

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

Hee-Kap Ahn^{*3}, Luis Barba^{1,2}, Prosenjit Bose¹, Jean-Lou de Carufel¹, Matias Korman^{4,5}, and Eunjin Oh³

¹ School of Computer Science, Carleton University, Ottawa, Canada.

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

² Département d'Informatique, Université Libre de Bruxelles, Brussels, Belgium.

lbarbafl@ulb.ac.be

³ Department of Computer Science and Engineering, POSTECH,

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

heekap@postech.ac.kr

⁴ National Institute of Informatics (NII), Tokyo, Japan.

korman@nii.ac.jp

⁵ JST, ERATO, Kawarabayashi Large Graph Project.

Abstract

Let P be a simple polygon with n vertices. Given two points in P , 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 to compute the geodesic center of P . Since then, a longstanding question, posed also by Mitchell [Handbook of Computational Geometry, 2000], has been whether this running time can be improved. 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 . Notice that if the straight-line segment connecting x with y is contained in P , then $\pi(x, y)$ is a straight-line segment. Otherwise, $\pi(x, y)$ is a polygonal chain containing only reflex vertices of P other than its endpoints. (For more information on geodesic paths refer to [19]).

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 assume this *general position* 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 80s 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 of whose applications was 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]).

Several other variations of these two problems have been considered. Nowadays there exist algorithms for computing the center and diameter 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

To guide the reader, we provide a rough sketch of our algorithm.

The $O(n \log n)$ -time algorithm proposed by Pollack et al. [22] could be summarized as follows: Given a chord C of P that splits the polygon into two sub-polygons, they describe a linear time *chord-oracle* that decides which sub-polygon contains c_P . Using this oracle together with the set of chords of a triangulation of P , they narrow the search of P to a triangle in which optimization techniques can be used to find c_P . 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. We overcome this issue using the following approach.

Our algorithm computes a set of $O(n)$ functions of constant description, 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 $\phi(x)$.

Because $\phi(x)$ is defined using $O(n)$ functions having triangular domains, we are able to use a prune and search approach using cuttings as follows: We first find a suitable set of $O(n)$ chords of P that splits the polygon into convex regions having constant size. Our objective is then to find the convex region that contains c_P . To this end, we use a cutting of these chords. This cutting has constant complexity and splits P into $O(1)$ cells. We then find the cell that contains c_P and recurse on this cell as a new subproblem having smaller complexity.

To decrease the complexity of the problem, we consider only the functions defined in this cell in the next iteration. Using the properties of the cutting, we show that the size of the subproblem decreases by a constant fraction which leads to a linear running time. This algorithm has however two stopping conditions, one is to reach a subproblem of constant size, and a second one is to find a convex trapezoid containing c_P . In the latter case, we are not able to proceed with the prune and search. Nevertheless, by restricting the search space to a convex object, we show that $\phi(x)$ is a convex function in this domain and hence, we are able to use optimization techniques using cuttings in \mathbb{R}^3 to find the geodesic center in linear time.

As mentioned above, the main idea of the algorithm is to compute a set of triangles whose union covers P such that: (1) each triangle has a distance function defined in it and (2) their upper envelope $\phi(x)$ coincides with $F_P(x)$. More formally, for each point $x \in P$, there is one triangle containing x and a function g defined in this triangle, such that $g(x) = F_P(x)$. Intuitively, we compute a set of functions that “shatter” $F_P(x)$ into small pieces. 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 compute the farthest neighbor of each vertex of P . In this way, we partition the boundary of P , denoted by ∂P , into connected edge disjoint chains, each grouping the vertices of P that share the same farthest neighbor. We say that a vertex is *marked* if it is represented by a chain in this partition of the boundary (not all vertices are marked). Further, this partition induces *transition edges* whose endpoints have different farthest neighbors.

In Section 4, we consider each transition edge ab of ∂P independently and compute its *hourglass*. The hourglass H_{ab} of ab is the geodesic convex hull of the segment ab and the chain contained in ∂P connecting $f(a)$ and $f(b)$; recall that $f(a)$ and $f(b)$ denote the farthest neighbors of a and b , respectively. 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 each of these hourglasses in linear time.

In Section 5 we show how to compute the triangles and their respective functions. 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. (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 $O(n)$ triangles and we show that this can be done in $O(n)$ time.

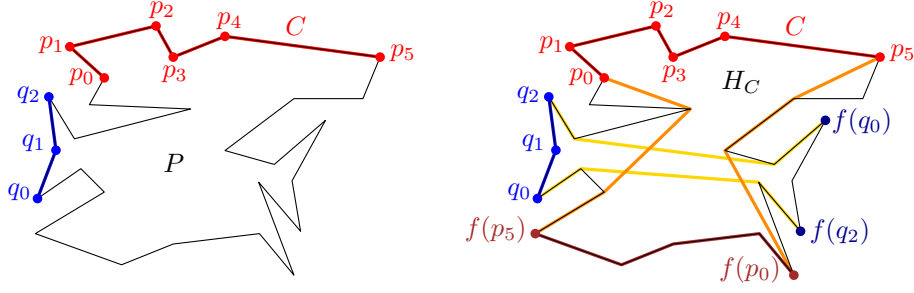
2 Hourglasses and Funnels

In this section, we introduce the main tools that are going to be used by the algorithm. Some of the result 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 ∂P , let $\partial P(x, y)$ be the polygonal chain that starts at x and follows the boundary of P clockwise until reaching y .

Let $C = (p_0, p_1, \dots, p_k)$ be a polygonal chain contained in ∂P sorted in clockwise order. 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 .



■ **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 ∂P .

131 We say that the hourglass H_C is *open* if its walls are vertex disjoint. We say C is a
 132 *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
 133 particular, if an edge ab of ∂P is a transition chain, we say that it is a *transition edge*.

134 ► **Lemma 1.** [Rephrase of Lemma 3.1.3 of [2]] If C is a transition chain of ∂P , then the
 135 hourglass H_C is an open hourglass.

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

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

141 ► **Lemma 2.** Let C_1, C_2, C_3 be three edge disjoint transition chains of ∂P that appear in this
 142 order when traversing clockwise the boundary of P . Then, the bottom chains of H_{C_1}, H_{C_2} and
 143 H_{C_3} are also edge disjoint and appear in this order when traversing clockwise the boundary
 144 of P .

145 Let γ be a geodesic path joining two points on the boundary of P . We say that γ *separates*
 146 two points x_1 and x_2 of ∂P if the points of $X = \{x_1, x_2\}$ and the endpoints of γ alternate
 147 along the boundary of P . Given a hourglass H , we say that γ *separates* H if it separates the
 148 points of its top chain from those of its bottom chain.

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

153 **Proof.** Aronov et al. showed that there exist four vertices v_1, \dots, v_4 of P and geodesic paths
 154 $\pi(v_1, v_2), \pi(v_2, v_3), \pi(v_3, v_4)$ such that for any point $x \in \partial P$, one of these paths separates x
 155 from $f(x)$ [2, Lemma 2.7.6]. Moreover, they show how to compute this set in $O(n)$ time.

156 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
 157 at most four connected components. If a chain C_i is completely contained in one of this
 158 components, then one path of Γ separates the top and bottom chain of H_{C_i} . Otherwise,
 159 some vertex v_j is an interior vertex of C_i . However, because the chains C_1, \dots, C_r are edge
 160 disjoint, there are at most four chains in this situation. For each chain C_i containing a vertex
 161 v_j , we add the geodesic path connecting the endpoints of C_i to Γ . Therefore, Γ consists of
 162 $O(1)$ geodesic paths and each hourglass H_{C_i} has its top and bottom chain separated by some
 163 path of Γ . Since only $O(1)$ paths are computed, this can be done in linear time. ◀

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

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

169 **Proof.** Assume for a contradiction that there is a chord st that appears in three hourglasses
 170 H_{C_i}, H_{C_j} and H_{C_k} such that $1 \leq i < j < k \leq r$. Note that chords can only appear on the
 171 walls of these hourglasses. Because the hourglasses are open, st must be an edge on exactly
 172 one wall of each of these hourglasses.

173 Assume that s is visited before t when going from the top to the bottom chain along
 174 these walls. Let $\pi(s_i, t_i)$ be the wall of S_i that contains st such that s_i and t_i lie in the top
 175 and bottom chains of H_{C_i} , respectively. Define $\pi(s_k, t_k)$ analogously.

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

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

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

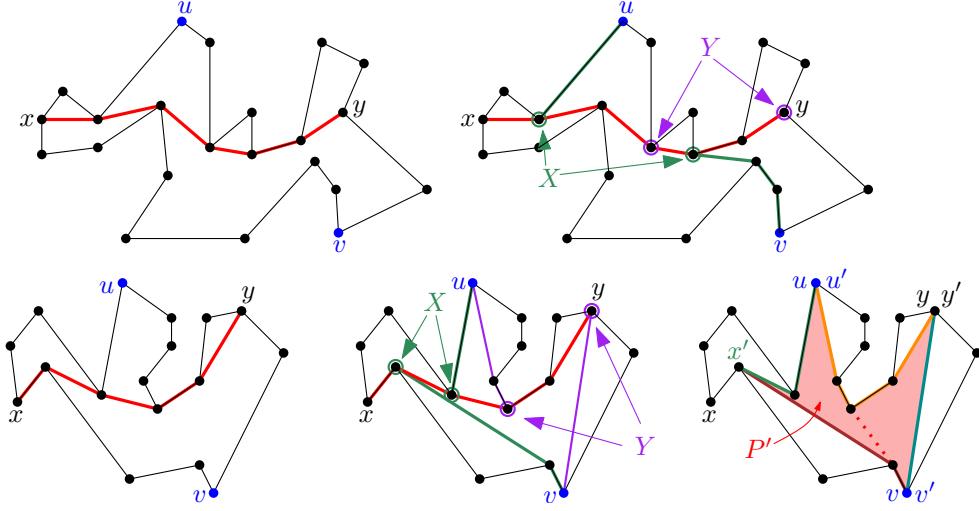
195 **Case 1.** If there is a vertex $x^* \in X$ lying after a vertex $y^* \in Y$ along $\pi(x, y)$, then the
 196 path $\pi(u, v)$ contains the path $\pi(y^*, x^*)$. In this case, the path $\pi(u, v)$ is the concatenation of
 197 the paths $\pi(u, y^*)$, $\pi(y^*, x^*)$, and $\pi(x^*, v)$ and that the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

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

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

205 Let P' be the polygon bounded by the geodesic paths $\pi(x', u')$, $\pi(u', y')$, $\pi(y', v')$ and
 206 $\pi(v', x')$. Because the vertices of X appear before those of Y along $\pi(x, y)$, P' is a simple
 207 polygon; see Figure 2 (bottom).

208 Note that the path $\pi(u, y)$ is the union of three paths $\pi(u, u')$, $\pi(u', v')$ and $\pi(v', v)$.
 209 Because $\pi(u, u')$ and $\pi(v', v)$ can be computed in time proportional to its length, it suffices
 210 to compute $\pi(u', v')$ in $O(|\pi(u', v')|)$ time.



■ **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 $\pi(v', x')$.

211 Note that P' is a simple polygon with only four convex vertices x', u', y' and v' connected
 212 by chains of reflex vertices. Regardless of the case, the shortest path from x' to y' can have
 213 at most one diagonal edge connecting distinct reflex chains of P' . Since the rest of the points
 214 in $\pi(u', v')$ lie on the boundary of P' and from the fact that each edge of P' is an edge of
 215 $T_x \cup T_y$, we conclude all edges of $\pi(u, v)$, except perhaps one, belong to $T_x \cup T_y$.

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

222 Since we do not know if the tangent lies between the paths $\pi(u', x')$ and $\pi(v', y')$, we
 223 process the chains $\pi(u', y')$ and $\pi(v', x')$ in parallel and stop when finding the desired tangent.
 224 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 ∂P , it holds that*

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

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

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

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

232 2.2 Funnels

233 Let $C = (p_0, \dots, p_k)$ be a chain of the boundary of P and let v be a vertex of P not in C .
 234 The *funnel* of v to C , denoted by $S_v(C)$, is the simple polygon bounded by C , $\pi(p_k, v)$ and
 235 $\pi(v, p_0)$; see Figure 3 (a). Note that the paths $\pi(v, p_k)$ and $\pi(v, p_0)$ may coincide for a while
 236 before splitting into disjoint chains. See Lee and Preparata [15] or Guibas et al. [10] for more
 237 details on funnels.

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

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

245 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).

247 ► **Lemma 7.** *Let v be a vertex of P and let C be a transition chain such that C contains*
 248 *$R(v) \cap \partial P$ and v is not contained in C . Then, $R(v)$ is contained in the funnel $S_v(C)$*

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

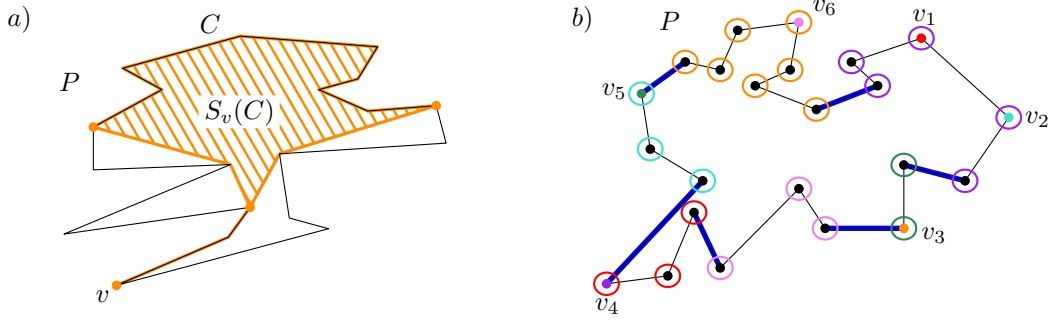
252 Let α (resp. β) be the bisector of v and $f(a)$ (resp. $f(b)$). Let h_a (resp. h_b) be the set of
 253 points of P that are farther from v than from $f(a)$ (resp. $f(b)$). Note that α is the boundary
 254 of h_a while β bounds h_b .

255 By definition, we know that $R(v) \subseteq h_a \cap h_b$. Therefore, it suffices to show that $h_a \cap h_b \subset S_v(C)$.
 256 Assume for a contradiction that there is a point of $h_a \cap h_b$ lying outside of $S_v(C)$.
 257 By continuity, the boundaries of $h_a \cap h_b$ and $S_v(C)$ intersect. Because $a \notin h_a$ and $b \notin h_b$,
 258 both α and β have an endpoint on the edge ab . Since the boundaries of $h_a \cap h_b$ and $S_v(C)$
 259 intersect, we infer that $\beta \cap \pi(v, b) \neq \emptyset$ or $\alpha \cap \pi(v, a) \neq \emptyset$. Without loss of generality, assume
 260 that there is a point $w \in \beta \cap \pi(v, b)$, the 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 ∂P only at its endpoints, we get that

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

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



■ **Figure 3** a) The funnel $S_v(C)$ of a vertex v and a chain C contained in ∂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.

263 3 Decomposing the boundary

264 In this section, we compute the farthest neighbor of each vertex of P . Note that the farthest
 265 neighbor of each vertex of P is always a convex vertex of P [3] and is unique by our general
 266 position assumption.

267 Using a result from Hershberger and Suri [13], in $O(n)$ time we can compute the farthest
 268 neighbor of each vertex of P . We then mark the vertices of P that are farthest neighbors
 269 of at least one vertex of P . Let M denote the set of marked vertices of P which can be
 270 computed in $O(n)$ time. In other words, M contains all vertices of P whose Voronoi region
 271 contains at least one vertex of P .

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

275 Let a and b be the endpoints of a transition edge of ∂P such that a appears before b in
 276 the clockwise order along ∂P . Because ab is a transition edge, we know that $f(a) \neq f(b)$.
 277 Recall that we have computed $f(a)$ and $f(b)$ in the previous step and note that $f(a)$ appears
 278 also before $f(b)$ along this clockwise order. For every vertex v that lies between $f(a)$ and $f(b)$
 279 in the bottom chain of H_{ab} , we know that there cannot be vertex u of P such that $f(u) = v$.
 280 As proved by Aronov et al. [2, Corollary 2.7.4], if there is a point x on ∂P whose farthest
 281 neighbor is v , then x must lie on the open segment (a, b) . In other words, the Voronoi region
 282 $R(v)$ restricted to ∂P is contained in (a, b) .

283 4 Building hourglasses

284 Let E be the set of transition edges of ∂P . Given a transition edge $ab \in E$, we say that H_{ab}
 285 is a *transition hourglass*. In order to construct the triangle cover of P , we need to construct
 286 the transition hourglass of each transition edge of E .

287 By Lemma 6, we know that $\sum_{ab \in E} |H_{ab}| = O(n)$. Therefore, an output sensitive algorithm
 288 would suffice for this task. In this section, we present an algorithm that computes each
 289 transition hourglass of P in $O(n)$ time.

290 By Lemma 3 we can compute a set of $O(1)$ separating paths such that for each transition
 291 edge ab , the transition hourglass H_{ab} is separated by some path in this set.

292 Let γ be a separating path whose endpoints are x and y . Note that γ separates the
 293 boundary of P into two chains S and S' such that $S \cup S' = \partial P$. Let \mathcal{H}_S be the set of each

transition hourglass separated by γ whose transition edge is contained in S . Note that \mathcal{H}_S can be constructed in $O(n)$ time. We claim that we can compute each transition hourglass of \mathcal{H}_S in $O(n)$ time. Note that the wall of each of these hourglasses consists of a (geodesic) path that connects a point in S with a point in S' .

To compute these walls, we start by computing the shortest-path trees T_x and T_y of x and y , respectively, in $O(n)$ time [10]. In addition, we preprocess these trees in linear time to support LCA queries [12]. Recall that there are $O(n)$ edges in total in both T_x and T_y .

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}_S . Because LCA queries can be answered in $O(1)$ time in T_x and T_y [12], Lemma 5 allows us to compute this path in $O(|\pi(u, v)|)$ time. Therefore, we can compute all hourglasses of \mathcal{H}_S in $O(\sum_{H \in \mathcal{H}_S} |H| + n)$ time. Which amounts to $O(n)$ by Lemma 6. Because there are only $O(1)$ separating paths by Lemma 3, we obtain the following result.

► **Lemma 8.** *If E is the set of transition edges of P , then we can construct the transition hourglass of each edge E in total $O(n)$ 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 Δ , such that (1) a is a vertex of P , (2) b and c are points on the boundary of P , and (3) there is a vertex w of P , called the *definer* of Δ , 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. Then, 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

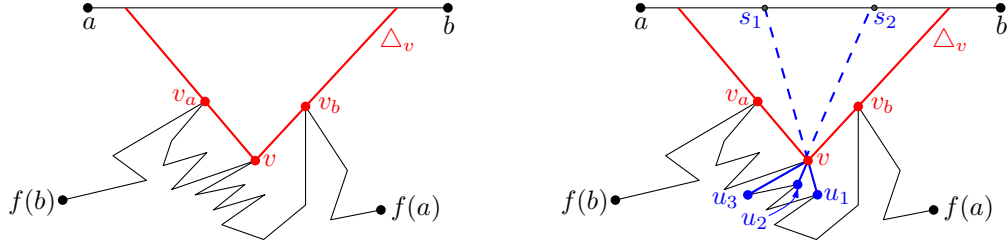
Let ab be a transition edge of P such that b is the clockwise neighbor of a along ∂P . Let B_{ab} denote the bottom chain of H_{ab} . As noticed above, a point on ∂P can be farthest from a vertex in B_{ab} only if it lies in the open segment ab . Formally, if v is a vertex of B_{ab} such that $R(v) \neq \emptyset$, then $R(v) \cap \partial P \subset ab$. We claim that not only this Voronoi region is inside H_{ab} when restricted to the boundary of P , but that $R(v) \subset H_{ab}$.

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”.

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



■ **Figure 4** (left) A vertex v visible from the segment ab lying on the bottom chain of H_{ab} , and the triangle Δ_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 Δ_v is split into apexed triangles by the rays going from u_1 and u_2 to v .

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

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

We further split Δ_v into a series of triangles with apex at v as follows: Let u be a child 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 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 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 T_a , and v_avu is a right turn.

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

$$f_i(x) = |xv| + \max_{j>i} \{c(v), |vu_j| + d_l(u_j)\}.$$

In a symmetric manner, we define a set of apexed triangles induced by the type (3) children 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 Δ_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 Δ_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 Δ_i . Intuitively, Δ_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. Note that by construction, there is a triangle $\Delta \in \mathcal{A}_{ab}$ apexed at v with definer w that contains p . Recall that by construction, the apex function $g(x)$ of Δ 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 by considering the apex functions of the apexed triangle in \mathcal{A}_{ab} , we do not lose any information inside any region $R(v)$ of any vertex $v \in B_{ab}$.

Following the same intuition, in the next section we construct a set of apexed triangles, and their apex functions, encoding 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 ∂P such that v is its farthest neighbor. 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 ∂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

396 $v = f(u_0) = f(u_{k-1})$, we know that v is a vertex of both $H_{u_0 u_1}$ and $H_{u_{k-1} u_k}$. Thus, since
 397 $\pi(v, u_0) \subset H_{u_0 u_1}$ while $\pi(v, u_k) \subset H_{u_{k-1} u_k}$, we can compute both $\pi(v, u_0)$ and $\pi(v, u_k)$
 398 in $O(|H_{u_0 u_1}| + |H_{u_{k-1} u_k}|)$ time. Consequently, the funnel $S_v(C_v)$ can be constructed in
 399 $O(k + |H_{u_0 u_1}| + |H_{u_{k-1} u_k}|)$.

400 Because a vertex on ∂P has a unique farthest neighbor by our general position assumption,
 401 and since the total sum of the complexities of the transition hourglasses is $O(n)$ by Lemma 6,
 402 we can compute the funnel of each vertex of M in total $O(n)$ time.

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

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

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

404 **Proof.** Because $f(u_0) \neq f(u_k)$, we know that C_v is a transition chain. Moreover, C_v contains
 405 $R(v) \cap \partial P$ by definition. Therefore, by Lemma 7, we know that $R(v) \subset S_v(C_v)$. Since
 406 $v = f(x)$, we know that $x \in R(v)$ and hence that $x \in S_v(C_v)$. ◀

407 Given a funnel $S_v(C_v)$, we would like to split it into $O(|S_v(C_v)|)$ apexed triangles that
 408 encode the distance function from v . To this end, we compute the shortest-path tree T_v of v
 409 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
 410 that for each node u of this tree we have stored the geodesic distance $|\pi(u, v)|$.

411 Let w_1 be the first leaf of T_v found when walking from v around T_v in clockwise order as
 412 in an Eulerian tour. Continue this Eulerian tour from w_1 and let w_2 and w_3 be the next two
 413 vertices visited. Two cases arise:

414 **Case 1.** If w_1, w_2, w_3 makes a right turn, then let s be the first point hit by the ray
 415 apexed at w_2 that shoots in the direction opposite to w_3 .

416 We claim that w_1 and s lie on the same edge of the boundary of $S_v(C_v)$. Otherwise,
 417 there would be a vertex u visible from w_2 inside the wedge with apex w_2 spanned by w_1 and
 418 w_3 . Note that the first edge of the path $\pi(u, v)$ is the edge uw_2 . Therefore, uw_2 belongs
 419 to the shortest-path T_v contradicting the Eulerian order in which the vertices of this tree
 420 are visited as u should be visited before w_3 . Thus, s and w_1 lie on the same edge and s
 421 can be computed in $O(1)$ time. At this point, we construct the apexed triangle $\Delta(w_1, w_2, s)$
 422 apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. In this case we modify tree T_v
 423 by removing the edge w_1w_2 and replacing the edge w_3w_2 by the edge w_3s ; see Figure 5 (b).

424 **Case 2.** If w_1, w_2, w_3 makes a left turn, then if w_1 and w_3 are adjacent, we construct
 425 an apexed triangle $\Delta(w_1, w_2, w_3)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$.
 426 Otherwise, let s be the first point of the boundary of $S_v(C_v)$ hit by the ray shooting from
 427 w_3 in the direction opposite to w_2 .

428 By the same argument as above, we can show that w_1 and s lie on the same edge of
 429 the boundary of $S_v(C_v)$. Therefore, we can compute s in $O(1)$ time. We then construct an
 430 apexed triangle $\Delta(w_1, w_2, s)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. We
 431 modify the tree T_v by removing the edge w_1w_2 and adding the edge w_3s ; see Figure 5 (c).

432 ► **Lemma 12.** *The above procedure runs in $O(|S_v(C_v)|)$ time and computes $O(|S_v(C_v)|)$
 433 interior disjoint apexed triangles such that their union covers $S_v(C_v)$. Moreover, for each
 434 point $x \in S_v(C_v)$, there is an apexed triangle Δ with apex function $g(x)$ such that (1) $x \in \Delta$
 435 and (2) $g(x) = |\pi(x, v)|$.*

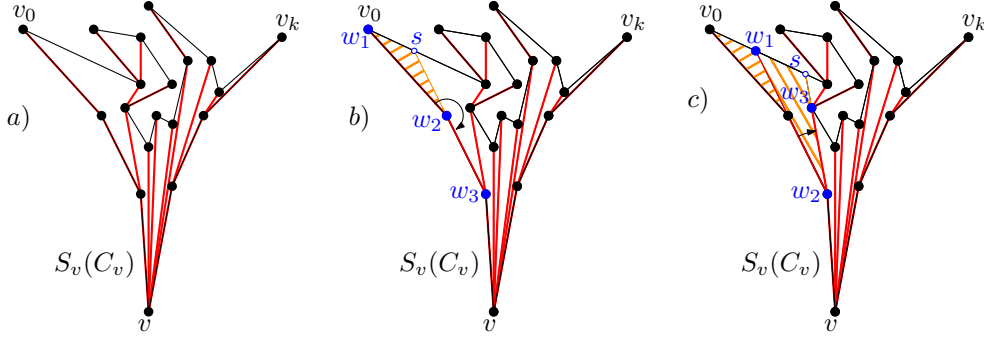


Figure 5 The funnel $S_v(C_v)$ and the shortest-path tree from v are depicted in (a). The two cases of the algorithm described in Lemma 12 are shown in (b) and (c).

Proof. The above procedure splits $S_v(C_v)$ into apexed triangles, such that the apex function in each of them is defined as the geodesic distance to v . Since the path towards v of every point in these triangles has to go through their apex, we obtain properties (1) and (2).

To bound the running time and complexity we proceed as follows. We can compute the shortest-path tree T_v from v in $O(|S_v(C_v)|)$ time [10]. Note that processing either Case 1 or 2 of the algorithm takes constant time. Therefore, we are only interested in the number of times these steps are performed. Further note that we are removing a leaf of the tree in each iteration. In Case 2, the number of leaves strictly decreases, while in Case 1 a new leaf is added if w_1 is not adjacent to w_3 . However, the number of leaves that can be added is at most the number of edges of T_v . Note that the edges added by either Case 1 or 2 are chords of the polygon and hence cannot generate further leaves. Because $|T_v| = O(|S_v(C_v)|)$, we conclude that both Case 1 and 2 are only executed $O(|S_v(C_v)|)$ times yielding the bound in the number of produced apexed triangles and in the running time. ◀

6 Prune and search

In this section, we describe a procedure that finds either the geodesic center of P , or a convex trapezoid that contains the geodesic center. The idea of the proof is to consider the chords of P that bound the apexed triangles computed in previous sections and use a cutting of them that splits P into $O(1)$ cells. Then, we determine on which cell of P the center lies and recurse on that cell as a new subproblem. Naturally, we can discard all apexed triangles that do not intersect the cell containing the center. Using the properties of the cutting, we are able to prove that the size of the subproblem decreases by a constant fraction at each iteration of the algorithm, which leads to an overall linear running time. The usual prune and search algorithm stops whenever only a constant number of apexed triangles intersect the cell containing the center. In this algorithm, we introduce an additional stopping condition: whenever the cell containing the center is a convex trapezoid. In this case we cannot proceed with the prune and search strategy. Nevertheless, by restricting the search space to a convex object, we can use a different optimization technique to find the geodesic center.

Let τ be the set all apexed triangles computed in previous sections.

► **Lemma 13.** *The set τ consists of $O(n)$ apexed triangles.*

Proof. To bound the complexity of τ , recall that E denotes the set of transition edges of P . Because $\sum_{ab \in E} H_{ab} = O(n)$ by Lemma 6 and since there are $O(|H_{ab}|)$ apexed triangles

in each transition hourglass H_{ab} , we conclude that there $O(n)$ apexed triangles constructed using Lemma 6.

Furthermore, we know that $\sum_{v \in M} |S_v(C_v)| = O(n)$. Because each funnel $S_v(C_v)$ is subdivided into $O(|S_v(C_v)|)$ apexed triangles by Lemma 12, we conclude that at most $O(n)$ apexed triangles area constructed inside funnels of marked vertices. Consequently $|\tau| = O(n)$. \blacktriangleleft

Let $\phi(x)$ be the upper envelope of the apex functions of every triangle in τ , i.e.,

$$\phi(x) = \max\{g(x) : g(x) \text{ is the apex function of some apexed triangle of } \tau\}.$$

The following result shows that the $O(n)$ apexed triangle of τ not only cover P , but their apex functions suffice to reconstruct the function $F_P(x)$.

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

Proof. Let p be a point in P , we want to prove that $\phi(p) = F_P(p)$. Two cases arise:

Case 1. If $f(p)$ is a marked vertex, then Lemma 11 implies that $p \in S_{f(p)}(C_{f(p)})$. Therefore by Lemma 12 there is an apexed triangle Δ with apex function $g(x)$ such that $p \in \Delta$ and $g(p) = |\pi(p, f(p))| = F_P(p)$.

Case 2. If $f(p)$ is not marked, then it belongs to the bottom chain of some transition hourglass. Thus, by Lemma 10 there is an apexed triangle Δ with apex function $g(x)$ such that $p \in \Delta$ and $g(p) = F_P(p)$.

Regardless of the case, there is an apexed triangle Δ that contains p such that its apex function $g(p) = F_P(p)$. Since each apex function represents the geodesic distance from some vertex of P , we know that $\phi(p) \leq F_P(p)$, and by construction, $g(p) \leq \phi(p)$. Since $g(p) = F_P(p)$ and as $g(p) \leq \phi(p) \leq F_P(p)$, we conclude that $\phi(p) = F_P(p)$. \blacktriangleleft

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

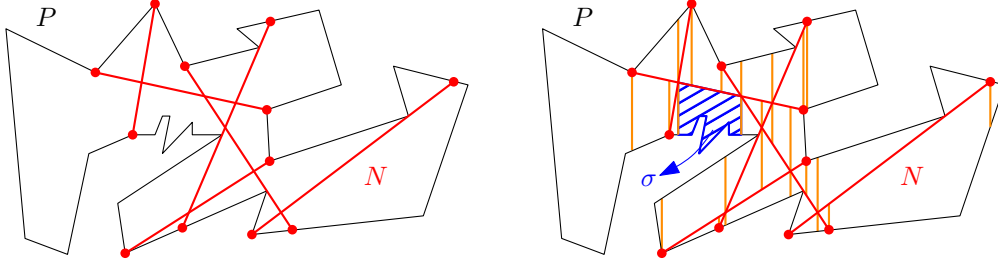
Intuitively, the algorithm described in this section takes as input a cell (in the first iteration the input is simply P) and the set of apexed triangles of τ that intersect this cell. Then, it produces a new cell of smaller complexity that intersects just a fraction of the apexed triangles and contains the geodesic center of P .

Let R be a cell of P and let τ_R be the set of apexed triangles of τ that intersect R . Let $m_R = \max\{|R|, |\tau_R|\}$. Note that each apexed triangle of τ is bounded by at least one chord of P . Let \mathcal{C} be the set containing all chords that bound a triangle of τ_R .

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

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

Observe that N partitions R into $O(1)$ sub-polygons (not necessarily cells). We further refine this partition by performing a cell decomposition. That is, we shoot vertical rays up



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

and down from each endpoint of N , and from the intersection point of any two segments of N , see Figure 6. Overall, this partitions R into $O(1)$ cells such that each either (i) is a convex polygon contained in P of size at most four, or otherwise (ii) contains some chain of ∂P . Since $|N| = O(1)$, the whole decomposition can be computed in $O(m_R)$ time (the intersections between segments of N are done in constant time, and for the ray shooting operations we walk along the boundary of R once).

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

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 cells has constant complexity, we need to perform $O(1)$ calls to the oracle before determining the cell σ that contains the geodesic center of P . Since N is a ϵ -net, we know that at most $\epsilon|C|$ chords of \mathcal{C} intersect σ . Because the chord-oracle computes the minimum of $F_P(x)$ restricted to the chord before determining the side containing the minimum, if c_P lies on any chord bounding σ , then the chord-oracle finds it. Therefore, we can assume that c_P lies in the interior of σ .

Because σ is geodesically convex, if σ is bounded only by chords, then we have found a convex trapezoid σ containing c_P and this phase of the algorithm finishes. Otherwise, σ is a cell bounded by at most three segments (contained in chords), and some chain contained in ∂P ; see Figure 6. In order to proceed with the algorithm on σ recursively, we need to compute the set τ_σ with the at most $\epsilon|C|$ apexed triangles of τ_R that intersect σ .

For each apexed triangle $\triangle \in \tau_R$, we can determine in constant time if it intersects σ (either one of the endpoints is in $\sigma \cap \partial P$ 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 τ_R that intersect σ . Since $|C| \leq 2m_R$, we guarantee that at most $2\epsilon m_R$ apexed triangles intersect σ . Moreover, because each vertex of σ is in at least one apexed triangle of τ_R and from the fact that each apexed triangle covers at most three vertices, we conclude that σ is a cell with at most $6\epsilon m_R$ vertices. Thus, by choosing $\epsilon = 1/12$, we guarantee that both the size of the cell σ and the number of apexed triangles in τ_σ are at most $m_R/2$.

By recursing on σ , we guarantee that after $O(\log m_R)$ iterations, we will find either a convex trapezoid contained in R that contains the center, or we reduce the size of τ_R to a constant in which case the minimum of $F_P(x)$ can be found using an exhaustive search in

547 $O(1)$ time. Since we halve the the size of the cell and the number of apexed triangles in
 548 each iteration, the total running time of this algorithm is $O(m_R)$. Because $|\tau| = O(n)$ by
 549 Lemma 13, we obtain the following result.

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

552 7 Solving the problem restricted to a convex trapezoid

553 To complete the algorithm it remains to show how to find the geodesic center of P within a
 554 convex trapezoid. Recall that $\phi(x)$ denotes the upper envelope of the apex functions of the
 555 triangles in τ . The important thing to notice is that, as in the case of chords, the function
 556 $\phi(x)$ restricted to σ is convex, which allows us to do prune and search using cuttings.

557 Let $\triangle_1, \triangle_2, \dots, \triangle_m$ be the set of $m = O(n)$ apexed triangles of τ that intersect σ . Let
 558 $g_i(x) = |xa_i| + \kappa_i$ be the apex function of \triangle_i , where a_i and w_i are the apex and the definer
 559 of \triangle_i , respectively, and $\kappa_i = |\pi(a_i, w_i)|$ is a constant.

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

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

$$g_i(x) = |xa_i| + \kappa_i \leq r, \text{ if } x \in \triangle_i \text{ for } 1 \leq i \leq m.$$

563 Thus, we need only to find the solution to (P1) to find the geodesic center of P . A similar
 564 optimization was studied by Megiddo in [18]. The main difference being that we have apex
 565 functions, defined only in their corresponding apexed triangles, instead of functions defined
 566 in the entire plane.

We use some remarks described by Megiddo in order to simplify the description of (P1).
 To simplify the formulas, we square the equations:

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

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

$$h_i(x, r) = \|x\|^2 - 2x \cdot a_i + \|a_i\|^2 - r^2 + 2r\kappa_i - \kappa_i^2 \leq 0$$

567 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 $x \in \sigma$ and

$$h_i(x, r) \leq 0, \text{ if } x \in \triangle_i \text{ for } 1 \leq i \leq m.$$

Although the functions $h_i(x, r)$ are not linear, 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*

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

568 As noted by Megiddo, this separating plane has the following property: If the solution
 569 (x, r) to our optimization problem is known to lie to one side of $\gamma_{i,j}$, then we know that one
 570 of the constraints is redundant.

571 In Megiddo's problem, it sufficed to have a *side-decision algorithm* to determine on which
 572 side of a plane $\gamma_{i,j}$ the solution lies. Megiddo showed how to implement such an algorithm
 573 in linear time on the number of constraints [18].

Using this side-decision algorithm, he shows how to solve the optimization problem. A reinterpretation of his technique could be described as follows: Start by pairing the functions arbitrarily, and then 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. An $1/r$ -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 algorithm. 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.

In this paper, 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 a triangular domain. Note that each apexed triangle that intersects σ has its endpoints either outside of σ or on its boundary, i.e., each chord bounding an apexed triangle splits σ into two convex regions.

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 (Δ_i, Δ_j) we define a plane-set as follows: For each chord bounding either Δ_i or Δ_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 (Δ_i, Δ_j) .

Let Γ be the union of all the plane-sets defined by the $m/2$ pairs of apexed triangles. Thus, Γ is a set that consists of $O(m)$ planes. Compute an $1/r$ -cutting of Γ 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]. By using a side-decision algorithm (to be specified later), we can determine the region Q of the cutting that contains the solution. Because Q is the region of a $1/r$ -cutting of Γ , we know that at most $|\Gamma|/r$ planes of Γ 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 (Δ_i, Δ_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 Δ_i nor that of Δ_j . Two cases arise:

Case 1. If either Δ_i or Δ_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 (Δ_i, Δ_j) , we can decide which side of $\gamma_{i,j}$ contains the minimum and drop one of the constraints.

Regardless of the case if the plane-set of a pair (Δ_i, Δ_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 minimum using exhaustive search. Thus, the running time of this algorithm is bounded by the recurrence $T(m) = T(3m/4) + O(m)$ which solves to $O(m)$. Because $m = O(n)$, we can find the solution to (P2) in $O(n)$ time.

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

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

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

The following theorem summarizes the result presented in this paper.

► **Theorem 17.** *Given a simple polygon P with n vertices, we can compute its geodesic center in $O(n)$ time.*

References

- 1 B. Aronov. On the geodesic Voronoi diagram of point sites in a simple polygon. *Algorithmica*, 4(1-4):109–140, 1989.
- 2 B. Aronov, S. Fortune, and G. Wilfong. The furthest-site geodesic Voronoi diagram. *Discrete & Computational Geometry*, 9(1):217–255, 1993.
- 3 T. Asano and G. Toussaint. Computing the geodesic center of a simple polygon. Technical Report SOCS-85.32, McGill University, 1985.
- 4 S. W. Bae, M. Korman, and Y. Okamoto. The geodesic diameter of polygonal domains. *Discrete & Computational Geometry*, 50(2):306–329, 2013.
- 5 S. W. Bae, M. Korman, and Y. Okamoto. Computing the geodesic centers of a polygonal domain. In *Proceedings of CCCG*, 2014.
- 6 S. W. Bae, M. Korman, Y. Okamoto, and H. Wang. Computing the L_1 geodesic diameter and center of a simple polygon in linear time. In *Proceedings of LATIN*, pages 120–131, 2014.
- 7 B. Chazelle. A theorem on polygon cutting with applications. In *Proceedings of FOCS*, pages 339–349, 1982.
- 8 H. Djidjev, A. Lingas, and J.-R. Sack. An $O(n \log n)$ algorithm for computing the link center of a simple polygon. *Discrete & Computational Geometry*, 8:131–152, 1992.
- 9 H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics (TOG)*, 9(1):66–104, 1990.
- 10 L. Guibas, J. Hershberger, D. Leven, M. Sharir, and R. E. Tarjan. Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, 2(1-4):209–233, 1987.

- 667 11 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In
668 *Proceedings of STOC*, pages 50–63. ACM, 1987.
- 669 12 D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM*
670 *Journal on Computing*, 13(2):338–355, 1984.
- 671 13 J. Hershberger and S. Suri. Matrix searching with the shortest path metric. In *Proceedings*
672 *of STOC*, pages 485–494. ACM, 1993.
- 673 14 Y. Ke. An efficient algorithm for link-distance problems. In *Proceedings of SoCG*, pages
674 69–78, 1989.
- 675 15 D.-T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear
676 barriers. *Networks*, 14(3):393–410, 1984.
- 677 16 J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proceedings*
678 *of STOC*, pages 505–511. ACM, 1991.
- 679 17 J. Matoušek. Construction of epsilon nets. In *Proceedings of SoCG*, pages 1–10, New York,
680 1989. ACM.
- 681 18 N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–
682 610, 1989.
- 683 19 J. S. B. Mitchell. Geometric shortest paths and network optimization. In J.-R. Sack and
684 J. Urrutia, editors, *Handbook of Computational Geometry*, pages 633–701. Elsevier, 2000.
- 685 20 B. Nilsson and S. Schuierer. Computing the rectilinear link diameter of a polygon. In
686 *Proceedings of CG*, volume 553 of *LNCS*, pages 203–215, 1991.
- 687 21 B. Nilsson and S. Schuierer. An optimal algorithm for the rectilinear link center of a
688 rectilinear polygon. *Computational Geometry: Theory and Applications*, 6:169–194, 1996.
- 689 22 R. Pollack, M. Sharir, and G. Rote. Computing the geodesic center of a simple polygon.
690 *Discrete & Computational Geometry*, 4(1):611–626, 1989.
- 691 23 S. Suri. *Minimum Link Paths in Polygons and Related Problems*. PhD thesis, Johns Hopkins
692 Univ., 1987.
- 693 24 S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer*
694 *and System Sciences*, 39(2):220–235, 1989.