecutive sites in $B$ in $O(\log^2 n)$ time.

▶ **Lemma 20.** *We can compute the topological structure of FVD in $O(m \log m \log^2 n)$ or $O(n + m \log m + m \log^2 n)$ time after computing the shortest path data structure for $P$.*

▶ **Theorem 21.** *Given a set of $m$ points contained in a simple polygon with $n$ vertices, we can compute the geodesic farthest-point Voronoi diagram of the points in $O(n + m \log m + m \log^2 n)$ time using $O(n + m)$ space.*

## 8 Dynamic Data Structures for Nearest or Farthest Point Queries

We showed that the topological structure of a Voronoi diagram can be computed without considering the whole polygon. The topological structure can be used for data structures for answering nearest or farthest point queries for dynamic point sets.

We apply the framework given by Bentley and Saxe [4]. At all times, we maintain $O(\sqrt{m})$ disjoint sets each of which consists of $O(\sqrt{m})$ points. For each set, we compute the topological structure of the nearest-point (or farthest-point) Voronoi diagram of the points in the set.

For a nearest (or farthest) point query, we simply find the Voronoi cells containing the query point for $O(\sqrt{m})$ Voronoi diagrams. Then we have $O(\sqrt{m})$ candidates for the nearest (or farthest) point from the query point. We find the nearest (or farthest) point from the query point directly among them. Inside a simple polygon, the complexity of each complete Voronoi diagram is $O(n + \sqrt{m})$. Thus each update takes $O(n + \sqrt{m}\,\mathrm{polylog}\{n, m\})$ time if we maintain the complete Voronoi diagrams. This is why we compute the topological structures of the Voronoi diagrams.

In the full version of this paper, we show how to find the Voronoi cell containing a query point using the topological structure of a Voronoi diagram. The key idea is to approximate the Voronoi diagram into a polygonal subdivision of complexity $O(\sqrt{m})$ using its adjacency graph and the exact positions of degree-1 and degree-3 vertices.

▶ **Theorem 22.** *We can construct a data structure of size $O(n+m)$ that supports a nearest-point (or a farthest) query for point insertions and deletions. Each query time is $O(\sqrt{m}\log(n+m))$ and each update time is $O(\sqrt{m}\log m\log^2 n))$.*

## References


1. 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. *Discrete & Computational Geometry*, 56(4):836–859, 2016.
2. Boris Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1):109–140, 1989.
3. Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.
4. Jon Louis Bentley and James B. Saxe. Decomposable searching problems 1: Static-to-dynamic transformations. *Journal of Algorithms*, 1(4):297–396, 1980.
5. Bernard Chazelle, Herbert Edelsbrunner, Michelangelo Grigni, Leonidas Guibas, John Hershberger, Micha Sharir, and Jack Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.
6. 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.
7. Steven Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2(1):153–174, 1987.
8. Leonidas Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1):209–233, 1987.
9. Leonidas J. Guibas and John Hershberger. Optimal shortest path queries in a simple polygon. *Journal of Computer and System Sciences*, 39(2):126–152, 1989.
10. John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, 1991.
11. Chih-Hung Liu and D. T. Lee. Higher-order geodesic Voronoi diagrams in a polygonal domain with holes. In *Proceedings of the 24th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2013)*, pages 1633–1645, 2013.
12. Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM Journal on Computing*, 12(4):759–776, 1983.
13. Joseph S. B. Mitchell. Geometric shortest paths and network optimization. In *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.
14. Eunjin Oh, Luis Barba, and Hee-Kap Ahn. The farthest-point geodesic voronoi diagram of points on the boundary of a simple polygon. In *Proceedings of the 32nd International Symposium on Computational Geometry (SoCG 2016)*, pages 56:1–56:15, 2016.
15. Evanthia Papadopoulou and D. T. Lee. A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains. *Algorithmica*, 1998(4):319–352, 1998.
16. Richard Pollack, Micha Sharir, and Günter Rote. Computing the geodesic center of a simple polygon. *Discrete & Computational Geometry*, 4(6):611–626, 1989.
17. Maksym Zavershynskyi and Evanthia Papadopoulou. A sweepline algorithm for higher order voronoi diagrams. In *Proceedings of the 10th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2013)*, pages 16–22, 2013.