

# A linear-time algorithm for the geodesic center of a simple polygon

Hee-Kap Ahn<sup>\*3</sup>, Luis Barba<sup>1,2</sup>, Prosenjit Bose<sup>1</sup>, Jean-Lou de Carufel<sup>1</sup>, Matias Korman<sup>4,5</sup>, and Eunjin Oh<sup>3</sup>

<sup>1</sup> School of Computer Science, Carleton University, Ottawa, Canada.

`jit@scs.carleton.ca, jdecaruf@cg.scs.carleton.ca`

<sup>2</sup> Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium.

`lbarbafl@ulb.ac.be`

<sup>3</sup> Department of Computer Science and Engineering, POSTECH,

77 Cheongam-Ro, Nam-Gu, Pohang, Gyeongbuk, Korea.

`{heekap, jin9082}@postech.ac.kr`

<sup>4</sup> National Institute of Informatics (NII), Tokyo, Japan.

`korman@nii.ac.jp`

<sup>5</sup> JST, ERATO, Kawarabayashi Large Graph Project.

## Abstract

Given two points in a simple polygon  $P$  of  $n$  vertices, its geodesic distance is the length of the shortest path that connects them among all paths that stay within  $P$ . The geodesic center of  $P$  is the unique point in  $P$  that minimizes the largest geodesic distance to all other points of  $P$ . In 1989, Pollack, Sharir and Rote [Disc. & Comput. Geom. 89] showed an  $O(n \log n)$ -time algorithm that computes the geodesic center of  $P$ . Since then, a longstanding question has been whether this running time can be improved (explicitly posed by Mitchell [Handbook of Computational Geometry, 2000]). In this paper we affirmatively answer this question and present a linear time algorithm to solve this problem.

## 1 Introduction

Let  $P$  be a simple polygon with  $n$  vertices. Given two points  $x, y$  in  $P$ , the *geodesic path*  $\pi(x, y)$  is the shortest-path contained in  $P$  connecting  $x$  with  $y$ . If the straight-line segment connecting  $x$  with  $y$  is contained in  $P$ , then  $\pi(x, y)$  is a straight-line segment. Otherwise,  $\pi(x, y)$  is a polygonal chain whose vertices (other than its endpoints) are reflex vertices of  $P$ . We refer the reader to [19] for more information on geodesic paths.

The *geodesic distance* between  $x$  and  $y$ , denoted by  $|\pi(x, y)|$ , is the sum of the Euclidean lengths of each segment in  $\pi(x, y)$ . Throughout this paper, when referring to the distance between two points in  $P$ , we refer to the geodesic distance between them. Given a point  $x \in P$ , a (geodesic) *farthest neighbor* of  $x$ , is a point  $f_P(x)$  (or simply  $f(x)$ ) of  $P$  whose geodesic distance to  $x$  is maximized. To ease the description, we assume that each vertex of  $P$  has a unique farthest neighbor. We can make this *general position* assumption using simulation of simplicity [9].

Let  $F_P(x)$  be the function that, for each  $x \in P$ , maps to the distance to a farthest neighbor of  $x$  (i.e.,  $F_P(x) = |\pi(x, f(x))|$ ). A point  $c_P \in P$  that minimizes  $F_P(x)$  is called the *geodesic center* of  $P$ . Similarly, a point  $s \in P$  that maximizes  $F_P(x)$  (together with its farthest neighbor) is called a *geodesic diametral pair* and their distance is known as

\* The work by H.-K. Ahn and E. Oh was supported by the NRF grant 2011-0030044 (SRC-GAIA) funded by the Korea government (MSIP).



licensed under Creative Commons License CC-BY



Leibniz International Proceedings in Informatics  
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

the *geodesic diameter*. Asano and Toussaint [3] showed that the geodesic center is unique (whereas it is easy to see that several geodesic diametral pairs may exist).

In this paper, we show how to compute the geodesic center of  $P$  in  $O(n)$  time.

## 1.1 Previous Work

Since the early 1980s the problem of computing the geodesic center (and its counterpart, the geodesic diameter) has received a lot of attention from the computational geometry community. Chazelle [7] gave the first algorithm for computing the geodesic diameter (which runs in  $O(n^2)$  time using linear space). Afterwards, Suri [24] reduced it to  $O(n \log n)$ -time without increasing the space constraints. Finally, Hershberger and Suri [13] presented a fast matrix search technique, one application of which is a linear-time algorithm for computing the diameter.

The first algorithm for computing the geodesic center was given by Asano and Toussaint [3], and runs in  $O(n^4 \log n)$ -time. In 1989, Pollack, Sharir, and Rote [22] improved it to  $O(n \log n)$  time. Since then, it has been an open problem whether the geodesic center can be computed in linear time (indeed, this problem was explicitly posed by Mitchell [19, Chapter 27]).

Several other variations of these two problems have been considered. Indeed, the same problem has been studied under different metrics. Namely, the  $L_1$  geodesic distance [6], the link distance [23, 14, 8] (where we look for the path with the minimum possible number of bends or *links*), or even rectilinear link distance [20, 21] (a variation of the link distance in which only isothetic segments are allowed). The diameter and center of a simple polygon for both the  $L_1$  and rectilinear link metrics can be computed in linear time (whereas  $O(n \log n)$  time is needed for the link distance).

Another natural extension is the computation of the diameter and center in polygonal domains (i.e., polygons with one or more holes). Polynomial time algorithms are known for both the diameter [4] and center [5], although the running times are significantly larger (i.e.,  $O(n^{7.73})$  and  $O(n^{12+\varepsilon})$ , respectively).

## 1.2 Outline

In order to compute the geodesic center, Pollack *et al.* [22] introduce a linear time *chord-oracle*. Given a chord  $C$  that splits  $P$  into two sub-polygons, the oracle determines which sub-polygon contains  $c_P$ . Combining this operation with an efficient search on a triangulation of  $P$ , Pollack *et al.* narrow the search of  $c_P$  within a triangle (and find the center using optimization techniques). Their approach however, does not allow them to reduce the complexity of the problem in each iteration, and hence it runs in  $\Theta(n \log n)$  time.

The general approach of our algorithm described in Section 6 is similar: partition  $P$  into  $O(1)$  cells, use an oracle to determine which cell contains  $c_P$ , and recurse within the cell. Our approach differs however in two important aspects that allows us to speed-up the algorithm. First, we do not use the chords of a triangulation of  $P$  to partition the problem into cells. We use instead a cutting of a suitable set of chords. Secondly, we compute a set  $\Phi$  of  $O(n)$  functions, each defined in a triangular domain contained in  $P$ , such that their upper envelope,  $\phi(x)$ , coincides with  $F_P(x)$ . Thus, we can “ignore” the polygon  $P$  and focus only on finding the minimum of the function  $\phi(x)$ .

The search itself uses  $\varepsilon$ -nets and cutting techniques, which certify that both the size of the cell containing  $c_P$  and the number of functions of  $\Phi$  defined in it decrease by a constant fraction (and thus leads to an overall linear time algorithm). This search has however two stopping conditions, (1) reach a subproblem of constant size, or (2) find a convex trapezoid

containing  $c_P$ . In the latter case, we show that  $\phi(x)$  is a convex function when restricted to this convex trapezoid. Thus, finding its minimum becomes an optimization problem that we solve in Section 7 using cuttings in  $\mathbb{R}^3$ .

The key of this approach lies in the computation of the functions of  $\Phi$  and their triangular domains. Each function  $g(x)$  of  $\Phi$  is defined in a triangular domain  $\triangle$  contained in  $P$  and is associated to a particular vertex  $w$  of  $P$ . Intuitively speaking,  $g(x)$  maps points in  $\triangle$  to their (geodesic) distance to  $w$ . We guarantee that, for each point  $x \in P$ , there is one function  $g$  defined in a triangle containing  $x$ , such that  $g(x) = F_P(x)$ . To compute these triangles and their corresponding functions, we proceed as follows.

In Section 3, we use the matrix search technique introduced by Hershberger and Suri [13] to decompose the boundary of  $P$ , denoted by  $\partial P$ , into connected edge disjoint chains. Each chain is defined by either (1) a consecutive list of vertices that have the same farthest neighbor  $v$  (we say that  $v$  is *marked* if it has such a chain associated to it), or (2) an edge whose endpoints have different farthest neighbors (such edge is called a *transition edge*).

In Section 4, we consider each transition edge  $ab$  of  $\partial P$  independently and compute its *hourglass*. Intuitively, the hourglass of  $ab$ ,  $H_{ab}$ , is the region of  $P$  between two chains, the edge  $ab$  and the chain of  $\partial P$  that contains the farthest neighbors of all points in  $ab$ . Inspired by a result of Suri [24], we show that the sum of the complexities of each hourglass defined on a transition edge is  $O(n)$ . In addition, we provide a new technique to compute all these hourglasses in linear time.

In Section 5 we show how to compute the functions in  $\Phi$  and their respective triangles. We distinguish two cases: (1) Inside each hourglass  $H_{ab}$  of a transition edge, we use a technique introduced by Aronov et al. [2] that uses the shortest-path trees of  $a$  and  $b$  in  $H_{ab}$  to decompose  $H_{ab}$  into  $O(|H_{ab}|)$  triangles with their respective functions (for more information on shortest-path trees refer to [10]). (2) For each marked vertex  $v$  we compute triangles that encode the distance from  $v$ . Moreover, we guarantee that these triangles cover every point of  $P$  whose farthest neighbor is  $v$ . Overall, we compute the  $O(n)$  functions of  $\Phi$  in linear time.

## 2 Hourglasses and Funnels

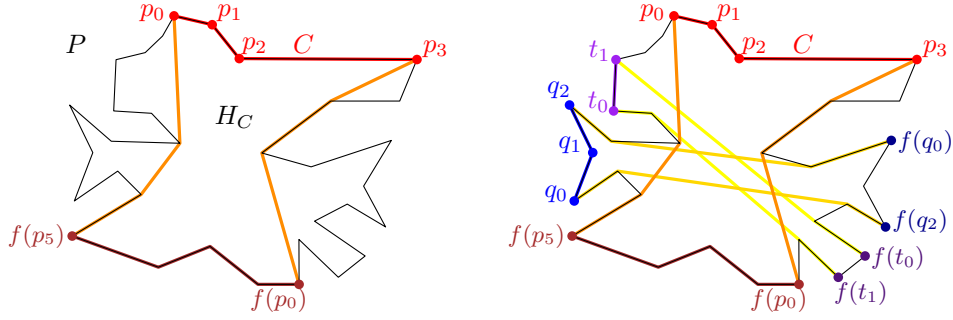
In this section, we introduce the main tools that are going to be used by the algorithm. Some of the results presented in this section have been shown before in different papers. For most of them, we present proof sketches.

### 2.1 Hourglasses

Given two points  $x$  and  $y$  on  $\partial P$ , let  $\partial P(x, y)$  be the polygonal chain that starts at  $x$  and follows the boundary of  $P$  clockwise until reaching  $y$ .

For any polygonal chain  $C = \partial P(p_0, p_1, \dots, p_k)$ , the *hourglass* of  $C$ , denoted by  $H_C$ , is the simple polygon contained in  $P$  bounded by  $C$ ,  $\pi(p_k, f(p_0))$ ,  $\partial P(f(p_0), f(p_k))$  and  $\pi(f(p_k), p_0)$ ; see Figure 1. We call  $C$  and  $\partial P(f(p_0), f(p_k))$  the *top* and *bottom* chains of  $H_C$ , respectively, while  $\pi(p_k, f(p_0))$  and  $\pi(f(p_k), p_0)$  are referred to as the *walls* of  $H_C$ .

We say that the hourglass  $H_C$  is *open* if its walls are vertex disjoint. We say  $C$  is a *transition chain* if  $f(p_0) \neq f(p_k)$  and neither  $f(p_0)$  nor  $f(p_k)$  are interior vertices of  $C$ . In particular, if an edge  $ab$  of  $\partial P$  is a transition chain, we say that it is a *transition edge* (see Figure 1).



■ **Figure 1** Given two edge disjoint transition chains, their hourglasses are open and the bottom chains of their hourglasses are also edge disjoint. Moreover, these bottom chains appear in the same cyclic order as the top chains along  $\partial P$ .

129 ► **Lemma 1.** [Rephrase of Lemma 3.1.3 of [2]] If  $C$  is a transition chain of  $\partial P$ , then the  
 130 hourglass  $H_C$  is an open hourglass.

131 Note that by Lemma 1, the hourglass of each transition chain is open. In the remainder  
 132 of the paper, all the hourglasses considered are defined by a transition chain, i.e., they are  
 133 open and their top and bottom chains are edge disjoint.

134 The following lemma is depicted in Figure 1 and is a direct consequence of the Ordering  
 135 Lemma proved by Aronov et al. [2, Corollary 2.7.4].

136 ► **Lemma 2.** Let  $C_1, C_2, C_3$  be three edge disjoint transition chains of  $\partial P$  that appear in this  
 137 order when traversing clockwise the boundary of  $P$ . Then, the bottom chains of  $H_{C_1}, H_{C_2}$  and  
 138  $H_{C_3}$  are also edge disjoint and appear in this order when traversing clockwise the boundary  
 139 of  $P$ .

140 Let  $\gamma$  be a geodesic path joining two points on the boundary of  $P$ . We say that  $\gamma$  *separates*  
 141 two points  $x_1$  and  $x_2$  of  $\partial P$  if the points of  $X = \{x_1, x_2\}$  and the endpoints of  $\gamma$  alternate  
 142 along the boundary of  $P$  ( $x_1$  and  $x_2$  could coincide with the endpoints of  $\gamma$  in degenerate  
 143 cases). We say that a geodesic  $\gamma$  *separates* an hourglass  $H$  if it separates the points of its  
 144 top chain from those of its bottom chain.

145 ► **Lemma 3.** Let  $C_1, \dots, C_r$  be edge disjoint transition chains of  $\partial P$ . Then, there is a set  
 146 of  $t = O(1)$  geodesic paths  $\gamma_1, \dots, \gamma_t$  with endpoints on  $\partial P$  such that for each  $1 \leq i \leq r$  there  
 147 exists  $1 \leq j \leq t$  such that  $\gamma_j$  separates  $H_{C_i}$ . Moreover, this set can be computed in  $O(n)$   
 148 time.

149 **Proof.** Aronov et al. showed that there exist four vertices  $v_1, \dots, v_4$  of  $P$  and geodesic paths  
 150  $\pi(v_1, v_2), \pi(v_2, v_3), \pi(v_3, v_4)$  such that for any point  $x \in \partial P$ , one of these paths separates  $x$   
 151 from  $f(x)$  [2, Lemma 2.7.6]. Moreover, they show how to compute this set in  $O(n)$  time.

152 Let  $\Gamma = \{\pi(v_i, v_j) : 1 \leq i < j \leq 4\}$  and note that  $v_1, \dots, v_4$  split the boundary of  $P$  into  
 153 at most four connected components. If a chain  $C_i$  is completely contained in one of these  
 154 components, then one path of  $\Gamma$  separates the top and bottom chain of  $H_{C_i}$ . Otherwise,  
 155 some vertex  $v_j$  is an interior vertex of  $C_i$ . However, because the chains  $C_1, \dots, C_r$  are edge  
 156 disjoint, there are at most four chains in this situation. For each chain  $C_i$  containing a vertex  
 157  $v_j$ , we add the geodesic path connecting the endpoints of  $C_i$  to  $\Gamma$ . Therefore,  $\Gamma$  consists  
 158 of  $O(1)$  geodesic paths and each hourglass  $H_{C_i}$  has its top and bottom chain separated by  
 159 some path of  $\Gamma$ . Since only  $O(1)$  additional paths are computed, this can be done in linear  
 160 time. ◀

161 A *chord* of  $P$  is an edge joining two non-adjacent vertices  $a$  and  $b$  of  $P$  such that  $ab \subseteq P$ .  
 162 Therefore, a chord splits  $P$  into two sub-polygons.

163 ► **Lemma 4.** [Rephrase of Lemma 3.4.3 of [2]] Let  $C_1, \dots, C_r$  be a set of edge disjoint  
 164 transition chains of  $\partial P$  that appear in this order when traversing clockwise the boundary of  
 165  $P$ . Then each chord of  $P$  appears in  $O(1)$  hourglasses among  $H_{C_1}, \dots, H_{C_r}$ .

166 **Proof.** Note that chords can only appear on walls of hourglasses. Because hourglasses are  
 167 open, any chord must be an edge on exactly one wall of each of these hourglasses. Assume,  
 168 for the sake of contradiction, that there exists two points  $s, t \in P$  whose chord  $st$  is in three  
 169 hourglasses  $H_{C_i}, H_{C_j}$  and  $H_{C_k}$  (for some  $1 \leq i < j < k \leq r$ ) such that  $s$  visited before  $t$   
 170 when going from the top chain to the bottom one along the walls of the three hourglasses.  
 171 Let  $s_i$  and  $t_i$  be the points in the in the top and bottom chains of  $H_{C_i}$ , respectively, such  
 172 that  $\pi(s_i, t_i)$  is the wall of  $H_{C_i}$  that contains  $st$  (analogously, we define  $s_k$  and  $t_k$ )

173 Because  $C_j$  lies in between  $C_i$  and  $C_k$ , Lemma 2 implies that the bottom chain of  $C_j$   
 174 appears between the bottom chains of  $C_i$  and  $C_k$ . Therefore,  $C_j$  lies between  $s_i$  and  $s_k$  and  
 175 the bottom chain of  $H_{C_j}$  lies between  $t_i$  and  $t_k$ . That is, for each  $x \in C_j$  and each  $y$  in the  
 176 bottom chain of  $H_{C_j}$ , the geodesic path  $\pi(x, y)$  is “sandwiched” by the paths  $\pi(s_i, t_i)$  and  
 177  $\pi(s_k, t_k)$ . In particular,  $\pi(x, y)$  contains  $st$  for each pair of points in the top and bottom  
 178 chain of  $H_{C_j}$ . However, this implies that the hourglass  $H_{C_j}$  is not open—a contradiction  
 179 that comes from assuming that  $st$  lies in the wall of three open hourglasses, when this wall  
 180 is traversed from the top chain to the bottom chain. Analogous arguments can be used to  
 181 bound the total number of walls that contain the edge  $st$  (when traversed in any direction)  
 182 to  $O(1)$ . ◀

183 ► **Lemma 5.** Let  $x, u, y, v$  be four vertices of  $P$  that appear in this cyclic order in a clockwise  
 184 traversal of  $\partial P$ . Given the shortest-path trees  $T_x$  and  $T_y$  of  $x$  and  $y$  in  $P$ , respectively, such  
 185 that  $T_x$  and  $T_y$  can answer lowest common ancestor (LCA) queries in  $O(1)$  time, we can  
 186 compute the path  $\pi(u, v)$  in  $O(|\pi(u, v)|)$  time. Moreover, all edges of  $\pi(u, v)$ , except perhaps  
 187 one, belong to  $T_x \cup T_y$ .

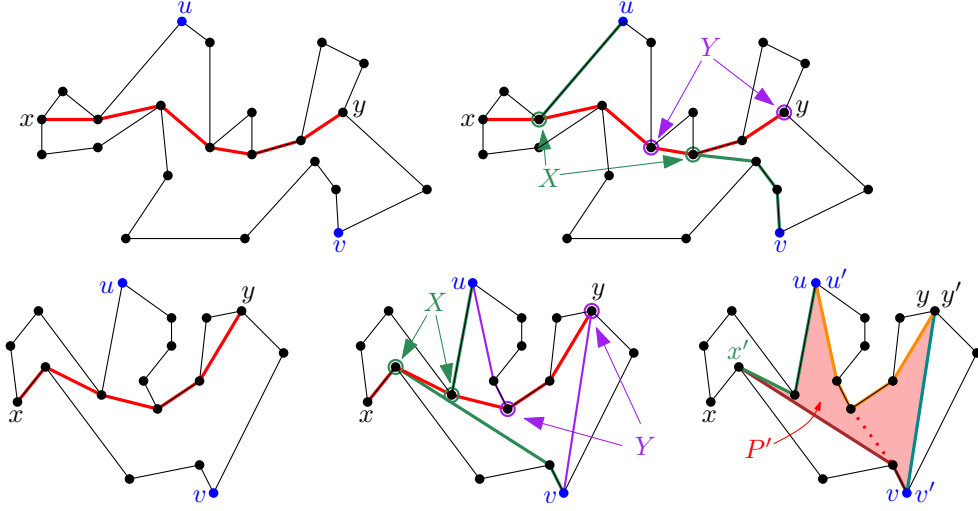
188 **Proof.** Let  $X$  (resp.  $Y$ ) be the set containing the LCA in  $T_x$  (resp.  $T_y$ ) of  $u, y$ , and of  $v, y$   
 189 (resp.  $u, x$  and  $x, y$ ). Note that the points of  $X \cup Y$  lie on the path  $\pi(x, y)$  and can be  
 190 computed in  $O(1)$  time by hypothesis. Moreover, using LCA queries, we can decide their  
 191 order along the path  $\pi(x, y)$  when traversing it from  $x$  to  $y$ . (Both  $X$  and  $Y$  could consist of  
 192 a single vertex in some degenerate situations). Two cases arise:

193 **Case 1.** If there is a vertex  $x^* \in X$  lying after a vertex  $y^* \in Y$  along  $\pi(x, y)$ , then the  
 194 path  $\pi(u, v)$  contains the path  $\pi(y^*, x^*)$ . In this case, the path  $\pi(u, v)$  is the concatenation of  
 195 the paths  $\pi(u, y^*)$ ,  $\pi(y^*, x^*)$ , and  $\pi(x^*, v)$  and that the three paths are contained in  $T_x \cup T_y$ .  
 196 Moreover,  $\pi(u, v)$  can be computed in time proportional to its length by traversing along the  
 197 corresponding tree; see Figure 2 (top).

198 **Case 2.** In this case the vertices of  $X$  appear before the vertices of  $Y$  along  $\pi(x, y)$ . Let  
 199  $x'$  (resp.  $y'$ ) be the vertex of  $X$  (resp.  $Y$ ) closest to  $x$  (resp.  $y$ ).

200 Let  $u'$  be the last vertex of  $\pi(u, x)$  that is also in  $\pi(u, y)$ . Note that  $u'$  can be constructed  
 201 by walking from  $u'$  towards  $x$  until the path towards  $y$  diverges. Thus,  $u'$  can be computed  
 202 in  $O(|\pi(u, u')|)$  time. Define  $v'$  analogously and compute it in  $O(|\pi(v, v')|)$  time.

203 Let  $P'$  be the polygon bounded by the geodesic paths  $\pi(x', u')$ ,  $\pi(u', y')$ ,  $\pi(y', v')$  and  
 204  $\pi(v', x')$ . Because the vertices of  $X$  appear before those of  $Y$  along  $\pi(x, y)$ ,  $P'$  is a simple  
 205 polygon; see Figure 2 (bottom).



■ **Figure 2** (top) Case 1 of the proof of Lemma 5 where the path  $\pi(u, v)$  contains a portion of the path  $\pi(x, y)$ . (bottom) Case 2 of the proof of Lemma 5 where the path  $\pi(u, v)$  has exactly one edge being the tangent of the paths  $\pi(u', y')$  and  $\pi(v', x')$ .

206 In this case the path  $\pi(u, y)$  is the union of  $\pi(u, u')$ ,  $\pi(u', v')$  and  $\pi(v', v)$ . Because  $\pi(u, u')$   
 207 and  $\pi(v', v)$  can be computed in time proportional to their length, it suffices to compute  
 208  $\pi(u', v')$  in  $O(|\pi(u', v')|)$  time.

209 Note that  $P'$  is a simple polygon with only four convex vertices  $x', u', y'$  and  $v'$ , which  
 210 are connected by chains of reflex vertices. Thus, the shortest path from  $x'$  to  $y'$  can have at  
 211 most one diagonal edge connecting distinct reflex chains of  $P'$ . Since the rest of the points  
 212 in  $\pi(u', v')$  lie on the boundary of  $P'$  and from the fact that each edge of  $P'$  is an edge of  
 213  $T_x \cup T_y$ , we conclude all edges of  $\pi(u, v)$ , except perhaps one, belong to  $T_x \cup T_y$ .

214 We want to find the common tangent between the reflex paths  $\pi(u', x')$  and  $\pi(v', y')$ , or  
 215 the common tangent of  $\pi(u', y')$  and  $\pi(v', x')$  as one of them belongs to the shortest path  
 216  $\pi(u', v')$ . Assume that the desired tangent lies between the paths  $\pi(u', x')$  and  $\pi(v', y')$ .  
 217 Since these paths consist only of reflex vertices, the problem can be reduced to finding the  
 218 common tangent of two convex polygons. By slightly modifying the linear time algorithm to  
 219 compute this tangents, we can make it run in  $O(|\pi(u', v')|)$  time.

220 Since we do not know if the tangent lies between the paths  $\pi(u', x')$  and  $\pi(v', y')$ , we  
 221 process the chains  $\pi(u', y')$  and  $\pi(v', x')$  in parallel and stop when finding the desired tangent.  
 222 Consequently, we can compute the path  $\pi(u, v)$  in time proportional to its length. ◀

► **Lemma 6.** Let  $P$  be a simple polygon with  $n$  vertices. Given  $k$  disjoint transition chains  $C_1, \dots, C_k$  of  $\partial P$ , it holds that

$$\sum_{i=1}^k |H_{C_i}| = O(n).$$

223 **Proof.** Because the given transition chains are disjoint, Lemma 2 implies that the bottom  
 224 chains of their respective hourglasses are also disjoint. Therefore, the sum of the complexities  
 225 of all the top and bottom chains of these hourglasses is  $O(n)$ . To bound the complexity of  
 226 their walls we use Lemma 4. Since no chord is used more than a constant number of times,  
 227 it suffices to show that the total number of chords used by all these hourglasses is  $O(n)$ .

To prove this, we use Lemma 3 to construct  $O(1)$  *split chains*  $\gamma_1, \dots, \gamma_t$  such that for each  $1 \leq i \leq k$ , there is a split chain  $\gamma_j$  that separates the top and bottom chains of  $H_{C_i}$ . For each  $1 \leq j \leq t$ , let

$$\mathcal{H}^j = \{H_{C_i} : \text{the top and bottom chain of } H_{C_i} \text{ are separated by } \gamma_j\}.$$

Since the complexity of the shortest-path trees of the endpoints of  $\gamma_j$  is  $O(n)$  [10], and from the fact that the chains  $C_1, \dots, C_k$  are disjoint, Lemma 5 implies that the total number of edges in all the hourglasses of  $\mathcal{H}^j$  is  $O(n)$ . Moreover, because each of these edges appears in  $O(1)$  hourglasses among  $C_1, \dots, C_k$ , we conclude that

$$\sum_{H \in \mathcal{H}^j} |H| = O(n).$$

228 Since we have only  $O(1)$  split chains, our result follows. ◀

## 2.2 Funnels

230 Let  $C = (p_0, \dots, p_k)$  be a chain of  $\partial P$  and let  $v$  be a vertex of  $P$  not in  $C$ . The *funnel* of  $v$  to  $C$ , denoted by  $S_v(C)$ , is the simple polygon bounded by  $C$ ,  $\pi(p_k, v)$  and  $\pi(v, p_0)$ ; see Figure 3  
231 (a). Note that the paths  $\pi(v, p_k)$  and  $\pi(v, p_0)$  may coincide for a while before splitting into  
232 disjoint chains. See Lee and Preparata [15] or Guibas et al. [10] for more details on funnels.

233 A subset  $R \subset P$  is *geodesically convex* if for every  $x, y \in R$ , the path  $\pi(x, y)$  is contained  
234 in  $R$ . This funnel  $S_v(C)$  is also known as the geodesic convex hull of  $C$  and  $v$ , i.e., the  
235 minimum geodesically convex set that contains  $v$  and  $C$ .  
236

237 Given two points  $x, y \in P$ , the (geodesic) *bisector* of  $x$  and  $y$  is the set of points contained  
238 in  $P$  that are equidistant from  $x$  and  $y$ . This bisector is a curve, contained in  $P$ , that  
239 consists of circular arcs and hyperbolic arcs. Moreover, this curve intersects  $\partial P$  only at its  
240 endpoints [1, Lemma 3.22].

241 The (farthest) *Voronoi region* of a vertex  $v$  of  $P$  is the set of points  $R(v) = \{x \in P : F_P(x) = |\pi(x, v)|\}$  (including boundary points).  
242

243 ► **Lemma 7.** *Let  $v$  be a vertex of  $P$  and let  $C$  be a transition chain such  $R(v) \cap \partial P \subseteq C$   
244 and  $v \notin C$ . Then,  $R(v)$  is contained in the funnel  $S_v(C)$*

245 **Proof.** Let  $a$  and  $b$  be the endpoints of  $C$  such that  $a, b, f(a)$  and  $f(b)$  appear in this order  
246 in a clockwise traversal of  $\partial P$ . Because  $R(v) \cap \partial P \subseteq C$ , we know that  $v$  lies between  $f(a)$   
247 and  $f(b)$ .

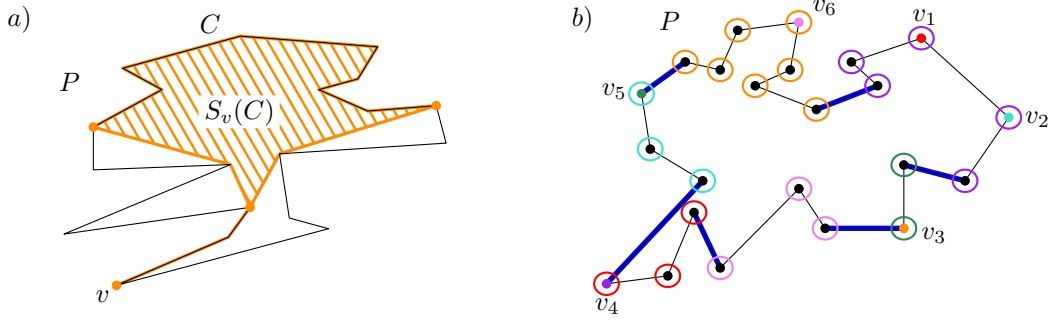
248 Let  $\alpha$  (resp.  $\beta$ ) be the bisector of  $v$  and  $f(a)$  (resp.  $f(b)$ ). Let  $h_a$  (resp.  $h_b$ ) be the set of  
249 points of  $P$  that are farther from  $v$  than from  $f(a)$  (resp.  $f(b)$ ). Note that  $\alpha$  is the boundary  
250 of  $h_a$  while  $\beta$  bounds  $h_b$ .

251 By definition, we know that  $R(v) \subseteq h_a \cap h_b$ . Therefore, it suffices to show that  $h_a \cap h_b \subseteq S_v(C)$ .  
252 Assume for a contradiction that there is a point of  $h_a \cap h_b$  lying outside of  $S_v(C)$ .  
253 By continuity of the geodesic distance, the boundaries of  $h_a \cap h_b$  and  $S_v(C)$  must intersect.  
254 Because  $a \notin h_a$  and  $b \notin h_b$ , both bisectors  $\alpha$  and  $\beta$  must have an endpoint on the edge  
255  $ab$ . Since the boundaries of  $h_a \cap h_b$  and  $S_v(C)$  intersect, we infer that  $\beta \cap \pi(v, b) \neq \emptyset$  or  
256  $\alpha \cap \pi(v, a) \neq \emptyset$ . Without loss of generality, assume that there is a point  $w \in \beta \cap \pi(v, b)$ , the  
257 case where  $w$  lies in  $\alpha \cap \pi(v, a)$  is analogous.

Since  $w \in \beta$ , we know that  $|\pi(w, v)| = |\pi(w, f(b))|$ . By the triangle inequality and since  $w$  cannot be a vertex of  $P$  as  $w$  intersects  $\partial P$  only at its endpoints, we get that

$$|\pi(b, f(b))| < |\pi(b, w)| + |\pi(w, f(b))| = |\pi(b, w)| + |\pi(w, v)| = |\pi(b, v)|.$$





**Figure 3** a) The funnel  $S_v(C)$  of a vertex  $v$  and a chain  $C$  contained in  $\partial P$  are depicted. b) Each vertex of the boundary of  $P$  is assigned with a farthest neighbor which is then marked. The boundary is then decomposed into vertex disjoint chains, each associated with a marked vertex, joined by transition edges (blue) whose endpoints have different farthest neighbors.

Which implies that  $b$  is farther from  $v$  than from  $f(b)$ —a contradiction that comes from assuming that  $h_a \cap h_b$  is not contained in  $S_v(C)$ .  $\blacktriangleleft$

### 3 Decomposing the boundary

In this section, we decompose the boundary of  $P$  into consecutive vertices that share the same farthest neighbor and edges of  $P$  whose endpoints have distinct farthest neighbors.

Using a result from Hershberger and Suri [13], in  $O(n)$  time we can compute the farthest neighbor of each vertex of  $P$ . Recall that the farthest neighbor of each vertex of  $P$  is always a convex vertex of  $P$  [3] and is unique by our general position assumption.

We mark the vertices of  $P$  that are farthest neighbors of at least one vertex of  $P$ . Let  $M$  denote the set of marked vertices of  $P$  (clearly this set can be computed in  $O(n)$  time after applying the result of Hershberger and Suri). In other words,  $M$  contains all vertices of  $P$  whose Voronoi region contains at least one vertex of  $P$ .

Given a vertex  $v$  of  $P$ , the vertices of  $P$  whose farthest neighbor is  $v$  appear contiguously along  $\partial P$  [2]. Therefore, after computing all these farthest neighbors, we effectively split the boundary into subchains, each associated with a different vertex of  $M$ ; see Figure 3 (b).

Let  $a$  and  $b$  be the endpoints of a transition edge of  $\partial P$  such that  $a$  appears before  $b$  in the clockwise order along  $\partial P$ . Because  $ab$  is a transition edge, we know that  $f(a) \neq f(b)$ . Recall that we have computed  $f(a)$  and  $f(b)$  in the previous step and note that  $f(a)$  appears also before  $f(b)$  along this clockwise order. For every vertex  $v$  that lies between  $f(a)$  and  $f(b)$  in the bottom chain of  $H_{ab}$ , we know that there cannot be a vertex  $u$  of  $P$  such that  $f(u) = v$ . As proved by Aronov et al. [2, Corollary 2.7.4], if there is a point  $x$  on  $\partial P$  whose farthest neighbor is  $v$ , then  $x$  must lie on the open segment  $(a, b)$ . In other words, the Voronoi region  $R(v)$  restricted to  $\partial P$  is contained in  $(a, b)$ .

### 4 Building hourglasses

Let  $E$  be the set of transition edges of  $\partial P$ . Given a transition edge  $ab \in E$ , we say that  $H_{ab}$  is a *transition hourglass*. In order to construct the triangle cover of  $P$ , we construct the transition hourglass of each transition edge of  $E$ . By Lemma 6, we know that  $\sum_{ab \in E} |H_{ab}| = O(n)$ . Therefore, our aim is to compute the cover in time proportional to the size of  $H_{ab}$ .

By Lemma 3 we can compute a set of  $O(1)$  separating paths such that for each transition



edge  $ab$ , the transition hourglass  $H_{ab}$  is separated by one (or more) paths in this set. For each endpoint of the  $O(1)$  separating paths we compute its shortest-path tree [10]. In addition, we preprocess these trees in linear time to support LCA queries [12]. Both computations need linear time per endpoint and use  $O(n)$  space. Since we do this process for a constant number of endpoints, overall this preprocessing takes  $O(n)$  time.

Let  $\gamma$  be a separating path whose endpoints are  $x$  and  $y$ . Note that  $\gamma$  separates the boundary of  $P$  into two chains  $S$  and  $S'$  such that  $S \cup S' = \partial P$ . Let  $\mathcal{H}(\gamma)$  be the set of each transition hourglass separated by  $\gamma$  whose transition edge is contained in  $S$  (whenever an hourglass is separated by more than one path, we pick one arbitrarily). Note that we can classify all transition hourglasses into the sets  $\mathcal{H}(\gamma)$  in  $O(n)$  time (since  $O(1)$  separating paths are considered).

We claim that we can compute all transition hourglass of  $\mathcal{H}(\gamma)$  in  $O(n)$  time. By construction, the wall of each of these hourglasses consists of a (geodesic) path that connects a point in  $S$  with a point in  $S'$ . Let  $u \in S$  and  $v \in S'$  be two vertices such that  $\pi(u, v)$  is the wall of a hourglass in  $\mathcal{H}(\gamma)$ . Because LCA queries can be answered in  $O(1)$  time [12], Lemma 5 allows us to compute this path in  $O(|\pi(u, v)|)$  time. Therefore, we can compute all hourglasses of  $\mathcal{H}(\gamma)$  in  $O(\sum_{H \in \mathcal{H}(\gamma)} |H| + n) = O(n)$  time by Lemma 6. Because only  $O(1)$  separating paths are considered, we obtain the following result.

► **Lemma 8.** *We can construct the transition hourglass of all transition edges of  $P$  in  $O(n)$  time.*

## 5 Covering the polygon with apexed triangles

An *apexed triangle*  $\Delta = (a, b, c)$  with *apex*  $a$  is a triangle contained in  $P$  with an associated distance function  $g_\Delta(x)$ , called the *apex function* of  $\Delta$ , such that (1)  $a$  is a vertex of  $P$ , (2)  $b, c \in \partial P$ , and (3) there is a vertex  $w$  of  $P$ , called the *definer* of  $\Delta$ , such that

$$g_\Delta(x) = \begin{cases} -\infty & \text{if } x \notin \Delta \\ |xa| + |\pi(a, w)| = |\pi(x, w)| & \text{if } x \in \Delta \end{cases}$$

In this section, we show how to find a set of  $O(n)$  apexed triangles of  $P$  such that the upper envelope of their apex functions coincides with  $F_P(x)$ . To this end, we first decompose the transition hourglasses into apexed triangles that encode all the geodesic distance information inside them. For each marked vertex  $v \in M$  we construct a funnel that contains the Voronoi region of  $v$ . We then decompose this funnel into apexed triangles that encode the distance from  $v$ .

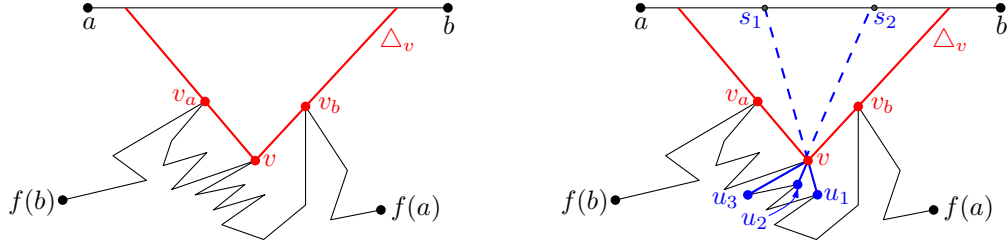
### 5.1 Inside the transition hourglass

Let  $ab$  be a transition edge of  $P$  such that  $b$  is the clockwise neighbor of  $a$  along  $\partial P$ . Let  $B_{ab}$  denote the bottom chain of  $H_{ab}$  after removing its endpoints. As noticed above, a point on  $\partial P$  can be farthest from a vertex in  $B_{ab}$  only if it lies in the open segment  $ab$ . That is, if  $v$  is a vertex of  $B_{ab}$  such that  $R(v) \neq \emptyset$ , then  $R(v) \cap \partial P \subset ab$ .

In fact, not only this Voronoi region is inside  $H_{ab}$  when restricted to the boundary of  $P$ , but also  $R(v) \subset H_{ab}$ . The next result follows trivially from Lemma 7.

► **Corollary 9.** *Let  $v$  be a vertex of  $B_{ab}$ . If  $R(v) \neq \emptyset$ , then  $R(v) \subset H_{ab}$ .*

Our objective is to compute  $O(|H_{ab}|)$  apexed triangles that cover  $H_{ab}$ , each with its distance function, such that the upper envelope of these apex functions coincides with  $F_P(x)$  restricted to  $H_{ab}$  where it “matters”.



■ **Figure 4** (left) A vertex  $v$  visible from the segment  $ab$  lying on the bottom chain of  $H_{ab}$ , and the triangle  $\Delta_v$  which contains the portion of  $ab$  visible from  $v$ . (right) The children  $u_1$  and  $u_2$  of  $v$  are visible from  $ab$  while  $u_3$  is not. The triangle  $\Delta_v$  is split into apexed triangles by the rays going from  $u_1$  and  $u_2$  to  $v$ .

325 The same approach was already used by Pollack et al. in [22, Section 3]. Given a segment  
 326 contained in the interior of  $P$ , they show how to compute a linear number of apexed triangles  
 327 such that  $F_P(x)$  coincides with the upper envelope of the corresponding apex functions in  
 328 the given segment.

329 While the construction we follow is analogous, we use it in the transition hourglass  $H_{ab}$   
 330 instead of the full polygon  $P$ . Therefore, we have to specify what is the relation between the  
 331 upper envelope of the computed functions and  $F_P(x)$ . We will show that the upper envelope  
 332 of the apex functions computed in  $H_{ab}$  coincides with  $F_P(x)$  inside the Voronoi region  $R(v)$   
 333 of every vertex  $v \in B_{ab}$ .

334 Let  $T_a$  and  $T_b$  be the shortest-path trees in  $H_{ab}$  from  $a$  and  $b$ , respectively. Assume  
 335 that  $T_a$  and  $T_b$  are rooted at  $a$  and  $b$ , respectively. We can compute these trees in  $O(|H_{ab}|)$   
 336 time [10]. For each vertex  $v$  between  $f(a)$  and  $f(b)$ , let  $v_a$  and  $v_b$  be the neighbors of  $v$   
 337 in the paths  $\pi(v, a)$  and  $\pi(v, b)$ , respectively. We say that a vertex  $v$  is *visible* from  $ab$  if  
 338  $v_a \neq v_b$ . Note that if a vertex is visible, then the extension of these segments must intersect  
 339 the top segment  $ab$ . Therefore, for each visible vertex  $v$ , we obtain a triangle  $\Delta_v$  as shown in  
 340 Figure 4.

341 We further split  $\Delta_v$  into a series of triangles with apex at  $v$  as follows: Let  $u$  be a child  
 342 of  $v$  in either  $T_a$  or  $T_b$ . As noted by Pollack et al.,  $v$  can be of three types, either (1)  $u$  is not  
 343 visible from  $ab$  (and is hence a child of  $v$  in both  $T_a$  and  $T_b$ ); or (2)  $u$  is visible from  $ab$ , is a  
 344 child of  $v$  only in  $T_b$ , and  $v_bvu$  is a left turn; or (3)  $u$  is visible from  $ab$ , is a child of  $v$  only in  
 345  $T_a$ , and  $v_avu$  is a right turn.

346 Let  $u_1, \dots, u_{k-1}$  be the children of  $v$  of type (2) sorted in clockwise order around  $v$ . Let  
 347  $c(v)$  be the maximum distance from  $v$  to any invisible vertex in the subtrees of  $T_a$  and  $T_b$   
 348 rooted at  $v$ ; if no such vertex exists, then  $c(v) = 0$ . Define a function  $d_l(v)$  on each vertex  $v$   
 349 of  $H_{ab}$  in a recursive fashion as follows: If  $v$  is invisible from  $ab$ , then  $d_l(v) = c(v)$ . Otherwise,  
 350 let  $d_l(v)$  be the maximum of  $c(v)$  and  $\max\{d_l(u_i) + |u_iv| : u_i \text{ is a child of } v \text{ of type (2)}\}$ .  
 351 Similarly we define a symmetric function  $d_r(v)$  using the children of type (3) of  $v$ .

For each  $1 \leq i \leq k-1$ , extend the segment  $u_iv$  past  $v$  until it intersects  $ab$  at a point  
 $s_i$ . Let  $s_0$  and  $s_k$  be the intersections of the extensions of  $vv_a$  and  $vv_b$  with the segment  $ab$ .  
 We define then  $k$  triangles contained in  $\Delta_v$  as follows. For each  $0 \leq i \leq k-1$ , consider the  
 triangle  $\Delta(s_i, v, s_{i+1})$  whose associated apexed (left) function is

$$f_i(x) = \begin{cases} |xv| + \max_{j>i}\{c(v), |vu_j| + d_l(u_j)\} & \text{if } x \in \Delta(s_i, v, s_{i+1}) \\ -\infty & \text{otherwise} \end{cases}$$

352 In a symmetric manner, we define a set of apexed triangles induced by the type (3) children  
 353 of  $v$  and their respective apexed (right) functions.

Let  $g_1, \dots, g_r$  and  $\Delta_1, \dots, \Delta_r$  respectively be an enumeration of all the generated apex functions and triangles such that  $g_i$  is defined in the triangle  $\Delta_i$ . Because each function is determined uniquely by a pair of adjacent vertices in  $T_a$  or in  $T_b$ , and since these trees have  $O(|H_{ab}|)$  vertices, we conclude that  $r = O(|H_{ab}|)$ .

Note that for each  $1 \leq i \leq r$ , the triangle  $\Delta_i$  has two vertices on the segment  $ab$  and a third vertex, say  $a_i$ , called its *apex* such that for each  $x \in \Delta_i$ ,  $g_i(x) = |\pi(x, w_i)|$  for some vertex  $w_i$  of  $H_{ab}$ . We refer to  $w_i$  as the *definer* of  $\Delta_i$ . Intuitively,  $\Delta_i$  defines a portion of the geodesic distance function from  $w_i$  in a constant complexity region.

► **Lemma 10.** *Given a transition edge  $ab$  of  $P$ , we can compute a set  $\mathcal{A}_{ab}$  of  $O(|H_{ab}|)$  apexed triangles in  $O(|H_{ab}|)$  time with the property that for any point  $p \in P$  such that  $f(p) \in B_{ab}$ , there is an apexed triangle  $\Delta \in \mathcal{A}_{ab}$  with apex function  $g$  and definer equal to  $f(p)$  such that*

1.  $p \in \Delta$  and
2.  $g(p) = F_P(p)$ .

**Proof.** Because  $p \in R(f(p))$ , Lemma 9 implies that  $p \in H_{ab}$ . Consider the path  $\pi(p, f(p))$  and let  $v$  be the neighbor of  $p$  along this path. By construction of  $\mathcal{A}_{ab}$ , there is a triangle  $\Delta \in \mathcal{A}_{ab}$  apexed at  $v$  with definer  $w$  that contains  $p$ . The apex function  $g(x)$  of  $\Delta$  encodes the geodesic distance from  $x$  to  $w$ . Because  $F_P(x)$  is the upper envelope of all the geodesic functions, we know that  $g(p) \leq F_P(p)$ .

To prove the other inequality, note that if  $v = f(p)$ , then trivially  $g(p) = |pv| + |\pi(v, w)| \geq |pv| = |\pi(p, f(p))| = F_P(p)$ . Otherwise, let  $z$  be the next vertex after  $v$  in the path  $\pi(p, f(p))$ . Three cases arise:

(a) If  $z$  is invisible from  $ab$ , then so is  $f(p)$  and hence,

$$|\pi(p, f(p))| = |pv| + |\pi(v, f(p))| \leq |pv| + c(v) \leq g(p).$$

(b) If  $z$  is a child of type (2), then  $z$  plays the role of some child  $u_j$  of  $v$  in the notation used during the construction. In this case:

$$|\pi(p, f(p))| = |pv| + |vz| + |\pi(z, f(p))| \leq |pv| + |vu_j| + d_l(u_j) \leq g(p).$$

(c) If  $z$  is a child of type (3), then analogous arguments hold using the (right) distance  $d_r$ .

Therefore, regardless of the case  $F_P(p) = |\pi(p, f(p))| \leq g(p)$ .

To bound the running time, note that the recursive functions  $d_l, d_r$  and  $c$  can be computed in  $O(|T_a| + |T_b|)$  time. Then, for each vertex visible from  $ab$ , we can process it in time proportional to its degree in  $T_a$  and  $T_b$ . Because the sum of the degrees of all vertices in  $T_a$  and  $T_b$  is  $O(|T_a| + |T_b|)$  and from the fact that both  $|T_a|$  and  $|T_b|$  are  $O(|H_{ab}|)$ , we conclude that the total running time to construct  $\mathcal{A}_{ab}$  is  $O(|H_{ab}|)$ . ◀

In other words, Lemma 10 says that no information on farthest neighbors is lost if we only consider the functions in  $\mathcal{A}_{ab}$  within  $H_{ab}$ . In the next section we use a similar approach to construct a set of apexed triangles (and their corresponding apex functions), so as to encode the distance from the vertices of  $M$ .

## 5.2 Inside the funnels of marked vertices

Recall that for each marked vertex  $v \in M$ , we know at least of one vertex on  $\partial P$  such that  $v$  is its farthest neighbor. For any marked vertex  $v$ , let  $u_1, \dots, u_{k-1}$  be the vertices of  $P$  such that  $v = f(u_i)$  and assume that they appear in this order when traversing  $\partial P$  clockwise. Let  $u_0$  and  $u_k$  be the neighbors of  $u_1$  and  $u_{k-1}$  other than  $u_2$  and  $u_{k-2}$ , respectively. Note that

both  $u_0u_1$  and  $u_{k-1}u_k$  are transition edges of  $P$ . Thus, we can assume that their transition hourglasses have been computed.

Let  $C_v = (u_0, \dots, u_k)$  and consider the funnel  $S_v(C_v)$ . We call  $C_v$  the *main chain* of  $S_v(C_v)$  while  $\pi(u_k, v)$  and  $\pi(v, u_0)$  are referred to as the *walls* of the funnel. Because  $v = f(u_1) = f(u_{k-1})$ , we know that  $v$  is a vertex of both  $H_{u_0u_1}$  and  $H_{u_{k-1}u_k}$ . By definition, we have  $\pi(v, u_0) \subset H_{u_0u_1}$  and  $\pi(v, u_k) \subset H_{u_{k-1}u_k}$ . Thus, we can explicitly compute both paths  $\pi(v, u_0)$  and  $\pi(v, u_k)$  in  $O(|H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$  time. So, overall, the funnel  $S_v(C_v)$  can be constructed in  $O(k + |H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$  time. Recall that, by Lemma 6, the total sum of the complexities of the transition hourglasses is  $O(n)$ . In particular, we can bound the total time needed to construct the funnels of all marked vertices by  $O(n)$ .

Since the complexity of the walls of these funnels is bounded by the complexity of the transition hourglasses used to compute them, we get that

$$\sum_{v \in M} |S_v(C_v)| = O\left(n + \sum_{ab \in E} |H_{ab}|\right) = O(n).$$

► **Lemma 11.** *Let  $x$  be a point in  $P$ . If  $v = f(x)$  is a vertex of  $M$ , then  $x \in S_v(C_v)$ .*

**Proof.** Since  $f(u_0) \neq f(u_k)$ ,  $C_v$  is a transition chain. Moreover,  $C_v$  contains  $R(v) \cap \partial P$  by definition. Therefore, Lemma 7 implies that  $R(v) \subset S_v(C_v)$ . Since  $v = f(x)$ , we know that  $x \in R(v)$  and hence that  $x \in S_v(C_v)$ . ◀

We now proceed to split a given funnel into  $O(|S_v(C_v)|)$  apexed triangles that encode the distance function from  $v$ . To this end, we compute the shortest-path tree  $T_v$  of  $v$  in  $S_v(C_v)$  in  $O(|S_v(C_v)|)$  time [11]. We consider the tree  $T_v$  to be rooted at  $v$  and assume that for each node  $u$  of this tree we have stored the geodesic distance  $|\pi(u, v)|$ .

Start an Eulerian tour from  $v$  walking in a clockwise order of the edges. Let  $w_1$  be the first leaf of  $T_v$  found, and let  $w_2$  and  $w_3$  be the next two vertices visited in the traversal. Two cases arise:

**Case 1:**  $w_1, w_2, w_3$  makes a right turn. We define  $s$  as the first point hit by the ray apexed at  $w_2$  that shoots in the direction opposite to  $w_3$ .

We claim that  $w_1$  and  $s$  lie on the same edge of the boundary of  $S_v(C_v)$ . Otherwise, there would be a vertex  $u$  visible from  $w_2$  inside the wedge with apex  $w_2$  spanned by  $w_1$  and  $w_3$ . Note that the first edge of the path  $\pi(u, v)$  is the edge  $uw_2$ . Therefore,  $uw_2$  belongs to the shortest-path  $T_v$  contradicting the Eulerian order in which the vertices of this tree are visited as  $u$  should be visited before  $w_3$ . Thus,  $s$  and  $w_1$  lie on the same edge and  $s$  can be computed in  $O(1)$  time.

At this point, we construct the apexed triangle  $\Delta(w_2, w_1, s)$  apexed at  $w_2$  with apex function

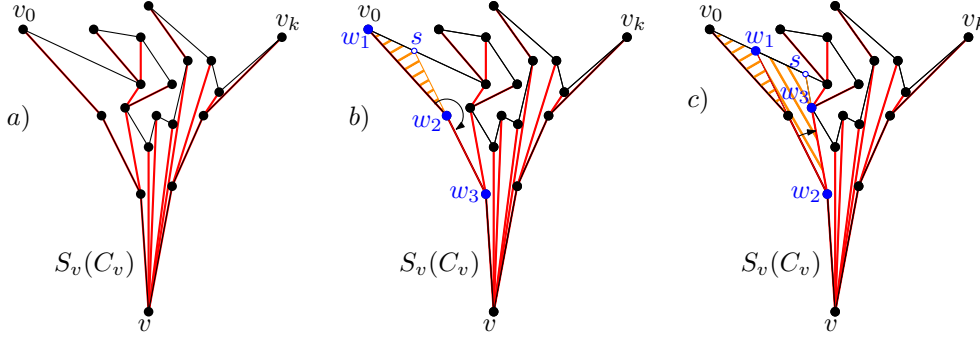
$$g(x) = \begin{cases} |xw_2| + |\pi(w_2, v)| & \text{if } x \in \Delta(w_2, w_1, s) \\ -\infty & \text{otherwise} \end{cases}$$

We modify tree  $T_v$  by removing the edge  $w_1w_2$  and replacing the edge  $w_3w_2$  by the edge  $w_3s$ ; see Figure 5.

**Case 2:**  $w_1, w_2, w_3$  makes a left turn and  $w_1$  and  $w_3$  are adjacent, then if  $w_1$  and  $w_3$  lie on the same edge of  $\partial P$ , we construct an apexed triangle  $\Delta(w_2, w_1, w_3)$  apexed at  $w_2$  with apex function

$$g(x) = \begin{cases} |xw_2| + |\pi(w_2, v)| & \text{if } x \in \Delta(w_2, w_1, w_3) \\ -\infty & \text{otherwise} \end{cases}$$

Otherwise, let  $s$  be the first point of the boundary of  $S_v(C_v)$  hit by the ray shooting from  $w_3$  in the direction opposite to  $w_2$ .



■ **Figure 5** The funnel  $S_v(C_v)$  and the shortest-path tree from  $v$  are depicted in (a) . The two cases of the algorithm described in Lemma 12 are shown in (b) and (c).

By the same argument as above, we can show that  $w_1$  and  $s$  lie on the same edge of the boundary of  $S_v(C_v)$  (and thus, we can compute  $s$  in  $O(1)$  time). We construct an apexed triangle  $\triangle(w_2, w_1, s)$  apexed at  $w_2$  with apex function

$$g(x) = \begin{cases} |xw_2| + |\pi(w_2, v)| & \text{if } x \in \triangle(w_2, w_1, s) \\ -\infty & \text{otherwise} \end{cases}.$$

424 We modify the tree  $T_v$  by removing the edge  $w_1w_2$  and adding the edge  $w_3s$ ; see Figure 5  
425 for an illustration.

426 ► **Lemma 12.** *The above procedure runs in  $O(|S_v(C_v)|)$  time and computes  $O(|S_v(C_v)|)$   
427 interior disjoint apexed triangles such that their union covers  $S_v(C_v)$ . Moreover, for each  
428 point  $x \in R(v)$ , there is an apexed triangle  $\triangle$  with apex function  $g(x)$  such that (1)  $x \in \triangle$   
429 and (2)  $g(x) = F_P(x)$ .*

430 **Proof.** The above procedure splits  $S_v(C_v)$  into apexed triangles, such that their apex function  
431 in each of them is defined as the geodesic distance to  $v$ . By Lemma 11, if  $x \in R(v)$ , then  
432  $x \in S_v(C_v)$ . Therefore, there is an apexed triangle  $\triangle$  with apex function  $g(x)$  such that  
433  $x \in \triangle$  and  $g(x) = |\pi(x, v)| = F_P(x)$ . Consequently, we obtain properties (1) and (2).

434 We now bound the running time of the algorithm. The shortest-path tree  $T_v$  from  $v$   
435 is computed in  $O(|S_v(C_v)|)$  time [10]. For each leaf of  $T_v$  we need a constant number of  
436 operations to determine in which of the cases we are in (and to treat it as well). Therefore, it  
437 suffices to bound the number of times these steps are performed. Note that a leaf is removed  
438 from the tree in each iteration. Since the number of leaves strictly decreases each time we  
439 are in Case 2, this step cannot happen more than  $O(|S_v(C_v)|)$  times. In Case 1 a new leaf  
440 is added if  $w_1$  and  $w_3$  do not lie on the same edge of  $\partial P$ . However, the number of leaves  
441 that can be added throughout is at most the number of edges of  $T_v$ . Note that the edges  
442 added by either Case 1 or 2 are chords of the polygon and hence do not generate further  
443 leaves. Because  $|T_v| = O(|S_v(C_v)|)$ , we conclude that both Case 1 and 2 are only executed  
444  $O(|S_v(C_v)|)$  times. ◀

## 445 6 Prune and search

446 With the tools introduced in the previous sections, we can proceed to give the prune and  
447 search algorithm to compute the geodesic center. The idea of the algorithm is to partition  $P$   
448 into  $O(1)$  cells, determine on which cell of  $P$  the center lies and recurse on that cell as a new  
449 subproblem with smaller complexity.

450 Naturally, we can discard all apexed triangles that do not intersect the new cell containing  
 451 the center. Using the properties of the cutting, we can show that both the complexity of the  
 452 cell containing the center, and the number of apexed triangles that intersect it decrease by  
 453 a constant fraction in each iteration of the algorithm. This process is then repeated until  
 454 either of the two objects has constant descriptive size.

455 Let  $\tau$  be the set all apexed triangles computed in previous sections. Lemmas 6 and 12  
 456 directly provide a bound on the complexity of  $\tau$ .

457 ► **Corollary 13.** *The set  $\tau$  consists of  $O(n)$  apexed triangles.*

458 Let  $\phi(x)$  be the upper envelope of the apex functions of every triangle in  $\tau$  (i.e.,  $\phi(x) =$   
 459  $\max\{g_i(x) : g_i(x) \in \tau\}$ ). The following result is a direct consequence of Lemmas 10 and 12,  
 460 and shows that the  $O(n)$  apexed triangles of  $\tau$  not only cover  $P$ , but their apex functions  
 461 suffice to reconstruct the function  $F_P(x)$ .

462 ► **Lemma 14.** *The functions  $\phi(x)$  and  $F_P(x)$  coincide in the domain of points of  $P$ , i.e., for*  
 463 *each  $p \in P$ ,  $\phi(p) = F_P(p)$ .*

464 Given a chord  $C$  of  $P$  a *half-polygon* of  $P$  is one of the two simple polygons in which  
 465  $C$  splits  $P$ . A *4-cell* of  $P$  is a simple polygon obtained as the intersection of at most four  
 466 half-polygons. Because a 4-cell is the intersection of geodesically convex sets, it is also  
 467 geodesically convex.

468 Let  $R$  be a 4-cell of  $P$  and let  $\tau_R$  be the set of apexed triangles of  $\tau$  that intersect  $R$ .  
 469 Let  $m_R = \max\{|R|, |\tau_R|\}$ . Recall that, by construction of the apexed triangles, for each  
 470 triangle of  $\tau_R$  at least one and at most two of its boundary segments is a chord of  $P$ . Let  $\mathcal{C}$   
 471 be the set containing all chords that belong to the boundary of a triangle of  $\tau_R$ . Therefore,  
 472  $|\tau_R| \leq |\mathcal{C}| \leq 2|\tau_R|$ .

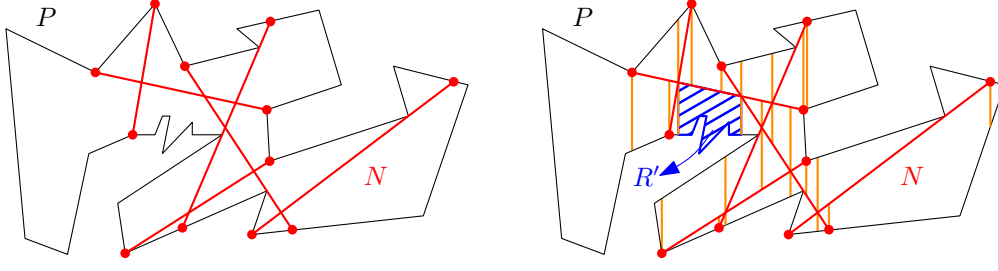
473 To construct an  $\varepsilon$ -net of  $\mathcal{C}$ , we need some definitions (for more information on  $\varepsilon$ -nets refer  
 474 to [17]). Let  $\varphi$  be the set of all open 4-cells of  $P$ . For each  $t \in \varphi$ , let  $\mathcal{C}_t = \{C \in \mathcal{C} : C \cap t \neq \emptyset\}$   
 475 be the set of chords of  $\mathcal{C}$  induced by  $t$ . Finally, let  $\varphi_{\mathcal{C}} = \{\mathcal{C}_t : t \in \varphi\}$  be the family of subsets  
 476 of  $\mathcal{C}$  induced by  $\varphi$ .

477 Let  $\varepsilon > 0$  (the exact value of  $\varepsilon$  will be specified later). Consider the range space  $(\mathcal{C}, \varphi_{\mathcal{C}})$   
 478 defined by  $\mathcal{C}$  and  $\varphi_{\mathcal{C}}$ . Because the VC-dimension of this range space is finite, we can compute  
 479 an  $\varepsilon$ -net  $N$  of  $(\mathcal{C}, \varphi_{\mathcal{C}})$  in  $O(n/\varepsilon) = O(n)$  time [17]. The size of  $N$  is  $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) = O(1)$  and  
 480 its main property is that any 4-cell that does not intersect a chord of  $N$  will intersect at  
 481 most  $\varepsilon|\mathcal{C}|$  chords of  $\mathcal{C}$ .

482 Observe that  $N$  partitions  $R$  into  $O(1)$  sub-polygons (not necessarily 4-cells). We further  
 483 refine this partition by performing a 4-cell decomposition. That is, we shoot vertical rays up  
 484 and down from each endpoint of  $N$ , and from the intersection point of any two segments  
 485 of  $N$ , see Figure 6. Overall, this partitions  $R$  into  $O(1)$  4-cells such that each either (i) is  
 486 a convex polygon contained in  $P$  of at most four vertices, or otherwise (ii) contains some  
 487 chain of  $\partial P$ . Since  $|N| = O(1)$ , the whole decomposition can be computed in  $O(m_R)$  time  
 488 (the intersections between segments of  $N$  are done in constant time, and for the ray shooting  
 489 operations we walk along the boundary of  $R$  once).

490 In order to determine which 4-cell contains the geodesic center of  $P$ , we extend each edge  
 491 of a 4-cell to a chord  $C$ . This can be done with two ray-shooting queries (each of which  
 492 takes  $O(m_R)$  time). We then use the chord-oracle from Pollack et al. [22, Section 3] to decide  
 493 which side of  $C$  contains  $c_P$ . The only requirement of this technique is that the function  
 494  $F_P(x)$  coincides with the upper envelope of the apex functions when restricted to  $C$ . Which  
 495 is true by Lemma 14 and from the fact that  $\tau_R$  consists of all the apexed triangles of  $\tau$  that  
 496 intersect  $R$ .





■ **Figure 6** The  $\epsilon$ -net  $N$  splits  $P$  into  $O(1)$  sub-polygons that are further refined into a 4-cell decomposition using  $O(1)$  ray-shooting queries from the vertices of the arrangement defined by  $N$ .

Because the chord-oracle described by Pollack et al. [22, Section 3] runs in linear time on the number of functions defined on  $C$ , we can decide in total  $O(m_R)$  time on which side of  $C$  the geodesic center of  $P$  lies. Since our decomposition into 4-cells has constant complexity, we need to perform  $O(1)$  calls to the oracle before determining the 4-cell  $R'$  that contains the geodesic center of  $P$ .

The chord-oracle computes the minimum of  $F_P(x)$  restricted to the chord before determining the side containing the minimum. In particular, if  $c_P$  lies on any chord bounding  $R'$ , then the chord-oracle will find it. Therefore, we can assume that  $c_P$  lies in the interior of  $R'$ . Moreover, since  $N$  is a  $\epsilon$ -net, we know that at most  $\epsilon|C|$  chords of  $C$  will intersect  $R'$ .

Using a similar argument, we can show that the complexity of  $R'$  also decreases: since  $|C| \leq 2|\tau_R| \leq 2m_R$ , we guarantee that at most  $2\epsilon m_R$  apexed triangles intersect  $R'$ . Moreover, each vertex of  $R'$  is in at least one apexed triangle of  $\tau_R$  by Lemma 14, and by construction, each apexed triangle can cover at most three vertices. Thus, by the pigeonhole principle we conclude that  $R'$  can have at most  $6\epsilon m_R$  vertices. Thus, if we choose  $\epsilon = 1/12$ , we guarantee that both the size of the 4-cell  $R'$  and the number of apexed triangles in  $\tau_{R'}$  are at most  $m_R/2$ .

In order to proceed with the algorithm on  $R'$  recursively, we need to compute the set  $\tau_{R'}$  with the at most  $\epsilon|C|$  apexed triangles of  $\tau_R$  that intersect  $R'$  (i.e., prune the apexed triangles that do not intersect with  $R'$ ). For each apexed triangle  $\Delta \in \tau_R$ , we can determine in constant time if it intersects  $R'$  (either one of the endpoints is in  $R' \cap \partial P$  or the two boundaries have non-empty intersection in the interior of  $P$ ). Overall, we need  $O(m_R)$  time to compute the at most  $\epsilon|C|$  triangles of  $\tau_R$  that intersect  $R'$ .

By recursing on  $R'$ , we guarantee that after  $O(\log m_R)$  iterations, we reduce the size of either  $\tau_R$  or  $R'$  to constant. In the former case, the minimum of  $F_P(x)$  can be found by explicitly constructing function  $\phi$  in  $O(1)$  time. In the latter case, we triangulate  $R'$  and apply the chord-oracle to determine which triangle will contain  $c_P$ . The details needed to find the minimum of  $\phi(x)$  inside this triangle are giving the next section.

► **Lemma 15.** *In  $O(n)$  time we can find either the geodesic center of  $P$  or a triangle containing the geodesic center.*

## 7 Solving the problem restricted to a triangle

In order to complete the algorithm it remains to show how to find the geodesic center of  $P$  for the case in which  $R'$  is a triangle. If this triangle is in the interior of  $P$ , it may happen that several apexed triangles of  $\tau$  fully contain  $R'$ . Thus, the pruning technique used in the previous section cannot be further applied. We solve this case with a different approach.



Recall that  $\phi(x)$  denotes the upper envelope of the apex functions of the triangles in  $\tau$ , and the geodesic center is the point that minimizes  $\phi$ . The key observation is that, as it happened with chords, the function  $\phi(x)$  restricted to  $R'$  is convex.

Let  $\triangle_1, \triangle_2, \dots, \triangle_m$  be the set of  $m = O(n)$  apexed triangles of  $\tau$  that intersect  $R'$ . Let  $g_i(x)$  be the apex function of  $\triangle_i$  such that

$$g_i(x) = \begin{cases} |xa_i| + \kappa_i & \text{if } x \in \triangle_i \\ -\infty & \text{otherwise} \end{cases},$$

where  $a_i$  and  $w_i$  are the apex and the definer of  $\triangle_i$ , respectively, and  $\kappa_i = |\pi(a_i, w_i)|$  is a constant.

By Lemma 14,  $\phi(x) = F_P(x)$ . Therefore, the problem of finding the center is equivalent to the following optimization problem in  $\mathbb{R}^3$ :

**(P1).** Find a point  $(x, r) \in \mathbb{R}^3$  minimizing  $r$  subject to  $x \in R'$  and

$$g_i(x) \leq r, \text{ for } 1 \leq i \leq m.$$

Thus, we need only to find the solution to (P1) to find the geodesic center of  $P$ . We use some remarks described by Megiddo in order to simplify the description of (P1) [18].

To simplify the formulas, we square the equation  $|xa_i| \leq r - \kappa_i$ :

$$\|x\|^2 - 2x \cdot a_i + \|a_i\|^2 = |xa_i|^2 \leq (r - \kappa_i)^2 = r^2 - 2r\kappa_i + \kappa_i^2.$$

And finally for each  $1 \leq i \leq m$ , we define the function  $h_i(x, r)$  as follows:

$$h_i(x, r) = \begin{cases} \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2 & \text{if } x \in \triangle_i \\ -\infty & \text{otherwise} \end{cases}$$

Therefore, our optimization problem can be reformulated as:

**(P2).** Find a point  $(x, r) \in \mathbb{R}^3$  such that  $r$  is minimized subject to  $x \in R'$  and

$$h_i(x, r) \leq 0 \text{ and } r > \max\{\kappa_i\}, \text{ for } 1 \leq i \leq m.$$

Let  $h'_i(x, r) = \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2$  be a function defined in the entire plane and let (P2') be an optimization problem analogous to (P2) where every instance of  $h_i(x, r)$  is replaced by  $h'_i(x, r)$ . The optimization (P2') was studied by Megiddo in [18]. We provide some of the intuition used by Megiddo to solve this problem.

Although the functions  $h'_i(x, r)$  are not linear inside  $\triangle_i$ , they all have the same non-linear terms. Therefore, for  $i \neq j$ , we get that  $h'_i(x, r) = h'_j(x, r)$  defines a *separating plane*

$$\gamma_{i,j} = \{(x, r) \in \mathbb{R}^3 : 2(\kappa_i - \kappa_j)r - 2(a_i - a_j) \cdot x + \|a_i\|^2 - \|a_j\|^2 - \kappa_i^2 + \kappa_j^2 = 0\}$$

As noted by Megiddo [18], this separating plane has the following property: If the solution  $(x, r)$  to (P2') is known to lie to one side of  $\gamma_{i,j}$ , then we know that one of the constraints is redundant.

Thus, to solve (P2') it sufficed to have a *side-decision oracle* to determine on which side of a plane  $\gamma_{i,j}$  the solution lies. Megiddo showed how to implement this oracle in a way that the running time is proportional to the number of constraints [18].

Once we have such an oracle, Megiddo's problem can be solved using a prune and search similar to that introduced in Section 6: pair the functions arbitrarily, and consider the set of  $m/2$  separating planes defined by these pairs. For some constant  $r$ , compute a  $1/r$ -cutting in  $\mathbb{R}^3$  of the separating planes. A  $1/r$ -cutting is a partition of the plane into  $O(r^2)$  convex

regions each of which is of constant size and intersects at most  $m/2r$  separating planes. A cutting of planes can be computed in linear time in  $\mathbb{R}^3$  for any  $r = O(1)$  [16]. After computing the cutting, determine in which of the regions the minimum lies by performing  $O(1)$  calls to the side-decision oracle. Because at least  $(r-1)m/2r$  separating planes do not intersect this constant size region, for each of them we can discard one of the constraints as it becomes redundant. Repeating this algorithm recursively we obtain a linear running time.

To solve (P2) we follow a similar approach, but our set of separating planes needs to be extended in order to handle apex functions as they are only defined in the same way as in (P2') in a triangular domain. Note that the vertices of each apexed triangle that intersect  $R'$  have their endpoints either outside of  $R'$  or on its boundary.

## 7.1 Optimization problem in a convex domain

In this section we describe our algorithm to solve the optimization problem (P2). To this end, we pair the apexed triangles arbitrarily to obtain  $m/2$  pairs. By identifying the plane where  $P$  lies with the plane  $Z_0 = \{(x, y, z) : z = 0\}$ , we can embed each apexed triangle in  $\mathbb{R}^3$ . A *plane-set* is a set consisting of at most five planes in  $\mathbb{R}^3$ . For each pair of apexed triangles  $(\Delta_i, \Delta_j)$  we define a plane-set as follows: For each chord bounding either  $\Delta_i$  or  $\Delta_j$ , consider the line extending this chord and the vertical extrusion of this line in  $\mathbb{R}^3$ , i.e., the plane containing this chord orthogonal to  $Z_0$ . Moreover, consider the separating plane  $\gamma_{i,j}$ . The set containing these planes is the plane-set of the pair  $(\Delta_i, \Delta_j)$ .

Let  $\Gamma$  be the union of all the plane-sets defined by the  $m/2$  pairs of apexed triangles. Thus,  $\Gamma$  is a set that consists of  $O(m)$  planes. Compute an  $1/r$ -cutting of  $\Gamma$  in  $O(m)$  time for some constant  $r$  to be specified later. Because  $r$  is constant, this  $1/r$ -cutting splits the space into  $O(1)$  convex regions, each bounded by a constant number of planes [16]. Using a side-decision algorithm (to be specified later), we can determine the region  $Q$  of the cutting that contains the solution to (P2). Because  $Q$  is the region of a  $1/r$ -cutting of  $\Gamma$ , we know that at most  $|\Gamma|/r$  planes of  $\Gamma$  intersect  $Q$ . In particular, at most  $|\Gamma|/r$  plane-sets intersect  $Q$  and hence, at least  $(r-1)|\Gamma|/r$  plane-sets do not intersect  $Q$ .

Let  $(\Delta_i, \Delta_j)$  be a pair such that its plane-set does not intersect  $Q$ . Let  $Q'$  be the projection of  $Q$  on the plane  $Z_0$ . Because the plane-set of this pair does not intersect  $Q$ , we know that  $Q'$  intersects neither the boundary of  $\Delta_i$  nor that of  $\Delta_j$ . Two cases arise:

**Case 1.** If either  $\Delta_i$  or  $\Delta_j$  does not intersect  $Q'$ , then we know that their apex function is redundant and we can drop the constraint associated with this apexed triangle.

**Case 2.** If  $Q' \subset \Delta_i \cap \Delta_j$ , then we need to decide which constrain to drop. To this end, we consider the separating plane  $\gamma_{i,j}$ . Notice that inside the vertical extrusion of  $\Delta_i \cap \Delta_j$  (and hence in  $Q$ ), the plane  $\gamma_{i,j}$  has the property that if we know its side containing the solution, then one of the constraints can be dropped. Since  $\gamma_{i,j}$  does not intersect  $Q$  as  $\gamma_{i,j}$  belongs to the plane-set of  $(\Delta_i, \Delta_j)$ , we can decide which side of  $\gamma_{i,j}$  contains the solution to (P2) and drop one of the constraints.

Regardless of the case if the plane-set of a pair  $(\Delta_i, \Delta_j)$  does not intersect  $Q$ , then we can drop one of its constraints. Since at least  $(r-1)|\Gamma|/r$  plane-sets do not intersect  $Q$ , we can drop at least  $(r-1)|\Gamma|/r$  constraints. Because  $|\Gamma| \geq m/2$  as each plane-set contains at least one plane, by choosing  $r = 2$ , we are able to drop at least  $|\Gamma|/2 \geq m/4$  constraints. Consequently, after  $O(m)$  time, we are able to drop  $m/4$  apexed triangles. By repeating this process recursively, we end up with a constant size problem in which we can compute the upper envelope of the functions explicitly and find the solution to (P2) using exhaustive search. Thus, the running time of this algorithm is bounded by the recurrence  $T(m) = T(3m/4) + O(m)$

601 which solves to  $O(m)$ . Because  $m = O(n)$ , we can find the solution to (P2) in  $O(n)$  time.

602 The last detail is the implementation of the side-decision algorithm. Given a plane  $\gamma$ ,  
 603 we want to decide on which side lies the solution to (P2). To this end, we solve (P2)  
 604 restricted to  $\gamma$ , i.e., with the additional constraint of  $(x, r) \in \gamma$ . This approach was used  
 605 by Megiddo [18], the idea is to recurse by reducing the dimension of the problem. Another  
 606 approach is to use a slight modification of the chord-oracle described by Pollack et al. [22,  
 607 Section 3].

608 Once the solution to (P2) restricted to  $\gamma$  is known, we can follow the same idea used by  
 609 Megiddo [18] to find the side of  $\gamma$  containing the global solution to (P2). Intuitively, we find  
 610 the apex functions that define the minimum restricted to  $\gamma$ . Since  $\phi(x) = F_P(x)$  is locally  
 611 defined by this functions, we can decide on which side the minimum lies using convexity. We  
 612 obtain the following result.

613 ► **Lemma 16.** *Let  $R'$  be a convex trapezoid contained in  $P$  such that  $R'$  contains the geodesic*  
 614 *center of  $P$ . Given the set of all apexed triangles of  $\tau$  that intersect  $R'$ , we can compute the*  
 615 *geodesic center of  $P$  in  $O(n)$  time.*

616 The following theorem summarizes the result presented in this paper.

617 ► **Theorem 17.** *We can compute the geodesic center of any simple polygon  $P$  of  $n$  vertices*  
 618 *in  $O(n)$  time.*

## 619 — References —

- 620 1 B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorith-*  
 621 *mica*, 4(1-4):109–140, 1989.
- 622 2 B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Dis-*  
 623 *crete & Computational Geometry*, 9(1):217–255, 1993.
- 624 3 T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical  
 625 Report SOCS-85.32, McGill University, 1985.
- 626 4 S. W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains.  
 627 *Discrete & Computational Geometry*, 50(2):306–329, 2013.
- 628 5 S. W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal  
 629 domain. In *Proceedings of CCCG*, 2014.
- 630 6 S. W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the  $L_1$  geodesic diameter  
 631 and center of a simple polygon in linear time. In *Proceedings of LATIN*, pages 120–131,  
 632 2014.
- 633 7 B. Chazelle. A theorem on polygon cutting with applications. In *Proceedings of FOCS*,  
 634 pages 339–349, 1982.
- 635 8 H. Djidjev, A. Lingas, and J.-R. Sack. An  $O(n \log n)$  algorithm for computing the link  
 636 center of a simple polygon. *Discrete & Computational Geometry*, 8:131–152, 1992.
- 637 9 H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with  
 638 degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104,  
 639 1990.
- 640 10 L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms  
 641 for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*,  
 642 2(1-4):209–233, 1987.
- 643 11 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In  
 644 *Proceedings of STOC*, pages 50–63. ACM, 1987.
- 645 12 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM*  
 646 *Journal on Computing*, 13(2):338–355, 1984.

- 647 **13** J. Hershberger and S. Suri. Matrix searching with the shortest path metric. In *Proceedings*  
648 *of STOC*, pages 485–494. ACM, 1993.
- 649 **14** Y. Ke. An efficient algorithm for link-distance problems. In *Proceedings of SoCG*, pages  
650 69–78, 1989.
- 651 **15** D.-T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear  
652 barriers. *Networks*, 14(3):393–410, 1984.
- 653 **16** J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proceedings*  
654 *of STOC*, pages 505–511. ACM, 1991.
- 655 **17** J. Matoušek. Construction of epsilon nets. In *Proceedings of SoCG*, pages 1–10, New York,  
656 1989. ACM.
- 657 **18** N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–  
658 610, 1989.
- 659 **19** J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and  
660 J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.
- 661 **20** B. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In  
662 *Proceedings of CG*, pages 203–215, 1991.
- 663 **21** B. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a  
664 rectilinear polygon. *Computational Geometry: Theory and Applications*, 6:169–194, 1996.
- 665 **22** R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon.  
666 *Discrete & Computational Geometry*, 4(1):611–626, 1989.
- 667 **23** S. Suri. *Minimum Link Paths in Polygons and Related Problems*. PhD thesis, Johns Hopkins  
668 Univ., 1987.
- 669 **24** S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer*  
670 *and System Sciences*, 39(2):220–235, 1989.