An Optimal Deterministic Algorithm for Geodesic Farthest-Point Voronoi Diagrams in Simple Polygons

Haitao Wang
Department of Computer Science, Utah State University, Logan, UT, USA

Abstract

Given a set S of m point sites in a simple polygon P of n vertices, we consider the problem of computing the geodesic farthest-point Voronoi diagram for S in P. It is known that the problem has an $\Omega(n + m \log m)$ time lower bound. Previously, a randomized algorithm was proposed [Barba, SoCG 2019] that can solve the problem in $O(n + m \log m)$ expected time. The previous best deterministic algorithms solve the problem in $O(n \log \log n + m \log m)$ time [Oh, Barba, and Ahn, SoCG 2016] or in $O(n + m \log m + m \log^2 n)$ time [Oh and Ahn, SoCG 2017]. In this paper, we present a deterministic algorithm of $O(n + m \log m)$ time, which is optimal. This answers an open question posed by Mitchell in the Handbook of Computational Geometry two decades ago.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Algorithm design techniques

Keywords and phrases farthest-sites, Voronoi diagrams, triple-point geodesic center, simple polygons

Digital Object Identifier 10.4230/LIPIcs.SoCG.2021.59

Related Version *Full Version*: https://arxiv.org/abs/2103.00076

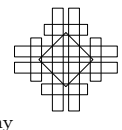
Funding This research was supported in part by NSF under Grant CCF-2005323.

1 Introduction

Let P be a simple polygon of n vertices in the plane. Let S be a set of m points, called *sites*, in P (each site can be either in the interior or on the boundary of P). For any two points in P, their *geodesic distance* is the length of their Euclidean shortest path in P. We consider the problem of computing the geodesic farthest-point Voronoi diagram of S in P, which is to partition P into Voronoi cells such that all points in the same cell have the same farthest site in S with respect to the geodesic distance.

This problem generalizes the Euclidean farthest Voronoi diagram of m sites in the plane, which can be computed in $O(m \log m)$ time [23]; this is optimal as $\Omega(m \log m)$ is a lower bound. For the more general geodesic problem in P, Aronov et al. [3] showed that the complexity of the diagram is $\Theta(n + m)$ and provided an $O(n \log n + m \log m)$ time algorithm. The runtime is close to optimal as $\Omega(n + m \log m)$ is a lower bound. No progress had been made for over two decades until in SoCG 2016 Oh et al. [20] proposed an $O(n \log \log n + m \log m)$ time algorithm. Later in SoCG 2017 Oh and Ahn [19] gave another $O(n + m \log m + m \log^2 n)$ time algorithm and in SoCG 2019 Barba [8] presented a randomized algorithm that can solve the problem in $O(n + m \log m)$ expected time.

In this paper, we give an $O(n + m \log m)$ time deterministic algorithm, which is optimal. The space complexity of the algorithm is $O(n + m)$. This answers an open question posed by Mitchell [17] in the Handbook of Computational Geometry two decades ago.




1.1 Related work

If all sites of S are on the boundary of P, then better results exist. The algorithm of Oh et al. [20] can solve the problem in $O((n+m)\log\log n)$ time while the randomized algorithm of Barba [8] runs in $O(n+m)$ expected time.

The geodesic nearest-point Voronoi diagram for sites in a simple polygon has also attracted much attention. The problem also has an $\Omega(n+m\log m)$ time lower bound. The first close-to-optimal algorithm was given by Aronov [2] and the running time is $O((n+m)\log(n+m)\log n)$. Papadopoulou and Lee [21] improved the algorithm to $O((n+m)\log(n+m))$ time. Recent progress has been made by Oh and Ahn [19] who presented an $O(n+m\log m\log^2 n)$ time algorithm and also by Liu [16] who designed an $O(n+m(\log m+\log^2 n))$ time algorithm. Finally the problem was solved optimally in $O(n+m\log m)$ time by Oh [18].

Another closely related problem is to compute the *geodesic center* of a simple polygon P, which is a point in P that minimizes the maximum geodesic distance from all points of P. Asano and Toussaint [4] first gave an $O(n^4\log n)$ time algorithm for the problem. Pollack et al. [22] derived an $O(n\log n)$ time algorithm. Recently the problem was solved optimally in $O(n)$ time by Ahn et al. [1]. The *geodesic diameter* of P is the largest geodesic distance between any two points in P. Chazelle [9] first gave an $O(n^2)$ time algorithm and then Suri [24] presented an improved $O(n\log n)$ time solution. Hershberger and Suri [13] finally solved the problem in $O(n)$ time.

All above results are for simple polygons. For polygons with holes, the problems become more difficult. The geodesic nearest-point Voronoi diagram for m point sites in a polygon with holes of n vertices can be solved in $O((n+m)\log(n+m))$ time by the algorithm of Hershberber and Suri [14]. Bae and Chwa [5] gave an algorithm for constructing the geodesic farthest-point Voronoi diagram and the algorithm runs in $O(nm\log^2(n+m)\log m)$ time. For computing the geodesic diameter in a polygon with holes, Bae et al. [6] solved the problem in $O(n^{7.73})$ or $O(n^7(h+\log n))$ time, where h is the number of holes. For computing the geodesic center, Bae et al. [7] first gave an $O(n^{12+\epsilon})$ time algorithm, for any constant $\epsilon>0$; Wang [26] solved the problem in $O(n^{11}\log n)$ time.

1.2 Our approach

We follow the algorithm scheme in [19], which follows [3]. Specifically, we first compute the geodesic convex hull of all sites of S in $O(n+m\log m)$ time [11, 12, 25], and then compute the geodesic center c^* of the hull in $O(n+m)$ time [1]. Aronov [3] showed that the farthest Voronoi diagram forms a tree with c^* as the root and all leaves on ∂P, the boundary of P. We construct the farthest Voronoi diagram restricted to ∂P; this can be done in $O(n+m)$ time by a recent algorithm of Oh et al. [20] once the geodesic convex hull of S is known.

Next we run a *reverse geodesic sweeping* algorithm to extend the diagram from ∂P to the interior of P (i.e., based on all leaves on ∂P and the root c^* of the tree, we want to construct the tree). Here we use a *geodesic sweeping circle* that consists of all points with the same geodesic distance from c^*. Aronov [3] implemented this sweeping algorithm in $O((n+m)\log(n+m))$ time. Oh and Ahn [19] gave an improved solution of $O(n+m\log m+m\log^2 n)$ time by using a data structure for the following query problem: Given three points in P, compute the point that is equidistant from them. Oh and Ahn [19] built a data structure in $O(n)$ time that can answer each query in $O(\log^2 n)$ time, and that is why the time complexity of their algorithm has a $\log^2 n$ factor. We improve the query time to $O(\log n)$ (with $O(n)$ time preprocessing) with the help of the following observations. First, the three points involved in a query are three sites of S whose Voronoi cells are adjacent along the sweeping circle. Second, among

the three sites involved in a query, for every two sites whose Voronoi cells are adjacent, the sweeping algorithm provides us with a point equidistant to them. These observations along with the tentative prune-and-search technique of Kirkpatrick and Snoeyink [15] lead us to a query algorithm of $O(\log n)$ time. Consequently, the sweeping algorithm can be implemented in $O(n + m \log m)$ time.

We should point out that in her algorithm for computing the geodesic nearest-point Voronoi diagram, Oh [18] also announced an $O(\log n)$ time algorithm for the above query problem and her algorithm also uses the tentative prune-and-search technique (although the details are omitted due to the page limit). However, the difference is that she uses a balanced geodesic triangulation [10] and her result is based on the assumption that the sought point of the query lies in a known geodesic triangle $\triangle$ and the three query points are in the same subpolygon of P separated by a side of $\triangle$ (see Lemma 4.2 in [18]). For our problem, we do not need the balanced geodesic triangulation and do not have such an assumption. Instead, our algorithm relies on the observations mentioned above.

The rest of the paper is organized as follows. Section 2 defines notation and introduces some concepts. The algorithm for constructing the geodesic Voronoi diagram is described in Section 3. Section 4 presents the algorithm for a lemma about the query problem discussed above. Due to the space limit, some proofs are omitted but can be found in the full paper.

2 Preliminaries

Like the previous work [3, 8, 19, 20], for ease of discussion, we make a general position assumption that no vertex of P is equidistant from two sites of S and no point of P has four farthest sites. We occasionally use *polygon vertex* to refer to a vertex of P and use *polygon edge* to refer to an edge of P.

For any two points p and q in P, let $\pi(p, q)$ denote the (Euclidean) shortest path from p to q in P; let $d(p, q)$ denote the length of $\pi(p, q)$. $\pi(p, q)$ is also called the *geodesic path* and $d(p, q)$ is called the *geodesic distance* between p and q. The vertex of $\pi(p, q)$ adjacent to q (resp., p) is called the *anchor* of q (resp., p) in $\pi(p, q)$.

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

For two sites s and t of S, their *bisector*, denoted by $B(s, t)$, consists of all points of P equidistant from them, i.e., $B(s, t) = \{p \mid d(s, p) = d(t, p), p \in P\}$. Due to the general position assumption, Aronov et al. [2] showed that $B(s, t)$ is a smooth curve connecting two points on ∂P with no other points common with ∂P and $B(s, t)$ comprises $O(n)$ straight and hyperbolic arcs (a straight arc is a line segment); the endpoints of the arcs are *breakpoints*, each of which is the intersection of $B(s, t)$ and a segment extended from a polygon vertex u to another polygon vertex v such that u is an anchor of v in $\pi(s, v)$ or in $\pi(t, v)$ (it is possible that $u = s$ or $u = t$); e.g., see Fig. 1.

For any site $s \in S$, define $C(s)$ as the region consisting of all points p of P whose farthest site is s, i.e., $C(s) = \{p \mid d(p, s) \geq d(p, s'), s' \in S\}$. We call $C(s)$ the (farthest) *Voronoi cell* of s. Note that $C(s)$ may be empty; if $C(s)$ is not empty, then it is simply connected [3]. The Voronoi cells of all sites of S form a partition of P. We define the *geodesic farthest-point Voronoi diagram* (or *farthest Voronoi diagram* for short), denoted by $FVD(S)$, as the closure of the interior of P minus the union of the interior of $C(s)$ for all $s \in S$; alternatively, $FVD(S) = \{p \in B(s, t) \mid s, t \in S \text{ and } d(s, p) = \max_{r \in S} d(r, p)\}$. A point v of $FVD(S)$ is a *Voronoi vertex* if it is an intersection of a bisector with ∂P or if it has degree 3 (i.e., it has three equidistant sites). The curve of $FVD(S)$ connecting two adjacent vertices is called a

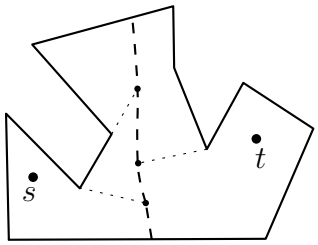


Figure 1 Illustrating the bisector $B(s,t)$ (the dashed curve) with three breakpoints.

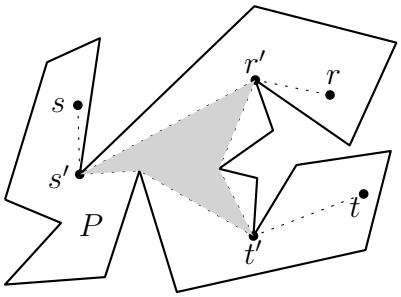


Figure 2 Illustrating a geodesic triangle $\triangle(s,t,r)$ (the gray region).

Voronoi edge, which is a portion of a bisector of two sites. Note that a Voronoi edge may not be of constant size because it may contain multiple breakpoints. While $FVD(S)$ has $O(m)$ Voronoi vertices and edges, the total complexity of $FVD(S)$ is $O(n+m)$ [3].

A subset P' of P is *geodesically convex* if $\pi(p,q)$ is in P' for any two points p and q in P'. The *geodesic convex hull* of S in P, denoted by $GCH(S)$, is the common intersection of all geodesically convex sets containing S. $GCH(S)$ is a weakly simple polygon of at most $n+m$ vertices. Let c^* be the geodesic center of $GCH(S)$, which is also the geodesic center of S [3]. Note that c^* must be on a Voronoi edge of $FVD(S)$. Indeed, if c^* has three farthest sites in S, then c^* is a Voronoi vertex; otherwise it has two farthest sites and thus is in the interior of an edge of $FVD(S)$. Aronov [3] proved that $FVD(S)$ is a tree with c^* as the root and all leaves on ∂P; he also showed that only sites on the boundary of $GCH(s)$ have nonempty cells in $FVD(S)$ and the ordering of the sites with nonempty cells around the boundary of $GCH(s)$ is the same as the ordering of their Voronoi cells around ∂P (the *ordering lemma*). Note that a site on the boundary of $GCH(s)$ may still have an empty Voronoi cell and intuitively this is because P is not large enough [3].

Consider any three points s,t,r in P. The vertex farthest to s in $\pi(s,t) \cap \pi(s,r)$ is called the *junction vertex* of $\pi(s,t)$ and $\pi(s,r)$. The closure of the interior of the geodesic convex hull $GCH(s,t,r)$ is called the *geodesic triangle* of s, t, and r, denoted by $\triangle(s,t,r)$, whose boundary is composed of three convex chains $\pi(s',t')$, $\pi(t',r')$, $\pi(r',s')$, where s' is the junction vertex of $\pi(s,t)$ and $\pi(s,r)$, and t' and r' are defined likewise; e.g., see Fig. 2. The three convex chains are called *sides* of $\triangle(s,t,r)$. The three vertices s', t', and r' are called the *apexes* of $\triangle(s,t,r)$.

3 Computing the farthest Voronoi diagram $FVD(S)$

In this section, we present our algorithm for computing the farthest Voronoi diagram $FVD(S)$.

First, we compute the geodesic convex hull $GCH(S)$ of S in $O(n+m\log m)$ time [11,12,25]. Second, we compute the geodesic center c^* of $GCH(S)$ in $O(n+m)$ time [1]. Third, we compute the portion of $FVD(S)$ restricted to the polygon boundary ∂P, i.e., the leaves of

$FVD(S)$. This can be done in $O(n+m)$ time by the algorithm in [20].[1] The fourth step is to extend the diagram to the interior of P, i.e., construct the tree $FVD(S)$ based on all its leaves and the root c^*. This is achieved by a reverse geodesic sweeping algorithm, whose details are described below.

The algorithm first computes the adjacency information of $FVD(S)$. Specifically, we will compute the locations of all Voronoi vertices of $FVD(S)$; for Voronoi edges, however, we will not compute them exactly (i.e., the locations of their breakpoints will not be computed) but only output their incident Voronoi vertices, i.e., if u and v are two Voronoi vertices incident to the same Voronoi edge, then we will output the pair (u, v) as an *abstract* Voronoi edge. In this way, we will output the abstract tree $FVD(S)$ with the exact locations of all Voronoi vertices; this is called the *topological structure* of $FVD(S)$ in [19]. After having the topological structure, Oh and Ahn [19] gave an algorithm that can construct $FVD(S)$ in additional $O(n + m \log m)$ time. More specifically, with $O(n)$ time preprocessing, each Voronoi edge can be computed in $O(\log n + k)$ time, where k is the number of breakpoints in the Voronoi edge (see Section 4 of [19] for details). As $FVD(S)$ has $O(m)$ Voronoi edges, the total time for computing all Voronoi edges is $O(m \log n + K)$, where K is the total number of breakpoints on all Voronoi edges. As $K = O(n + m)$ [3], the total time is bounded by $O(n + m \log n)$, which is $O(n + m \log m)$.[2] In the following, we will focus on computing the topological structure of $FVD(S)$.

We use a reverse geodesic sweeping as in [3, 19]. Roughly speaking, the sweep line is a *geodesic circle* C consisting of all points in P that have the same geodesic distance from the geodesic center c^* of S. This statement is actually not quite accurate as initially the sweep circle is just the boundary of P. During the sweeping, we maintain the sites whose Voronoi cells currently intersect C; these sites are stored in a cyclic linked list $\mathcal{L}$ ordered by their intersections with C. Initially when $C = \partial P$, as we already have the leaves of $FVD(S)$, we can build $\mathcal{L}$ in $O(n + m)$ time. Note that $|\mathcal{L}| = O(m)$. During the algorithm, C will shrink until c^*; an event happens when C hits a Voronoi vertex, which will be computed on the fly. Specifically, for each triple of adjacent sites s, t, r in the list $\mathcal{L}$, we compute the point, denoted by $\alpha(s, t, r)$, equidistant from them, which is the intersection of the bisectors $B(s, t)$ and $B(t, r)$. Due to our general position assumption, $\alpha(s, t, r)$ is unique if it exists (see Lemma 2.5.3 [3]). We store all these α-points in a priority queue Q, ordered by decreasing geodesic distance from c^*. In order to compute the α-points, for any pair of adjacent sites s and t in $\mathcal{L}$, we maintain a Voronoi vertex, denoted by $\beta(s, t)$, on their bisector $B(s, t)$ with the following property: $\beta(s, t)$ is outside or on the current geodesic circle C. Initially, we set $\beta(s, t)$ to be the Voronoi vertex on ∂P incident to the Voronoi cells of s and t; so the above property holds as $C = \partial P$.

The main loop of the algorithm works as follows. As long as Q is not empty, we repeatedly extract the point with largest geodesic distance from c^* and let the point be $\alpha(s, t, r)$ defined by three sites s, t, r in this order in $\mathcal{L}$. We report $\alpha(s, t, r)$ as a Voronoi vertex and report

[1] Note that the result was not explicitly given in [20] but can be obtained from their $O(n \log\log n + m \log m)$-time algorithm for computing $FVD(S)$. Indeed, given $GCH(S)$, the algorithm first partitions P in $O(n + m)$ time into $O(1)$ subpolygons such that each subpolygon P' is for a problem instance where all involved sites are on the boundary of P' (see Section 7 [20]). Then, each problem instance is further reduced in linear time to a problem instance where all sites are vertices of P' (see Section 6 [20]), and each such problem instance can be solved in linear time (see Section 3 [20]). The total running time of all above is $O(n + m)$ (for computing $FVD(S)$ restricted to the boundary of P only). This result was also used by Oh and Ahn [19] in their $O(n + m \log m + m \log^2 n)$-time algorithm for computing $FVD(S)$.

[2] Indeed, if $m < n/\log n$, then $n + m \log n = \Theta(n)$, which is $O(n + m \log m)$; otherwise, $\log n = O(\log m)$ and $n + m \log n = O(n + m \log m)$.

$(\beta(s,t),\alpha(s,t,r))$ and $(\beta(t,r),\alpha(s,t,r))$ as two abstract Voronoi edges. We remove t from $\mathcal{L}$ and set $\beta(s,r)=\alpha(s,t,r)$. Let x be the neighbor of s other than r in $\mathcal{L}$ and let y be the neighbor of r other than s. We remove $\alpha(x,s,t)$ and $\alpha(t,r,y)$ from Q if they exist. Next, we compute $\alpha(x,s,r)$ and $\alpha(s,r,y)$ (if exist) as well as their geodesic distances from c^*, and insert them into Q.

For the running time, there are $O(m)$ events, for the total number of Voronoi vertices of $FVD(S)$ is $O(m)$ [3], and thus the total time of the algorithm is $O(m\cdot\sigma)$, where $O(\sigma)$ is the time for computing each α point. Lemma 1, which will be proved later in Section 4, is for computing the α-points.

▶ **Lemma 1.** *With $O(n)$ time preprocessing, for any triple of adjacent sites s,t,r in $\mathcal{L}$ at any moment during the algorithm, given the two Voronoi vertices $\beta(s,t)$ and $\beta(t,r)$, our algorithm can do the following in $O(\log n)$ time: if $\alpha(s,t,r)$ is a Voronoi vertex, then compute it; otherwise, either compute $\alpha(s,t,r)$ or return null.*

We remark that Lemma 1 is sufficient for the correctness of our geodesic sweeping algorithm as only Voronoi vertices are essential. If the algorithm returns null, the event will not be inserted to Q. With Lemma 1 at hand, our geodesic sweeping algorithm computes the topological structure of $FVD(S)$ in $O(n+m\log m)$ time. After that, as discussed above, we can compute the full diagram $FVD(S)$ in additional $O(n+m\log m)$ time by the techniques of Oh and Ahn [19]. Also, the space of the algorithm is bounded by $O(n+m)$.

▶ **Theorem 2.** *The geodesic farthest-point Voronoi diagram of a set of m points in a simple polygon of n vertices can be computed in $O(n+m\log m)$ time and $O(n+m)$ space.*

4 Algorithm for Lemma 1

In this section, we present our algorithm for Lemma 1. We first present an algorithm in Section 4.1 for the following *triple-point geodesic center query* problem: given any three points in P, compute their geodesic center in P, which is a point that minimizes the largest geodesic distance from the three query points. Oh and Ahn [19] solved this problem in $O(\log^2 n)$ time, after $O(n)$ time preprocessing. Our algorithm runs in $O(\log n)$ time also with $O(n)$ time preprocessing.[3] This algorithm will be used as a subroutine in our algorithm for Lemma 1, which will be discussed in Section 4.2.

4.1 The triple-point geodesic center query problem

As preprocessing, we construct the two-point shortest path query data structure by Guibas and Hershberger [11, 12] and we refer to it as the *GH data structure*. The data structure can be constructed in $O(n)$ time, after which given any two points p and q in P, the geodesic distance $d(p,q)$ can be computed in $O(\log n)$ time and the geodesic path $\pi(p,q)$ can be output in additional time linear in the number of edges of $\pi(p,q)$.

Consider three query points s, t, and r in P. Our goal is to compute their geodesic center, denoted by c. We follow the algorithm scheme in [19]. Consider the geodesic convex hull $GCH(s,t,r)$ and the geodesic triangle $\triangle(s,t,r)$. We know that c is the geodesic center of $GCH(s,t,r)$ [3]. Depending on whether c is in the interior of $\triangle(s,t,r)$, there are two cases.

[3] To be fair, this problem is not a dominant one in their algorithm, which might be a reason Oh and Ahn [19] did not push their result further.

4.1.1 c is not in the interior of $\triangle(s,t,r)$

If c is not in the interior of $\triangle(s,t,r)$, then it must be on the geodesic path of two points of $\{s,t,r\}$. Without loss of generality, we assume that $c \in \pi(s,t)$. Note that c must be the middle point of $\pi(s,t)$. To locate c in $\pi(s,t)$, we wish to do binary search on the vertices of $\pi(s,t)$. It was claimed in [19] that the query algorithm of the GH data structure returns $\pi(s,t)$ as a binary tree (so that binary search can be done in a straightforward way), in particular, when the simpler approach in [12] is utilized. In fact, this is not quite correct. Indeed, the binary tree structures in both [11] and [12] are used for representing convex chains (or more rigorously, *semiconvex chains* [12]). However, $\pi(s,t)$ is actually a *string* [11], which in general is not a semiconvex chain. The data structure for representing a string is a tree but not a binary tree because a node in the tree may have three children.

For completeness, we provide a binary search scheme on the geodesic path $\pi(p,q)$ returned by the GH data structure for any two query points p and q in P. Suppose we are looking for either a vertex or an edge of $\pi(p,q)$, denoted by w^* in either case, and we have access to an *oracle* such that given any vertex $v \in \pi(p,q)$, the oracle can determine whether w^* is in $\pi(p,v)$ or in $\pi(v,q)$. Then, we have the following lemma (whose proof is omitted).

▶ **Lemma 3.** *With $O(n)$ time preprocessing, given any two query points p and q, the sought vertex or edge w^* can be located by a binary search algorithm that calls the oracle on $O(\log n)$ vertices of $\pi(p,q)$, and the total time of the binary search excluding the time for calling the oracle is $O(\log n)$. In particular, the middle point of $\pi(p,q)$ can be found in $O(\log n)$ time.*

With Lemma 3 at hand, we can find c on $\pi(s,t)$ in $O(\log n)$ time.

We can determine whether c is in the interior of $\triangle(s,t,r)$ in $O(\log n)$ time using Lemma 3 as follows. First, we determine whether c is the middle point of $\pi(s,t)$. To do so, we first compute the middle point p_{st} of $\pi(s,t)$ by Lemma 3. Then, we compute $d(s,p_{st})$ and $d(r,p_{st})$ in $O(\log n)$ time using the GH data structure. It is not difficult to see that p_{st} is c if and only if $d(s,p_{st}) \geq d(r,p_{st})$. If $p_{st} \neq c$, then we use the same way to determine whether the middle point of $\pi(s,r)$ (resp., $\pi(r,t)$) is c. If the above algorithm fails to locate c, then we know that c is in the interior of $\triangle(s,t,r)$.

The above finds c in $O(\log n)$ time for the case where c is not in the interior of $\triangle(s,t,r)$.

4.1.2 c is in the interior of $\triangle(s,t,r)$

We proceed to the case where c is in the interior of $\triangle(s,t,r)$. Our algorithm utilizes the tentative prune-and-search technique of Kirkpatrick and Snoeyink [15].

First observe that in this case c must be equidistant from all three points s, t, and r. Let s' be the junction vertex of $\pi(s,t)$ and $\pi(s,r)$. Define t' and r' similarly. With the GH data structure, each junction vertex can be computed in $O(\log n)$ time [11].

Define p_s, p_t, and p_r to be the anchors of c in $\pi(s,c)$, $\pi(t,c)$, and $\pi(r,c)$, respectively. Note that the segment connecting c to p_s (resp., p_t, p_r) is tangent to the side of $\triangle(s,t,r)$ that contains it. As c is equidistant to s, t, and r, c is the common intersection of the three bisectors $B(s,t)$, $B(s,r)$, and $B(t,r)$. The proof of the lemma below is omitted.

▶ **Lemma 4.** *The middle point p_{st} of $\pi(s,t)$ must be in $\pi(s',t')$; the middle point p_{sr} of $\pi(s,r)$ must be in $\pi(s',r')$; the middle point p_{tr} of $\pi(t,r)$ must be in $\pi(t',r')$.*

Since p_s is the anchor of c in $\pi(s,c)$, p_s has smaller geodesic distance from s than from t or r. Hence, by Lemma 4, p_s must be in $\gamma_s = \pi(s',p_{st}) \cup \pi(s',p_{sr})$, which consists of two convex chains; we call γ_s a *pseudo-convex chain*. Similarly, p_t must be in $\gamma_t = \pi(t',p_{st}) \cup \pi(t',p_{tr})$

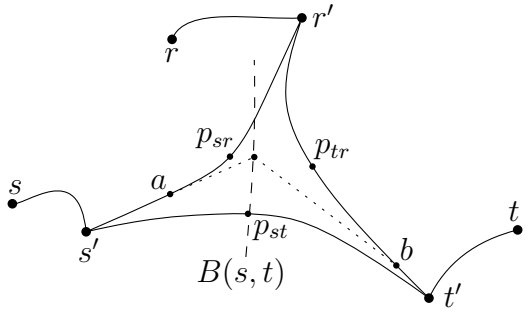


Figure 3 Illustrating the geodesic triangle $\triangle(s,t,r)$ and the function $f(a)$ for $a \in A = \gamma_s$.

and p_r must be in $\gamma_r = \pi(r', p_{tr}) \cup \pi(r', p_{sr})$. We consider p_{st} and p_{sr} as two ends of γ_s. If we move a point p on γ_s from one end to the other, then the slope of the tangent line of γ_s at p continuously changes. Further, p_s is the only point on γ_s such that $\overline{cp_s}$ is tangent to γ_s. Similar properties hold for γ_t, p_t, γ_r, and p_r.

With the above discussion, we now describe our algorithm for computing c. First, we compute the three junction vertices and the three middle points s', t', r', p_{st}, p_{sr}, and p_{tr}. This can be done in $O(\log n)$ time using the GH data structure. To compute c (as well as locate p_s, p_t, and p_r), we resort to the tentative prune-and-search technique [15], as follows.

To avoid the lengthy background explanation, we follow the notation in [15] without definition. We will rely on Theorem 3.9 in [15]. To this end, we need to define three continuous and monotone-decreasing functions f, g, and h. We define them in a way similar to Theorem 4.10 in [15] for finding a point equidistant to three convex polygons. Indeed, our problem may be considered as a weighted case of their problem because each point in our pseudo-convex chains has a weight equal to its geodesic distance from one of s, t, and r.

We parameterize over $[0,1]$ each of the three pseudo-convex chains $A = \gamma_s$, $B = \gamma_t$, and $C = \gamma_r$ from one end to the other in counterclockwise order around $\triangle(s,t,r)$. For example, without loss of generality, we assume that s', t', and r' are counterclockwise around $\triangle(s,t,r)$. Then, γ_s is parameterized from p_{sr} to p_{st} over $[0,1]$, i.e., each value of $[0,1]$ corresponds to a slope of a tangent at a point on γ_s. For each point a of A, we define $f(a)$ to be the parameter of the point $b \in B$ such that the tangent of A at a and the tangent of B at b intersect at a point on the bisector $B(s,t)$ of s and t (e.g., see Fig. 3). Similarly, we define $g(b)$ for $b \in B$ with respect to C and define $h(c)$ for $c \in C$ with respect to A. One can verify that all three functions are continuous and monotone-decreasing (the tangent at an apex of $\triangle(s,t,r)$ is not unique but the issue can be handled [15]). The fixed-point of the composition of the three functions $h \cdot g \cdot f$ corresponds to c, which can be computed by applying the tentative prune-and-search algorithm of Theorem 3.9 [15].

To see that the algorithm can be implemented in $O(\log n)$ time, we need to show that given any $a \in A$ and any $b \in B$, we can determine whether $f(a) > b$ in $O(1)$ time. To this end, we first find the intersection p of the tangent of A at a and the tangent of B at b. Then, $d(s,a) + |\overline{pa}| < d(t,b) + |\overline{pb}|$ if and only if $f(a) > b$. We will discuss below that the values $d(s,a)$ and $d(t,b)$ will be available during the tentative prune-and-search algorithm. Note that here the tangent of A at a actually refers to the half-line of the tangent whose concatenation with $\pi(s',a)$ is still a convex chain (so that the shortest path can follow that half-line), as shown in Fig. 3. Hence, it is possible that the tangent half-line of a does not intersect the tangent half-line of b. If that happens, either the tangent half-line of a intersects the backward extension of the tangent half-line of b or the backward extension of the tangent

half-line of a intersects the tangent half-line of b; in the former case we have $f(a) < b$ and in the latter case $f(a) > b$. Similar properties hold for functions of g and h. Finally, we show that we have appropriate data structures to represent the three pseudo-convex chains A, B, and C so that the algorithm can terminate in $O(\log n)$ rounds. We only discuss A since other two are similar. When the algorithm picks the first vertex of A to test, we will use the vertex s'. After the test, the algorithm will proceed on A on one side of s', say, on $\pi(s', p_{st})$. We apply the binary search scheme of Lemma 3 on $\pi(s', p_{st})$, which will test $O(\log n)$ vertices. Further, whenever a vertex $a \in \pi(s', p_{st})$ is tested, the binary search scheme of Lemma 3 can keep track of $d(s, a)$. Therefore, applying the tentative prune-and-search technique in Theorem 3.9 [15] can compute the geodesic center c in $O(\log n)$ time.

The lemma below summarizes our result on the triple-point geodesic center query problem.

▶ **Lemma 5.** *With $O(n)$ time preprocessing, the geodesic center of any three query points in P can be computed in $O(\log n)$ time.*

4.2 Proving Lemma 1

With Lemma 5, we are ready to present our algorithm for Lemma 1. Consider any three sites s, t, r as specified in the statement of Lemma 1. Our goal is to compute the point $\alpha(s, t, r)$, which is equidistant from the three sites. Recall that we have two points $\beta(s, t)$ and $\beta(t, r)$ available for us, which are critical to the success of our approach.

The first step of our algorithm is to apply the algorithm for Lemma 5 to compute the geodesic center c of the three sites. We check whether c is equidistant to the three sites, in $O(\log n)$ time. If yes, $\alpha(s, t, r) = c$ and we are done. In the following we assume otherwise.

Our algorithm may not compute $\alpha(s, t, r)$ even if it exists, but will guarantee to do so if $\alpha(s, t, r)$ is a Voronoi vertex of $FVD(S)$. This is sufficient for constructing $FVD(S)$ correctly. Hence, in what follows we assume that $\alpha(s, t, r)$ is a Voronoi vertex. This implies that there are two Voronoi edges connecting $\alpha(s, t, r)$ with $\beta(s, t)$ and $\beta(t, r)$, respectively. Recall that c^* is the geodesic center of S. The following lemma was discovered by Aronov et al. [3].

▶ **Lemma 6** (Aronov et al. [3])**.** *If a point p moves from $\beta(s, t)$ (resp., $\beta(t, r)$) to $\alpha(s, t, r)$ along the Voronoi edge, both $d(c^*, p)$ and $d(t, p)$ are monotonically decreasing.*

To simplify the notation, unless otherwise stated, we use α to refer to $\alpha(s, t, r)$. We define the three junction vertices s', t', and r' in the same way as before, which are the three apexes of the geodesic convex hull $\triangle(s, t, r)$. Without loss of generality, we assume that s', t', and r' are counterclockwise around the boundary of $\triangle(s, t, r)$ (e.g., see Fig. 2). We define p_s, p_t, and p_r as the anchors of α in $\pi(s, \alpha)$, $\pi(t, \alpha)$, and $\pi(r, \alpha)$, respectively. Define p_{st}, p_{sr}, and p_{tr} as the middle points of $\pi(s, t)$, $\pi(s, r)$, and $\pi(t, r)$, respectively.

Lemma 7, obtained from the results of Aronov [2], will occasionally be used later.

▶ **Lemma 7** (Aronov [2])**.** *Suppose x and y are two points of P such that their bisector $B(x, y)$ does not contain any vertex of P. Then, for any point $z \in P$, the shortest path $\pi(x, z)$ (resp., $\pi(y, z)$) either does not intersect $B(x, y)$ or intersects it at a single point.*

Recall that $\alpha(s, t, r)$ is not c. Hence, c must be equidistant to two sites and the geodesic distance from them to c is strictly larger than that from the third site to c. Depending on what the two sites are, there are three cases $d(c, s) = d(c, r) > d(c, t)$, $d(c, t) = d(c, r) > d(c, s)$, and $d(c, t) = d(c, s) > d(c, r)$. The following lemma (whose proof is omitted) shows that the latter two cases cannot happen.

▶ **Lemma 8.** *Neither $d(c, t) = d(c, r) > d(c, s)$ nor $d(c, t) = d(c, s) > d(c, r)$ can happen.*

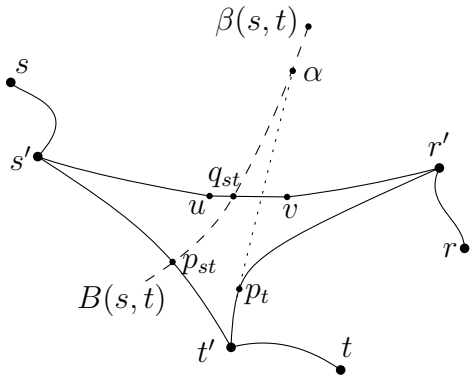


Figure 4 Illustrating the subcase $c \in \pi(s', r')$: c is on $\overline{uv}$.

Note that Lemma 8 is obtained based on the assumption that $\alpha(s,t,r)$ is a Voronoi vertex of $FVD(S)$. Therefore, if one of the two cases in Lemma 8 happens during the algorithm, then we can simply return null.

In what follows, we assume that $d(c,s) = d(c,r) > d(c,t)$. Thus, c must be the middle point of $\pi(s,r)$. Depending on where the location of c is, there are three cases: $c \in \pi(s', r')$, $c \in \pi(s, s') \setminus \{s'\}$, and $c \in \pi(r', r) \setminus \{r'\}$. The latter two cases are symmetric, so we will only discuss the first two cases.

4.2.1 The case $c \in \pi(s', r')$

Let $\overline{uv}$ be the edge of $\pi(s', r')$ containing c such that $d(s,u) < d(s,v)$. It possible that u is s or/and v is t. We first assume that both u and v are polygon vertices; we will show later the other case can be reduced to this case.

Since both u and v are polygon vertices, $\overline{uv}$ divides P into two sub-polygons; one of them, denoted by P_1, does not contain t and we use P_2 to denote the other one. Note that all three sites s, t, r are in P_2. According to Oh and Ahn [19] (see the proof of Lemma 3.6), α is in P_1 and $\overline{p_t\alpha}$ intersects $\overline{uv}$, i.e., p_t is in the pseudo-convex chain $\pi(t', r') \cup \pi(t', s')$; e.g., see Fig. 4.

Consider the bisector $B(s,t)$. Because α is equidistant from s and t, $\alpha \in B(s,t)$. As both s and t are in P_2, the middle point p_{st} of $\pi(s,t)$ is also in P_2. Note that $p_{st} \in B(s,t)$. As $\alpha \in P_1$ and $p_{st} \in P_2$, $B(s,t)$ must cross $\overline{uv}$; further, by Lemma 7, as $\overline{uv} \subseteq \pi(s,r)$, $B(s,t)$ intersects $\overline{uv}$ at a single point, denoted by q_{st} (e.g., see Fig. 4). Hence, q_{st} partitions $B(s,t)$ into two parts, one inside P_1 and the other in P_2. We use $B_1(s,t)$ to denote the part in P_1. Recall that $\beta(s,t)$ is on $B(s,t)$.

▶ **Lemma 9.** *$\beta(s,t)$ is on $B_1(s,t)$, and α is on $B_1(s,t)$ between $\beta(s,t)$ and q_{st} (e.g., see Fig. 4).*

Proof. First of all, since $\alpha \in P_1$, $\alpha \in B(s,t)$, and $B_1(s,t) = B(s,t) \cap P_1$, we obtain that $\alpha \in B_1(s,t)$. Notice that p_{st} is the point on $B(s,t)$ closest to t and if we move p from one end of $B(s,t)$ to the other end, $d(s,p)$ will first decrease until $p = p_{st}$ and then increase. By Lemma 6, if we move a point p along $B(s,t)$ from $\beta(s,t)$ to α, $d(t,p)$ decreases. Since $\alpha \in B_1(s,t)$, $\beta(s,t)$ must be on $B_1(s,t)$ and α is between $\beta(s,t)$ and q_{st}. ◀

Our algorithm relies on the following lemma to compute α (e.g., see Fig. 5).

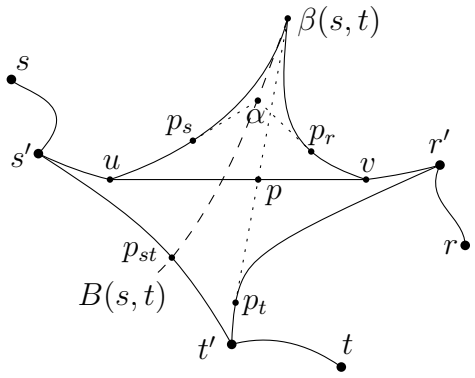


Figure 5 Illustrating the proof of Lemma 10.

▶ **Lemma 10.**

1. *α must be in the geodesic triangle $\triangle(s,r,\beta(s,t))$.*
2. *The apexes of $\triangle(s,r,\beta(s,t))$ are u, v, and $\beta(s,t)$.*
3. *p_s must be on the convex chain $\pi(u,\beta(s,t))$ and $\overline{\alpha p_s}$ is tangent to the chain.*
4. *p_r must be on the convex chain $\pi(v,\beta(s,t))$ and $\overline{\alpha p_r}$ is tangent to the chain.*
5. *p_t must be on the pseudo-convex chain $\pi(t_{uv},u) \cup \pi(t_{uv},v)$ and $\overline{\alpha p_t}$ is tangent to the chain, where t_{uv} is the junction vertex of $\pi(t,u)$ and $\pi(t,v)$.*
6. *$\overline{\alpha p_t}$ intersects $\overline{uv}$.*

Proof. To simplify notation, let $q = \beta(s,t)$. Since $q \in B(s,t)$, due to the general position assumption, the incident edges of q in $\pi(q,s)$ and $\pi(q,t)$ cannot be coincident [3]. Hence, q is the junction vertex of $\pi(q,s)$ and $\pi(q,t)$. As $t \in P_2$ and $q \in P_1$, $\pi(q,t)$ must cross $\overline{uv}$ at a point p; see Fig. 5. This implies that q is the junction vertex of $\pi(q,u)$ and $\pi(q,p)$. As $p \in \overline{uv}$, q must also be the junction vertex of $\pi(q,u)$ and $\pi(q,v)$. Since $u \in \pi(s,q)$ and $v \in \pi(q,r)$, q must be the junction vertex of $\pi(q,s)$ and $\pi(q,r)$. Hence, q is an apex of $\triangle(s,r,q)$.

Next we show that u and v are the other two apexes of $\triangle(s,r,q)$. By the definition of P_1, for any point $p' \in P_1$, $\pi(s,p')$ must contain u and $\pi(r,p')$ must contain v. Recall that $B(s,t)$ intersects $\overline{uv}$ at a single point q_{st}. Due to the general position assumption, q_{st} is in the interior of $\overline{uv}$. Hence, u and v are on different sides of $B(s,t)$. As $q \in B(s,t)$, $\pi(u,q)$ cannot contain v and $\pi(v,q)$ cannot contain u. Since $\pi(s,q)$ contains u, we obtain that $\pi(s,q) = \pi(s,u) \cup \pi(u,q)$ and thus $\pi(s,q)$ does not contain v. Similarly, since $\pi(r,q)$ contains v, we obtain that $\pi(r,q) = \pi(r,v) \cup \pi(v,q)$ and thus $\pi(r,q)$ does not contain u. Therefore, u is the junction vertex of $\pi(s,q)$ and $\pi(s,r)$; v is the junction vertex of $\pi(r,q)$ and $\pi(r,s)$. Thus, both u and v are apexes of $\triangle(s,r,q)$.

The above argument proves that the three apexes of $\triangle(s,r,q)$ are u, v, and q.

We proceed to show that $\alpha \in \triangle(s,r,q)$. Indeed, as α is on $B(s,t)$ between q and q_{st} by Lemma 9, α must be in the geodesic triangle $\triangle(s,t,q)$. As $\alpha \in P_1$, $\alpha \in P_1 \cap \triangle(s,t,q)$. Since $\pi(q,s) \cap P_1 = \pi(q,u)$ and $\pi(q,t) \cap P_1 = \pi(q,p)$, we have $P_1 \cap \triangle(s,t,q) = \triangle(u,p,q)$. Hence, $\alpha \in \triangle(u,p,q)$. Further, since $p \in \overline{uv}$, $\triangle(u,p,q) \subseteq \triangle(u,v,q)$. Thus, we obtain $\alpha \in \triangle(s,r,q)$, for $\triangle(s,r,q) = \triangle(u,v,q)$. This proves the first two lemma statements.

The lemma statement (6) was already proved in [19]. Finally we prove the lemma statements (3-5). Clearly, the anchor p_s must be on the two sides of $\triangle(s,r,q)$ incident to the apex u, i.e., $p_s \in \pi(u,q) \cup \overline{uv}$. Note that $\pi(u,\alpha)$ cannot contain v unless $\alpha = v$. However, $\alpha \neq v$ because α is not in $\triangle(s,t,r)$ (otherwise the first step of our algorithm, which is the algorithm for Lemma 5, would have already computed $\alpha(s,t,r)$). Hence, p_s must be on

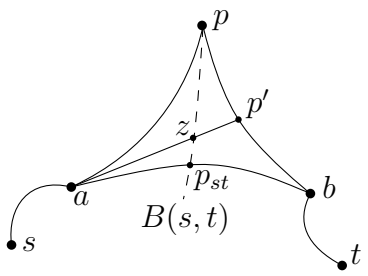


Figure 6 Illustrating the pseudo-triangle $\triangle(s,t,p)$.

$\pi(u,q)$, which is a convex chain. This also means that $\overline{\alpha p_s}$ is tangent to $\pi(u,q)$ at p_s. For the same reason, $\overline{\alpha p_r}$ is tangent to $\pi(v,q)$ at p_r. For p_t, because $\overline{\alpha p_t}$ intersects $\overline{uv}$, it is obviously true that $\overline{\alpha p_t}$ must be tangent to the pseudo-convex chain $\pi(t_{uv},u) \cup \pi(t_{uv},v)$ at p_t. ◀

In light of Lemma 10, $\overline{uv}$, it is actually tangent to $\pi(t',u) \cup \pi(t',v)$. Then, we can apply the tentative prune-and-search technique [15] on the three chains specified in the lemma in a similar way as before to compute α in $O(\log n)$ time.

We summarize our algorithm for this case. First, we compute the edge $\overline{uv}$, in $O(\log n)$ time using the GH data structure by Lemma 3. Second, we compute the junction vertex t_{uv} of $\pi(t,u)$ and $\pi(t,v)$ in $O(\log n)$ time by the GH data structure [11]. Third, we apply the tentative prune-and-search technique on the three chains as specified in Lemma 10, along with the binary search scheme in Lemma 3 on the chains, to compute α in $O(\log n)$ time.

Recall that the above algorithm is based on the assumption that α is a Voronoi vertex of $FVD(S)$. However, when we invoke the procedure during the geodesic sweeping algorithm we do not know whether the assumption is true. Therefore, as a final step, we add a validation procedure as follows. Suppose α is the point returned by the algorithm. First, we check whether $d(s,\alpha) = d(t,\alpha) = d(r,\alpha)$. If not, we return null. Otherwise, we further check whether $d(c^*,\alpha) \leq \min\{d(c^*,\beta(s,t)), d(c^*,\beta(t,r))\}$. This is because the Voronoi vertex α is only useful if it is inside the current sweeping circle C, whose geodesic distance to c^* is at most $\min\{d(c^*,\beta(s,t)), d(c^*,\beta(t,r))\}$ (because neither $\beta(s,t)$ nor $\beta(t,r)$ is in the interior of C). Hence, if $d(c^*,\alpha) \leq \min\{d(c^*,\beta(s,t)), d(c^*,\beta(t,r))\}$, then we return α; otherwise, we return null. This validation step takes $O(\log n)$ time by the GH data structure.

At least one of u and v is not a polygon vertex. The above discusses the case where both u and v are polygon vertices. In the following, we consider the other case where at least one of them is not a polygon vertex, i.e., $u = s$ or/and $v = r$ (because all vertices of $\pi(s,r)$ except s and r are polygon vertices). In fact, this case is missed from the algorithm of Oh and Ahn [19] (see the proof of Lemma 3.6 [19]). It turns out that Lemma 10 still holds for this case and thus we can apply exactly the same algorithm as above. By reducing this case to the previous case where u and v are polygon vertices. we can obtain the following lemma (whose proof is omitted).

▶ **Lemma 11.** *Lemma 10 still holds when $u = s$ or/and $v = r$.*

We finally prove the following technical lemma, which is needed in the proof of Lemma 11. The lemma, which establishes a very basic property of shortest paths in simple polygons, may be interesting in its own right. The proof is omitted.

▶ **Lemma 12.** *Let s and t be any two points in P such that $B(s,t)$ does not contain any vertex of P. Suppose p is a point in $B(s,t)$ and p' is a point in $\pi(t,p)$. Then, $\pi(s,p')$ intersects $B(s,t)$ at a single point z and $d(s,z) \geq d(z,p')$ (in particular, $d(s,z) > d(z,p')$ if $p' \neq t$); e.g., see Fig. 6.*

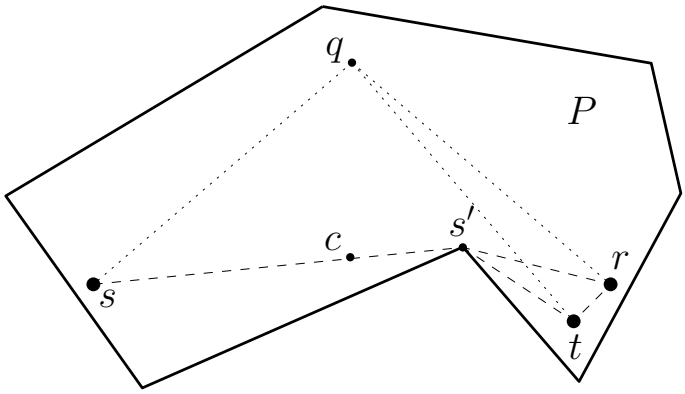


Figure 7 Illustrating an example where α exists when $c \in \pi(s, s') \setminus \{s'\}$. The apexes of the geodesic triangle $\triangle(s, r, t)$ are s', $r' = r$, and $t' = t$. $|\overline{s't}| < |\overline{s'r}|$. c is the middle point of $\pi(s, r) = \overline{ss'} \cup \overline{s'r}$. However, one can verify (e.g., by a ruler) that q is equidistant to s, r, and t, and thus $\alpha = q$ exists.

4.2.2 The case $c \in \pi(s, s') \setminus \{s'\}$

We now consider the case $c \in \pi(s, s') \setminus \{s'\}$. For this case, Oh and Ahn [19] (see the proof of Lemma 3.6 [19]) claimed that α does not exist. However, this is not correct; see Fig. 7 for a counterexample.

Let v be the vertex incident to s' in $\pi(s', r')$. To make the notation consistent with the previous subcase, we let $u = s'$. We have the following lemma (e.g., see Fig. 8), which is analogous to Lemma 10. The proof is omitted.

▶ **Lemma 13.**

1. *α must be in the geodesic triangle $\triangle(s, r, \beta(t, r))$.*
2. *The apexes of $\triangle(s, r, \beta(t, r))$ are s'', v, and $\beta(t, r)$, where s'' be the junction vertex of $\pi(s, r)$ and $\pi(s, \beta(t, r))$.*
3. *p_s must be on the pseudo-convex chain $\pi(s'', \beta(t, r)) \cup \pi(s'', v)$ and $\overline{\alpha p_s}$ is tangent to the chain.*
4. *p_r must be on the convex chain $\pi(v, \beta(t, r))$ and $\overline{\alpha p_r}$ is tangent to the chain.*
5. *p_t must be on pseudo-convex chain $\pi(t_{uv}, u) \cup \pi(t_{uv}, v)$ and $\overline{\alpha p_t}$ is tangent to the chain, where t_{uv} is the junction vertex of $\pi(t, u)$ and $\pi(t, v)$.*
6. *$\overline{\alpha p_t}$ must intersect $\overline{uv}$.*

Due to the preceding lemma, our algorithm works as follows. First, we compute the vertices s'', u, v, and t_{uv}, which can be done in $O(\log n)$ time by the GH data structure. Then we apply the tentative prune-and-search technique [15] on the three pseudo-convex chains specified in the lemma in a similar way as before to compute α in $O(\log n)$ time. Finally, we validate α in $O(\log n)$ time in a similar way as before. The overall time of the algorithm is $O(\log n)$. Lemma 1 is thus proved.

▶ Remark. As discussed above, there are two mistakes in the algorithm of Oh and Ahn [19] (Lemma 3.6): (1) In the subcase $c \in \pi(s', r')$, the case where not both u and v are polygon vertices is missed; (2) in the subcase $c \notin \pi(s', r')$, they erroneously claimed that α does not exist. Both mistakes can be corrected with our new results. Indeed, in both cases we have proved that $\overline{\alpha p_t}$ intersects $\overline{uv}$. With this critical property, their algorithm of Lemma 3.6 [19] (which was originally designed for the case where $c \in \pi(s', r')$ and both u and v are polygon vertices) can be applied to compute α in $O(\log^2 n)$ time. In this way, Lemma 3.6 of [19] is remedied and thus all other results of [19] that rely on Lemma 3.6 are not affected.

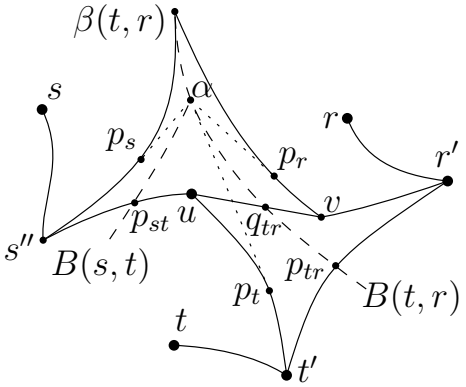


Figure 8 Illustrating Lemma 13.

References

1 H.-K. Ahn, L. Barba, P. Bose, J.-L. De Carufel, M. Korman, and E. Oh. A linear-time algorithm for the geodesic center of a simple polygon. *Discrete and Computational Geometry*, 56:836–859, 2016.

2 B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4:109–140, 1989.

3 B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete and Computational Geometry*, 9:217–255, 1993.

4 T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, Montreal, Canada, 1985.

5 S.W. Bae and K.Y. Chwa. The geodesic farthest-site Voronoi diagram in a polygonal domain with holes. In *Proceedings of the 25th ACM Symposium on Computational Geometry (SoCG)*, pages 198–207, 2009.

6 S.W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains. *Discrete and Computational Geometry*, 50:306–329, 2013.

7 S.W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal domain. *Computational Geometry: Theory and Applications*, 77:3–9, 2019.

8 L. Barba. Optimal algorithm for geodesic farthest-point Voronoi diagrams. In *Proceedings of the 35th International Symposium on Computational Geometry (SoCG)*, pages 12:1–12:14, 2019.

9 B. Chazelle. A theorem on polygon cutting with applications. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 339–349, 1982.

10 B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink. Ray shooting in polygons using geodesic triangulations. *Algorithmica*, 12(1):54–68, 1994.

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

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

13 J. Hershberger and S. Suri. Matrix searching with the shortest-path metric. *SIAM Journal on Computing*, 26(6):1612–1634, 1997.

14 J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

15 D. Kirkpatrick and J. Snoeyink. Tentative prune-and-search for computing fixed-points with applications to geometric computation. *Fundamenta Informaticae*, 22(4):353–370, 1995.

16 C.-H. Liu. A nearly optimal algorithm for the geodesic Voronoi diagram of points in a simple polygon. *Algorithmica*, 82:915–937, 2020.

17 J.S.B. Mitchell. Geometric shortest paths and network optimization, in *Handbook of Computational Geometry,* J.-R Sack and J. Urrutia (eds.), pages 633–702. Elsevier, Amsterdam, the Netherlands, 2000.

18 E. Oh. Optimal algorithm for geodesic nearest-point Voronoi diagrams in simple polygons. In *Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 391–409, 2019.

19 E. Oh and H.-K. Ahn. Voronoi diagrams for a moderate-sized point-set in a simple polygon. *Discrete and Computational Geometry*, 63:418–454, 2020.

20 E. Oh, L. Barba, and H.-K. Ahn. The geodesic farthest-point Voronoi diagram in a simple polygon. *Algorithmica*, 82:1434–1473, 2020.

21 E. 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*, 20:319–352, 1998.

22 R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon. *Discrete and Computational Geometry*, 4(1):611–626, 1989.

23 F.P. Preparata and M.I. Shamos. *Computational Geometry.* Springer-Verlag, New York, 1985.

24 S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39:220–235, 1989.

25 G. Toussaint. Computing geodesic properties inside a simple polygon. Technical report, McGill University, Montreal, Canada, 1989.

26 H. Wang. On the geodesic centers of polygonal domains. *Journal of Computational Geometry*, 9:131–190, 2018.