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

Some Authors[1] and Some more[2]

1 **Dummy University Computing Laboratory**
Address, Country
open@dummyuni.org

2 **Department of Informatics, Dummy College**
Address, Country
access@dummycollege.org

Abstract

Let P be a simple polygon with n vertices. The geodesic center of P is the point inside P that minimizes the geodesic distance to its farthest neighbor. The best known algorithm to compute the geodesic center of P runs in $O(n \log n)$ time and was presented in 1989. Since then, improving this running time has been a longstanding open problem. In this paper, we show how to compute the geodesic center of P in $O(n \log^* n)$ time.

1 Outline

Let P be a simple polygon with n vertices. Given two points $x, y \in P$, let $\pi(x, y)$ denote shortest path contained in P with endpoints x and y, i.e., $\pi(x, y)$ is the *geodesic path* connecting x with y. Notice that the straight-line segment connecting x with y is contained in P, then $\pi(x, y)$ is a straight-line segment. Otherwise, it is a polygonal chain containing only reflex vertices of P other than its endpoints. The *geodesic distance* between x and y, denoted by $|\pi(x, y)|$, is the euclidean length of the path $\pi(x, y)$, i.e., the sum of the lengths of each segment in this path. Throughout this paper, when referring to the distance between two points in P, we refer to the geodesic distance between them.

Given a point $x \in P$, let $f_P(x)$ (or simply $f(x)$ when the context is clear) denote the vertex of P that is (geodesically) farthest from x (if two or more vertices are at the same distance from x, we choose one of them arbitrarily).

Let $F_P(x) = |\pi(x, f(x))|$, i.e., $F_P(x)$ is the distance from x to its farthest neighbor in P. Another way to think of the function $F_P(x)$ is as the upper envelope of all the (geodesic) distance functions from x to the vertices of P. This is related with the farthest-point geodesic Voronoi diagram (FGV) of the vertices of P. The *Voronoi cell* of a vertex v of P is the set of points $R(v) = \{x \in P : F_P(x) = |\pi(x, v)|\}$.

In this paper, we represent the function $F_P(x)$ using a set of constant complexity distance functions defined on triangles contained in P in such a way that the union of these triangles covers P. Moreover, the upper envelope of these functions coincides with $F_P(x)$. Then, we proceed to prune and search for the geodesic of center of P by restricting our search to a sub-polygon of P and the triangles that cover this sub-polygon. By pruning a constant fraction of these triangles on each iteration, we are able to obtain a near-linear time algorithm to compute the geodesic center of P. The boottleneck of this algorithm comes form computing the set of triangles that covers P and their respective distance functions.

2 Hourglasses and Funnels

In this section, we provide 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 $C = (p_0, p_1, \ldots, p_k)$ 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 **??**. We call C and $\partial P(f(a), f(b))$ 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 $f(p_0) \neq f(p_k)$. In particular, an edge ab of ∂P is a *transition edge* if $f(a) \neq f(b)$.

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

Note that we have at most n transition edges, as the FGV of the vertices of P restricted to ∂P partitions the boundary into at most n disjoint connected components [2].

Given a transition chain C of ∂P, we say that the hourglass H_C is the *transition hourglass* of C. By Lemma 1, each transition hourglass is open.

The following results are similar or have already been proved by Suri [12] and Aronov et al. [2]. We provide a sketch of the proof of some of them for completeness.

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

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

▶ **Lemma 3.** *[Rephrase of Lemma 3.4.3 of [2]] Let $C_1, \ldots, C_r$ be a set of edge disjoint transition chains of ∂P that appear in this order when traversing clockwise the boundary of P. Then each chord of P appears in $O(1)$ hourglasses among $H_{C_1}, \ldots, H_{C_r}$.*

Proof. Assume for a contradiction that there is an edge st that appears in the left wall of three hourglasses H_{C_i}, H_{C_j} and H_{C_k} such that $1 \leq i < j < k \leq r$.

Because C_i, C_j and C_k are edge disjoint, their bottom chains are also edge disjoint by Lemma 2. Therefore, st must be a vertex on the walls of these hourglasses. Assume that s is visited before t when going from the top to the bottom chain along the walls of these hourglasses. Let $\pi(s_i, t_i)$ be the wall of S_i that passes through st such that s_i and t_i lie in the top and bottom chains of H_{C_i}, respectively. Define $\pi(s_k, t_k)$ analogously.

Because C_j lies in between C_i and C_k, Lemma 2 implies that the bottom chain of C_j appears between the bottom chains of C_i and C_k. Therefore, C_j lie between s_i and s_k and 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 bottom chain of H_{C_j}, the geodesic path $\pi(x, y)$ is "sandwiched" by the paths $\pi(s_i, t_i)$ and

$\pi(s_k, t_k)$. Thus, $\pi(x, y)$ contains st. However, this implies that the hourglass H_{C_j} is not open—a contradiction that comes from assuming that st lies in the wall of three hourglasses, when this wall is traversed from the top chain to the bottom chain. Analogous arguments can be used to bound the total number of walls that contain the edge st (when traversed in any direction) to $O(1)$. ◀

▶ **Lemma 4.** *[Rephrase of Lemma 4 of [12]] Let C be a transition chain and let T and B be the top and bottom chains of H_C. Let x, y be two vertices such that $\pi(x, y)$ separates T from B. If T_x and T_y are the shortest path trees of x and y in P, then for each $u \in T$ and each $v \in B$, all edges of $\pi(u, v)$, expect perhaps one, belong to $T_x \cup T_y$.*

▶ **Lemma 5.** *Let $C_1, \ldots, C_k$ be a set disjoint transition chains of ∂P. Then*

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

Proof. We claim that then number of chords used by these hourglasses is $O(n)$. If this claim is true, then by Lemma 3 we know that no edge is used more that a constant number of ties yielding our result.

We construct $O(1)$ *split chains* $\gamma_1, \ldots, \gamma_t$ such that for each $1 \leq i \leq k$, there is a split chain γ_j that separates the top and bottom chain of H_{C_i} (to see how to construct this chains see Lemma 2.7.5 of [2]). 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 $O(n)$, and from the fact that the chains $C_1, \ldots, C_k$ are disjoint, Lemma 4 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 $C_1, \ldots, C_k$, we conclude that

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

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

2.2 Funnels

Let $C = (p_0, \ldots, p_k)$ be a chain of the boundary of 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 **??**. Note that the paths $\pi v p_k$ and $\pi v p_0$ may coincide for a while before splitting into disjoint chains. The last vertex in which they coincide is the *apex* of the funnel. See Lee and Preparata [6] or Guibas et al. [3] for more details on funnels.

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

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

▶ **Lemma 6.** *Let v be a vertex of P and let C be a transition chain such that C contains $R(v) \cap \partial P$. Then, $R(v)$ is contained in the funnel $S_v(C)$*

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

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 points of P that are farther from v than from $f(a)$ (*resp.* $f(b)$). Note that α is the boundary of h_a while β bounds h_b.

By definition, we know that $R(v) \subseteq h_a \cap h_b$. Therefore, it suffices to show that $h_a \cap h_b \subset S_v(C)$. Assume for a contradiction that there is a point of $h_a \cap h_b$ lying outside of $S_v(C)$. By continuity, there is a point w lying in the intersection of their boundaries. Without loss of generality, assume that w lies 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

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

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 $S_v(C)$. ◀

3 Decomposing the boundary

Using a result from Hershberger and Suri [5], in $O(n)$ time we can compute the farthest neighbor of each vertex of P. We then 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 which can be computed in $O(n)$ time. In other words, M contains vertices that are guaranteed to have a non-empty Voronoi cell in the FGV of the vertices of P.

Given a vertex v of P, the vertices of P that are farthest from v than from any other vertex appear contiguously along ∂P [2]. Therefore, after computing all this farthest neighbors, we effectively split the boundary into subchains, each associated with a different vertex of M; see Figure **??**.

Let a and b be the endpoints of an edge of ∂P and assume that a appears before b in the clockwise order along ∂P. 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 (if $f(a) \neq f(b)$). For every vertex v that lies between $f(a)$ and $f(b)$ in this clockwise order, we know that there is no vertex u of P such that $f(u) = v$. As proved by Aronov et al. (see Ordering Lemma of [2]), we know that if there is a point x on ∂P such that $v = f(x)$, then x has to lie on the open segment (a, b). In other words, the Voronoi cell $R(v)$ restricted to ∂P is contained in (a, b).

4 Building hourglasses

Let E be the set of transition edges of ∂P. We would like to construct the corresponding hourglass of each transition edges of E. By Lemma 5 $\sum_{ab \in E} |H_{ab}| = O(n)$. Therefore, an output sensitive algorithm would be enough for this task. However, we have no knowledge of the existence of such an algorithm. The algorithm proposed in this section achieves this task at the expense of adding a $O(\log^* n)$ factor to the running time.

To construct these hourglasses, we use the shortest-path data structure of Guibas and Hershberger which can be constructed in $O(n)$ time [4]. Given two points $p, q \in P$, this structure allows us to compute the path $\pi(p, q)$ in $O(\log n + t)$ time, where t is the number of vertices in $\pi(p, q)$. While constructing these paths using directly will take $O(n \log n)$ time,

we overcome this issue by splitting the problem into subproblems whose total complexity amounts to $O(n)$.

To construct the hourglasses, we start by choosing $k = n/\log n$ vertices along ∂P, say $v_0, v_1, \ldots, v_{k-1}$, chosen by walking along ∂P in clockwise order and choosing a vertex each time we skip $\log n$ vertices. That is, there are at most $\log n$ vertices along ∂P between v_i and v_{i+1} (index taken modulo k).

We now compute k hourglasses, that will have at most $\log n$ vertices on the top chain. To this end, for each $0 \leq i \leq k-1$, we compute the path $\pi(v_i, f(v_{i+1}))$ and the path $\pi(v_{i+1}, f(v_i))$. This two paths together with the chain between v_i and v_{i+1} and the chain between $f(v_{i+1})$ and $f(v_i)$ form a hourglass. In this way, we obtain $k = n/\log n$ hourglasses $H_0, \ldots, H_{k-1}$ such that for each $1 \leq i \leq k-1$, the top chain of H_i has at most $\log n$ vertices. Since $\sum_{ab \in E} |H_{ab}| = O(n)$ and from the fact that the boundary of each hourglass is contained in the boundary of a hourglass of an edge of E, we conclude that $\sum_{i=0}^{k-1} |H_{k-1}| = O(n)$. Since $k = n/\log n$, the $O(\log n)$ overhead of the shortest-path queries amounts only to $O(n)$. Thus, we can compute these k hourglasses in $O(n)$ time [4].

We now look at each H_i independently and recursively split H_i into smaller hourglasses as follows. Let E_i be the number of transition edges in the top chain of H_i and let $m_i = |H_i|$. We start by computing the shortest-path data structure for H_i in $O(m_i)$ time. Recall that $|E_i| \leq \log n$ by the way we constructed each H_i. Note that the hourglass of each transition edge of E_i is contained in H_i. Because each of these hourglasses is open, we know that $\sum_{ab \in E_i} |H_{ab}| = O(m_i)$.

Two cases arise: if (1) $|E_i| \leq m_i/\log m_i$, then computing the $|E_i|$ hourglasses defined by the transition edges of E_i takes $O(|E_i| \log m_i + \sum_{ab \in E_i} |H_{ab}|) = O(m_i)$ time. If we are in this case, then we compute each of the hourglasses and finish the recursion. Otherwise, we know that (2) $m_i/\log m_i \leq |E_i|$, and because $|E_i| \leq \log n$, we conclude that $m_i \leq \log^2 n$. Therefore, by splitting the top chain of H_i into $m_i/\log m_i$ chains of length at most $\log m_i$, and repeating the process described above, we can construct at most $m_i/\log m_i$ hourglasses, whose upper chains have at most $\log m_i = O(\log \log n)$ vertices. Moreover, because the boundary of each of these hourglasses is contained in the boundary of a hourglass of an edge of E_i, we conclude their total complexity amounts to $O(m_i)$. Using the shortest-path data structure of H_i, we can compute these $m_i \log m_i$ hourglasses in $O(m_i)$ time. Since $\sum_{i=0}^{k-1} m_i = O(n)$, the total complexity of this step in all hourglasses is at most $O(n)$.

Because the length of the top chain of each hourglasses degreases by taking the logarithmic function, after $O(\log^* n)$ rounds we will be able to compute all the hourglasses for the edges of E. We obtain the following result.

▶ **Lemma 7.** *If E is the set of transition edges of P, then we can construct the hourglass of each edge E in total $O(n \log^* n)$ time.*

5 Covering the polygon with apexed triangles

An *apexed triangle* $\triangle = (a, b, c)$ with *apex* a is triangle contained in P with an associated distance function $g_\triangle(x)$, called the *apex function* of $\triangle$, 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 $\triangle$, such that

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

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 apex triangles that encode all the geodesic distance information inside them. Finally, for each marked vertex $v \in M$, we we construct a funnel that contains the Voronoi region of v. We then decompose this funnel into apex triangles that encode the distance from v.

5.1 Inside the 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 cell is inside H_{ab} when restricted to the boundary of P, but that $R(v) \subset H_{ab}$.

▶ **Lemma 8.** *Let v be a vertex of B_{ab}. If $R(v) \neq \emptyset$, then $R(v) \subset H_{ab}$.*

Proof. Because $R(v) \cap \partial P \subset ab$, by Lemma 6, we know that $R(v) \subset S_v(ab)$. Because H_{ab} is geodesically convex, we conclude that $R(v) \subset S_v(ab) \subset H_{ab}$. ◀

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".

A similar approach was already carried on by Pollack et al. in [11, Section 3]. They show, given a segment contained in the interior of P, how to compute a linear number of apexed triangles such that $F_P(x)$ coincides with the upper envelope of the corresponding apex functions when both are restricted to the given segment.

While the construction we follow is analogous, we use it in the 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 cell $R(v)$ 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. We can compute these trees in $O(|H_{ab}|)$ time [3]. 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 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 **??**.

We further split Δ_v into a series of triangles with apex at v as follows: Let u be a children of v in the tree T_a rooted at a. 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 $v_b v u$ is a left turn; or (3) u is visible from ab, is a child of v only in T_a, and $v_a v u$ is a right turn.

Let $u_1, \ldots, u_{k-1}$ 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 $\max\{d_l(u_i) + |u_i v| : u_i \text{ is a child of } v \text{ of type (2)}\}$. 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 $\triangle(s_i, v, s_{i+1})$ 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 $g_1, \ldots, g_r$ and $\triangle_1, \ldots, \triangle_r$ respectively be an enumeration of all the generated apex functions and triangles such that g_i is defined in the triangle $\triangle_i$. 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 $\triangle_i$ has two vertices on the segment ab and a third vertex, say a_i, called its *apex* such that for each $x \in \triangle_i$, $g_i(x) = |\pi(x, w_i)|$ for some vertex w_i of H_{ab}. We refer to w_i as the *definer* of $\triangle_i$. Intuitively, $\triangle_i$ defines a portion of the geodesic distance function from w_i in a constant complexity region.

▶ **Lemma 9.** *Given a transition edge ab of P, we can compute $O(|H_{ab}|)$ apexed triangles and their respective apex functions in $O(|H_{ab}|)$ time. Moreover, for any point $p \in P$ such that $f(p) \in B_{ab}$, there is a computed apexed triangle $\triangle_i$ with apex function g_i and definer equal to $f(p)$ such that*

1. *$p \in \triangle_i$ and*
2. *$g_i(p) = F_P(p)$.*

Proof. Because $p \in R(f(p))$, Lemma 8 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 $\triangle_i$ apexed at v_i with definer w_i that contains p. Recall that by construction, the apex function $g_i(x)$ of $\triangle_i$ encodes the geodesic distance from x to w_i. Because $F_P(x)$ is the upper envelope of all the geodesic functions, we know that $g_i(p) \leq F_P(p)$.

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

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

$$|\pi(p, f(p))| = |pv_i| + |\pi(v_i, f(p))| \leq |pv_i| + c(v_i) \leq g_i(p).$$

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

$$|\pi(p, f(p))| = |pv_i| + |v_iz| + |\pi(z, f(p))| \leq |pv_i| + |v_iu_j| + d_l(u_j) \leq g_i(p).$$

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

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

In other words, Lemma 9 says that by considering the apex functions we computed, 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 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 $u_1, \ldots, u_{k-1}$ be the set of 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 $C_v = (u_0, \ldots, u_k)$ 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 $\pi(v, u_1)$ 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 $H_{u_{k-1}u_k}$. Thus, since $\pi(v, u_1) \subset H_{u_0u_1}$ while $\pi(v, u_k) \subset H_{u_{k-1}u_k}$, we can compute both $\pi(v, u_1)$ and $\pi(v, u_k)$ in $O(|H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$ time. Consequently, the funnel $S_v(C_v)$ can be constructed in $O(k + |H_{u_0u_1}| + |H_{u_{k-1}u_k}|)$.

Because a vertex on ∂P has a unique farthest neighbor by our general position assumption, and since the total sum of the complexities of the transition hourglasses is $O(n)$ by Lemma 5, we can compute the funnel of each vertex of M in total $O(n)$ time. Since the walls of these funnels are walls of transition hourglasses, and each wall appears at most twice, we get that

$$\sum_{v \in M} |S_v(C_v)| = O(n).$$

▶ **Lemma 10.** *The farthest point Voronoi region of each vertex $v \in M$ is contained in $S_v(C)$.*

Given a funnel $S_v(C)$, we would like to split it into $O(|S_v(C)|)$ apexed triangles. To this end, we compute the shortest path tree T_v of v in $S_v(C)$ in $O(|S_v(C)|)$ time [4]. 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 $|\pi(u, v)|$.

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

Case 1. If w_1, w_2, w_3 makes a left turn, then let s be the first point of the boundary of $S_v(C)$ hit by the ray shooting from w_3 in the direction opposite to w_2 (s could be equal to w_3 if w_3 already lies on the boundary). We claim that s and w_1 lie on the same edge of the boundary of $S_v(C)$. 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 $O(1)$. We then construct an apexed triangle $\triangle(w_1, w_2, s)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. We now modify the tree T_v by removing the edge w_1w_2 and adding the edge w_3s (no edge is added if $w_3 = s$); see Figure **??** for an illustration.

Case 2. If w_1, w_2, w_3 makes a right turn, then let s be the first point hit by the ray apexed at w_2 that shoots in the direction opposite to w_3. By the same argument as above, we can show that w_1 and s lie on the same edge of the boundary of $S_v(C)$. Therefore, we can compute s in $O(1)$ time. At this point, we construct the apexed triangle $\triangle(w_1, w_2, s)$ apexed at w_2 with apex function $g(x) = |xw_2| + |\pi(w_2, v)|$. We now modify the tree T_v by removing the edges w_1w_2 and w_2w_3, and adding the edge w_3s (no edge is added if $w_3 = s$); see Figure **??**.

▶ **Lemma 11.** *The above procedure runs in $O(|S_v(C)|)$ time and computes $O(|S_v(C)|)$ interior disjoint apexed triangles such that their union covers $S_v(C)$. Moreover, for each point $x \in S_v(C)$, there is an apexed triangle $\triangle$ with apex a and apex function $g(x)$ such that (1) $x \in \triangle$ and (2) $g(x) = |\pi(x, v)|$.*

Proof. ◀

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 the apexed triangles computed in previous sections and use a cutting of them that splits P into $O(1)$ cells. Then, we test on which cell the geodesic center lies and recurse on that cell as a new subproblem having smaller complexity. To decrease the complexity of the problem, we consider only the apexed triangles intersecting this cell in the next iteration. Using the properties of the cutting, we are able to prove 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 the geodesic center. 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 are able to perform standard optimization techniques to find the geodesic center.

A *P-chain* is a polygonal chain contained in the boundary of P. A *P-cell* is a simple polygon contained in P bounded by a P-chain and a polygonal chain of length at most four contained in the interior of P that connects the endpoints of this P-chain. Moreover, a P-cell contains the geodesic center of P. The recursive algorithm described in this section takes as input a P-cell (originally the whole polygon P) and the set of apexed triangles that intersect this P-cell, and produces then a new P-cell of smaller complexity.

Let τ be the set all apexed triangles computed in previous sections. Notice that by Lemmas 9 and 11 and since the FGV is defined in the whole polygon P, we know that the union of τ covers P. Moreover, $|\tau| = O(n)$. Given a P-cell R, let τ_R be the set of apexed triangles of τ that intersect R.

Let R be a P-cell and assume that the set τ_R has been computed. Let $m = \max\{|R|, |\tau_R|\}$.

Note that each triangle of τ_R consists of at least one chord of R. Let C be the set containing all chords that bound a triangle of τ_R. A *half-chord* of R is either of the simple polygons in which a chord of R splits this polygon. An *R-trapezoid* is the simple polygon obtained as the intersection of at most four half-chords. Consider a set Q of all open R-trapezoids. For each $q \in Q$, let $C_q = \{c \in C : c \cap q \neq \emptyset\}$ be the set of chords of C induced by q. Finally, let $Q_C = \{C_q : q \in Q\}$ be the family of subsets of C induced by Q.

Consider the range space defined by C and F_C. Let $\varepsilon > 0$. Because the VC-dimension of this range space is finite, we can compute an ε-net N of (C, Q_C) of size $O(\frac{1}{\varepsilon} \log \frac{1}{\varepsilon}) = O(1)$ such that for any R-trapezoid q, if q intersects no chord of N, then q intersects at most $\varepsilon|C|$ chords of C. Note that N can be computed in $O(n)$ time [8].

Since $|N| = O(1)$, we can compute all the intersections in this arrangement in $O(1)$ time. Moreover, by looking at the endpoints of all the chords in N, we can implicitly compute the partition of R into $O(1)$ sub-polygons that this arrangement induces. While each cell of the arrangement is bounded by a constant number of chords from N and a connected chain of the boundary of R, it may not be an R-trapezoid. Therefore, we split them into R-trapezoids by doing a vertical ray-shooting up and down from every vertex of the arrangement; see

Figure **??**. Since only $O(1)$ ray-shootings are performed, this can be done in additional $O(m)$ time by walking the boundary of the polygon R.

We want to decide now which R-trapezoid contains the geodesic center of P. To this end, for each edge of an R-trapezoid, we can extend it to a chord C by doing two ray-shooting queries in $O(m)$ time. Then, we can use the second part of the relative center algorithm introduced by Pollack et al. [11, Section 3] to find the point on C that minimizes $F_P(x)$ (during the first part of their algorithm they compute the equivalent of apex functions restricted to C). This algorithm is an extension of the linear programming technique introduced by Megiddo [9]. 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.

Recall that τ_R consists of all the apexed triangles that intellect R. Thus, we have all the apexed triangles that intersect C. Consequently, Lemmas 9 and 11 imply that the upper envelope of the apex functions coincides with $F_P(x)$ when restricted to C. Let $p \in C$ be the point that archives the minimum of $F_P(x)$ (note that p may be an endpoint of C). We want to decide now on which side of C lies the optimum of $F_P(x)$, i.e., the geodesic center of x. To this end, we consider the apexed triangles whose apex functions define the value of $F_P(x)$ at p. They can be found in $O(m)$ time by looking at all the apexed triangles of τ_R that contain p. We then consider the definers of these apexed triangles. By looking at their distance function to p, which is encoded by the apex functions, we can decide locally on which side of C the function decreases and determine the side that contains the optimum of $F_P(x)$.

Because the algorithm described by Pollack et al. [11, Section 3] runs in linear time on the number of functions defined on C, we can decide in total $O(m)$ time on which side of C the geodesic center of P lies.

Because our decomposition into R-trapezoids has constant complexity, we need to perform this test only $O(1)$ times before determining the R-trapezoid q^* that contains the geodesic center of P. Since N is a ε-net, we know that at most $\varepsilon|C|$ chords of C intersect q^*.

If q^* is contained in the interior of R, then q^* is convex. In this case, we have found a convex trapezoid that contains the solution and this algorithms finishes. Otherwise, q^* is a P-cell bounded by at most three segments, say α, β and γ, and some P-chain R_{q^*}; see Figure **??**. In order to proceed with the algorithm on q^* recursively, we need to compute the set τ_{q*} of at most $\varepsilon|C|$ apexed triangles of τ_R that intersect q^*. We proceed as follows.

For each apexed triangle $\triangle \in \tau_R$, we consider the index of its endpoints and test in $O(1)$ time if any of them lies on the P-chain R_{q^*}. If they do, then they intersect q^*. Otherwise, we know that this triangle has no endpoint in R_{q^*} and could only intersect q^* if one of its edges intersects either α, β or γ. Since this can be tested in $O(1)$ time, we conclude that the at most $\varepsilon|C|$ triangles of τ_R that intersect q^* can be found in $O(m)$ time. Because $|C| \leq 2m$, we guarantee that at most $2\varepsilon m$ apexed triangles intersect q^*. Moreover, because each vertex of q^* 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 q^* consists of at most $6\varepsilon m$. Thus, by choosing $\varepsilon = 1/12$, we guarantee that both the size of the P-cell q^* and the number of apexed triangles in τ_{q^*} are at most $m/2$.

By recursing on q^*, we guarantee that after $O(\log m)$ 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 optimum of $F_P(x)$ can be found using an exhaustive search in $O(1)$ time. Since we halve the the size of the P-cell and the number of apexed triangles in each iteration, the total running time of this algorithm is given by the recurrence $T(m) = T(m/2) + O(m)$ which solves to $T(m) = O(m)$. Because $|\tau| = O(n)$, the total running time of this algorithm on P is $O(n)$.

▶ **Lemma 12.** *In $O(n)$ time we can find either the geodesic center of P or a convex trapezoid containing this center.*

7 Solving the problem restricted to a convex trapezoid

In the previous section we show how to find either the geodesic center of P, or a convex trapezoid q^*contained in P that contains this center. The important thing to notice is that, as in the case of chords, the upper envelope of the apex functions restricted to q^* is a convex function, which allow us to do prune and search using cuttings.

Formally, we have $m = O(n)$ apex triangles $\tau^* = \{\triangle_1, \triangle_2, \ldots, \triangle_m\}$ each with an apex function restricted to this triangle of the form $g_i(x) = |xa_i| + \kappa_i$, where $\kappa_i = |\pi(a_i, w_i,)|$ is a constant, a_i and w_i are the apex and the definer of $\triangle_i$, respectively. At this point, our problem can be reduced to the following optimization problem in $\mathbb{R}^3$:

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

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

This optimization is almost identical to that studied by Megiddo in [10]. The main difference being that we have apex functions instead defined only in their corresponding apexed triangles.

To simplify the formulas, we square the equations:

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

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

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

Therefore, our optimization problem can be reformulated as:

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

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

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*

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

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

In Megiddo's problem, it sufficed to have an *side-decision algorithm* to determine on which side of a plane $\gamma_{i,j}$ the solution lies. Megiddo showed how to implement such an algorithm in linear time on the number of constraints [10].

His technique could be described as follows: Start by pairing the functions arbitrarily, and then consider the set of separating planes defined by these pairs. Then 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 cells such that each intersects at most n/r separating planes. A cutting of planes can be computed in $O(n)$ time in $\mathbb{R}^3$ for any $r = O(1)$ [7]. After computing the cutting, we determine in which of the cells the optimum lies by performing $O(1)$ calls to the side-decision algorithm. Because at least $(r - 1)n/r$ separating planes do not intersect

this constant size cell, 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 partially defined. Note that each apexed triangle that intersects q^* has its endpoints either outside of q^* or on its boundary, i.e., each chord bounding an apexed triangle splits q^* into two convex regions.

7.1 Optimization problem in a convex domain

In this section we describe our algorithm to solve the following optimization problem: Find a point $(x, r) \in \mathbb{R}^3$ such that r is minimized subject to $x \in q^*$ and

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

To this end, we start by pairing 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 eac 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 $(\triangle_i, \triangle_j)$ we define a plane-set as follows: For each chord bounding either $\triangle_i$ or $\triangle_j$, consider the line extending this chord and the vertical extrusion of this line in $\mathbb{R}^3$, that is the vertical 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 $(\triangle_i, \triangle_j)$.

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 cells, each bounded by a constant number of planes [7]. By using a side-decision algorithm (to be specified later), we can determine the cell Q of the cutting that contains the solution. Because Q is the cell 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 $(\triangle_i, \triangle_j)$ 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 doesn't intersect Q, we know that Q' intersects neither the boundary of $\triangle_i$ nor that of $\triangle_j$. Two cases arise:

Case 1. If either $\triangle_i$ or $\triangle_j$ 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 $Q' \subset \triangle_i \cap \triangle_j$, 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 $\triangle_i \cap \triangle_j$ (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 $(\triangle_i, \triangle_j)$, we can decide which side of $\gamma_{i,j}$ contains the optimum and drop one of the constraints.

Regardless of the case if the plane-set of a pair $(\triangle_i, \triangle_j)$ does not intersect Q, then we can drop one of the 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 $m/2 \geq |\Gamma|$ 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 $O(m)$.

▶ **Theorem 13.** *Given a convex trapezoid q^* contained in P such that q^* contains the geodesic center of P. We can compute the geodesic center of P in $O(n)$ time.*

▶ **Corollary 14.** *Given a simple polygon P with n vertices, we can compute its geodesic center in $O(n \log^* n)$.*

8 Conclusions

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 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.

4 L. J. Guibas and J. Hershberger. Optimal shortest path queries in a simple polygon. In *Proceedings of the third annual symposium on Computational geometry*, pages 50–63. ACM, 1987.

5 J. Hershberger and S. Suri. Matrix searching with the shortest path metric. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 485–494. ACM, 1993.

6 D.-T. Lee and F. P. Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

7 J. Matoušek. Approximations and optimal geometric divide-and-conquer. In *Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 505–511. ACM, 1991.

8 J. Matoušek. Construction of epsilon nets. In *Proceedings of the 5th Annual Symposium on Computational Geometry*, pages 1–10, New York, 1989. ACM.

9 N. Megiddo. Linear-time algorithms for linear programming in r3 and related problems. In *Foundations of Computer Science, 1982. FOCS'08. 23rd Annual Symposium on*, pages 329–338. IEEE, 1982.

10 N. Megiddo. On the ball spanned by balls. *Discrete & Computational Geometry*, 4(1):605–610, 1989.

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

12 S. Suri. Computing geodesic furthest neighbors in simple polygons. *Journal of Computer and System Sciences*, 39(2):220–235, 1989.