Efficient Computation of the Geodesic Voronoi Diagram of Points in a Simple Polygon

(Extended Abstract)

Evanthia Papadopoulou and D. T. Lee*

Northwestern University, Evanston, Illinois 60208, USA.

Abstract. We present an $O((n+k)\log(n+k))$ time algorithm for computing the *geodesic Voronoi diagram* of k points in a simple polygon of n vertices improving upon the previously known results. The method introduces a new approach to the construction of geodesic Voronoi diagrams by combining a sweep of the polygon and the merging step of a usual divide-and-conquer strategy.

1 Introduction

Given a simple polygon P with n sides, and a set $S = \{s_1, \ldots, s_k\}$ of k point sites inside P, the *geodesic Voronoi diagram* of S in P is the partitioning of P into k cells $V(s_1), \ldots, V(s_k)$ such that a point $x \in P$ belongs to $V(s_i)$ if and only if the geodesic distance of x from s_i is less than (or equal to) the geodesic distance of x from any other site $s_j \in S$. This diagram is a powerful tool for the solution of several proximity problems involving point sites enclosed in a simple polygon. Examples include the minimum spanning tree, finding the closest pair of sites, and finding the nearest neighbor for every site. Furthermore, it encodes shortest path information from the sites to all points in the polygon, and can thus be used to efficiently answer shortest path queries.

We present an $O((n+k)\log(n+k))$-time algorithm for computing the *geodesic Voronoi diagram* of points in a simple polygon, improving upon Aronov's $O((n+k)\log(n+k)\log n)$ result [1]. An earlier result by Asano and Asano [3] solves the problem in time $O(nk + n\log\log n + k\log k)$. Our method combines a sweep of the polygon and the merging step of a usual divide-and-conquer strategy. It introduces a new approach to the construction of Voronoi diagrams, and can be extended for computing the *geodesic Voronoi diagram* of points in polygons with holes under certain restrictions (see conclusions).

The ordinary Voronoi diagram of point sites in the plane is a classical mathematical object. It is well-known that it can be computed in optimal $O(n\log n)$ time by divide-and-conquer [13], or a sweepline approach [5]. For more information about the Voronoi diagrams, refer to the survey by Aurenhammer [2]. Aronov used a divide-and-conquer algorithm [1] for computing the geodesic Voronoi diagram of point sites in a simple polygon, in which the merge step

* Supported in part by the National Science Foundation under the Grant CCR-9309743, and by the Office of Naval Research under the Grant No. N00014-93-1-0272.

is preceded by a procedure that extends a recursively computed diagram of the subset of sites inside half the polygon to the full polygon, which is the reason for the additional $\log(n+k)$ factor in the time complexity. For the special case where the set of sites includes *all* the reflex vertices of the polygon, the extension phase is skipped and thus, the geodesic Voronoi diagram is computed in $O((n+k)\log(n+k))$ time. For the special case where the set of sites coincides with *the vertices* of a simple n-gon, an alternative $O(n \log n)$ algorithm can be obtained from the *generalized Delaunay triangulation* of a simple polygon [4, 10, 14]. In a general polygonal domain of n vertices, there is basically one subquadratic result by Mitchell [12] which computes the geodesic Voronoi diagram of a set of sites in $O((n+k)^{5/3+\epsilon})$ time and space. This result uses the *continuous Dijkstra paradigm* which simulates the effect of "wavefronts" propagating out of the sites. Very recently, Hershberger and Suri [8] gave an $O(n \log^2 n)$-time and $O(n \log n)$-space algorithm to compute the *shortest path map* of a single source point in a general polygonal domain of n vertices, i.e., $k = 1$. The shortest path map is a planar subdivision that encodes shortest path information from a fixed source to all other points in the plane. Their approach is also based on the continuous Dijkstra paradigm. In a recent manuscript, they improve the complexity to $O(n \log n)$.

In the following, we present an $O((n+k)\log(n+k))$-time algorithm that computes the geodesic Voronoi diagram of a set of k sites in a simple n-gon, using a different approach.

2 An overview of the Algorithm

Consider the dual tree of an arbitrary triangulation of polygon P, where a node corresponds to a triangle, and an edge corresponds to a diagonal of the triangulation. See Figure 1. Assign an arbitrary node of degree one as the *root* of the tree. Then, there is a unique directed path from the root to every triangle which defines a parent-child relation between the triangles of the triangulation. Every triangle Δ has a unique *parent* Δ' which is its predecessor on the dual path from the root to Δ. Let d be the diagonal shared by Δ and Δ'. Diagonal d partitions the polygon into two parts: The part of the polygon "below" d, denoted by $P(d)$, which corresponds to the subtree of the dual tree rooted at Δ, and the part of the polygon "above" d, denoted by $P'(d)$, the complement of $P(d)$. Δ' and Δ are referred to as the triangle *above* d and the triangle *below* d, and are denoted as $\Delta'(d)$ and $\Delta(d)$ respectively.

Our algorithm computes the geodesic Voronoi diagram of S in P in three phases. In the first phase, we sweep the triangulated polygon in the order specified by the postorder traversal of the rooted dual tree. For each diagonal d, we compute the geodesic Voronoi diagram that lies in $\Delta(d)$ of all the sites in $P(d)$. We create an $O(n+k)$ subdivision of P, $SD(P)$, which contains the above information for every triangle. In the second phase, we sweep the triangulated polygon in the order specified by the preorder traversal of the rooted dual tree. We compute the geodesic Voronoi diagram that lies in $\Delta(d)$ of all the sites in

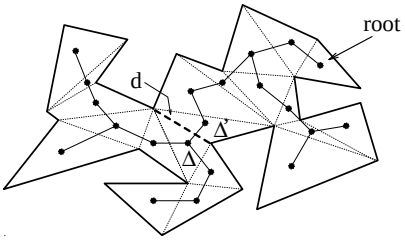

Fig. 1. The dual tree of P.

$P'(d)$, and store it in another $O(n+k)$ subdivision, $SD'(P)$. Finally, we merge the two subdivisions and obtain the geodesic Voronoi diagram of S in P. It is easy to see, that merging in $\Delta(d)$, the Voronoi diagram of sites in $P(d)$ with the Voronoi diagram of sites in $P'(d)$ yields the geodesic Voronoi diagram of S in $\Delta(d)$.

3 Preliminaries

We are given a simple polygon P of n vertices arbitrarily triangulated, and a set of k point sites $S = \{s_1, \ldots, s_k\}$ inside P. Let V denote the vertex set of P. We denote the shortest path between two points $p, q \in P$, that lies totally in P as $\pi(p,q)$. The length of $\pi(p,q)$ is the *geodesic distance* between p and q, and is denoted by $gd(p,q)$. The last vertex on $\pi(p,q)$ before q is called the *anchor* of q (with respect to p). To simplify the presentation, we assume that P and S are in *general position* i.e., no vertex is equidistant from two distinct sites and that no four sites lie on the same geodesic circle. We review some definitions from [1]. See Figure 2 for an illustration.

The Voronoi cell of a site $s_i \in S$ is $V(s_i) = \{x \in P \mid \forall s_j \in S : gd(s_i,x) \leq gd(s_j,x)\}$; s_i is called the *owner* of $V(s_i)$. $V(s_i)$ is partitioned into finer regions by the *shortest path map* from s_i. The *shortest path map* of $V(s_i)$ from s_i is the partition of $V(s_i)$ into maximal sets of points that have the same anchor v with respect to s_i. We refer to such a maximal set of points as the *sub-cell* of $V(s_i)$ induced by vertex v. Vertex v is referred to as the *anchor* of the sub-cell, and it is weighted with its *geodesic distance* from s_i ($w(v) = gd(s_i,v)$). It is well known [6, 1] that the edges of the shortest path map is a collection of *extension segments* emanating from reflex vertices in $V(s_i)$, where the extension segment emanating from a vertex v is the initial part of a half-line collinear with the last edge on $\pi(s_i,v)$, extending in the direction away from s_i, until it hits the boundary of the polygon or the cell $V(s_i)$. The boundary shared by cells $V(s_i)$ and $V(s_j)$ consists of points that are of the same geodesic distance from s_i and s_j, and is the union of portions of bisectors of p and q, where $p, q \in V \cup S$. The bisector of p and q, $p, q \in V \cup S$, is $b(p,q) = \{z \in P | d(z,p) + w(p) = d(z,q) + w(q)\}$, where $d(p,q)$ is the Euclidean distance between p and q. $b(p,q)$ belongs to one of the two branches of the hyperbola with foci p, q and eccentricity $d(p,q)/|w(p) - w(q)|$

[11]. If $w(p) < w(q)$, $b(p,q)$ belongs to the branch closer to q, otherwise it belongs to the branch closer to p. For $w(p) = w(q)$, $b(p,q)$ is a straight line. A point which is common to three or more Voronoi cells or on the boundary of P and two or more Voronoi cells is called a *Voronoi vertex*. Bisectors that are common to two Voronoi cells are called *Voronoi edges*. Endpoints of bisector portions which are not Voronoi vertices are called *breakpoints*. A breakpoint is the point of intersection of a Voronoi edge and an edge of the shortest path map of the incident Voronoi cell. As it was shown in [1], the complexity of the geodesic Voronoi diagram is $O(n+k)$. Augmenting Voronoi cells with the their shortest path map yields the *augmented geodesic Voronoi diagram*. Our algorithm computes the *augmented geodesic Voronoi diagram* of S in P. In the following, we always imply the augmented structure when we refer to the geodesic Voronoi diagram of S in P, and we denote it by $Vor_P(S)$ (see Figure 2). Note that given $Vor_P(S)$ and a query point t, a point location query [9], yields in $O(\log(n+k))$-time the site $s_i \in S$ nearest to t, and also retrieves $\pi(s_i, t)$ in additional time proportional to the number of links in the path.

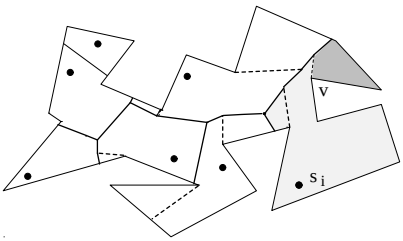

Fig. 2. The geodesic Voronoi diagram of S in P, $Vor_P(S)$. Dashed lines indicate the shortest path map in Voronoi cells; the shaded region is $V(s_i)$, $s_i \in S$; the dark shaded part is the sub-cell of $V(s_i)$ induced by v.

$Vor_P(S)$ can be viewed as a planar graph embedded in P whose vertices are the Voronoi vertices and breakpoints, and edges are the Voronoi edges and shortest path map edges. For notational convenience, we identify the Voronoi diagram with the induced planar graph and use the same notation ($Vor_P(S)$) to denote both. The geodesic Voronoi diagram of $\hat{S} \subseteq S$, restricted to a triangle Δ is denoted as $Vor_\Delta(\hat{S})$. The planar graph induced by $Vor_\Delta(\hat{S})$ consists of the Voronoi vertices and breakpoints of $Vor_P(\hat{S})$ that lie in Δ and the incident Voronoi edges truncated by the sides of Δ. $Vor_\Delta(\hat{S})$ denotes both the diagram and the induced planar graph.

The bisector of two disjoint sets of sites, $S_1, S_2 \subset S$ is $b(S_1,S_2) = \{x \in P \mid gd(x,S_1) = gd(x,S_2)\}$, where $gd(x,S_i) = \min_{s \in S_i}\{gd(x,s)\}$. $b(S_1,S_2)$ is the set of Voronoi edges of $Vor_P(S_1 \cup S_2)$ that are shared by Voronoi cells $V(s_i)$ and $V(s_j)$, for $s_i \in S_1$ and $s_j \in S_2$. As it was shown in [1], for S_1 and S_2 lying on opposite sides of a polygon diagonal, $b(S_1,S_2)$ contains no cycles and all of its components meet the boundary of P. Under the general position assumption, all

components of $b(S_1, S_2)$ are simple paths whose vertices (except the endpoints) have degree two.

As defined in section 2, a triangulation diagonal d partitions the polygon into two parts, $P(d)$ and $P'(d)$. We say that $P(d)$ is the part of the polygon "below" d, and that $P'(d)$ is the part of the polygon "above" d. We assume that $d \in P(d)$. Let $S(d) = S \cap P(d)$ and $S'(d) = S \cap P'(d)$. Consider the triangles incident to d. Throughout the paper, we adopt the following convention: $\Delta(d)$ and $\Delta'(d)$ denote the two triangles that are *below* and *above* the common diagonal d respectively, where $d = \overline{v_1 v_2}$. d_1 and d_2 denote the remaining diagonals of $\Delta(d)$, and d_3 and d_4 denote the remaining diagonals of $\Delta'(d)$. We assume that d_4 is the diagonal shared by $\Delta'(d)$ and its parent. We further assume that d_1, d and d_4 are incident to vertex v_1, and that d_2, d and d_3 are incident to vertex v_2. We denote the vertex shared by d_1 and d_2 as $v_{1,2}$ and the vertex shared by d_3 and d_4 by $v_{3,4}$ (see Figure 3). Let the set of sites in $\Delta(d)$ be $S(\Delta(d)) = S(d) - S(d_1) - S(d_2)$.

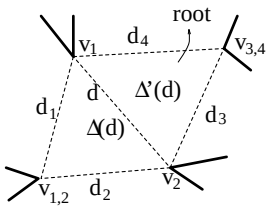

Fig. 3. d,$\Delta(d)$, $\Delta'(d)$

4 The subdivisions

Let $SD(P)$ be the following planar subdivision of P. For every diagonal d, the subdivision contains the Voronoi diagram of $S(d)$ that lies in $\Delta(d)$, $Vor_{\Delta(d)}(S(d))$. See Figure 4. Consider the points of intersection of d with the Voronoi diagrams in $\Delta(d)$ and $\Delta'(d)$ (*i.e.*, $Vor_{\Delta(d)}(S(d))$ and $Vor_{\Delta'(d)}(S(d_4))$). Intersection points that are common to both diagrams belong to the same Voronoi or shortest-path-map edge, and thus are not considered. Intersection points that belong to only one of the diagrams, induce vertices which are referred to as *border* vertices. (In Figure 4c, the border vertex is indicated by an arrow). Any two consecutive border vertices on d (including the endpoints of d) are joined by an edge, if the incident Voronoi cells in $\Delta(d)$ and $\Delta'(d)$ belong to different sites. Edges in $SD(P)$ joining two border vertices or a border vertex and a polygon vertex are called *border* edges. A border edge on d, implies that the owner of the adjacent Voronoi cell in $Vor_{\Delta'(d)}(S(d_4))$ has not been considered in $P(d)$, and that its Voronoi cell should expand in $P(d)$ (see Figures 4c and 5).

$SD'(P)$ is defined similarly. For every diagonal d, $SD'(P)$ contains the Voronoi diagram of $S'(d)$ that lies in $\Delta(d)$, $Vor_{\Delta(d)}(S'(d))$. See Figure 6. The Voronoi diagrams in $\Delta(d)$ and $\Delta'(d)$ (*i.e.*, $Vor_{\Delta(d)}(S'(d))$ *and* $Vor_{\Delta'(d)}(S'(d_4))$) induce

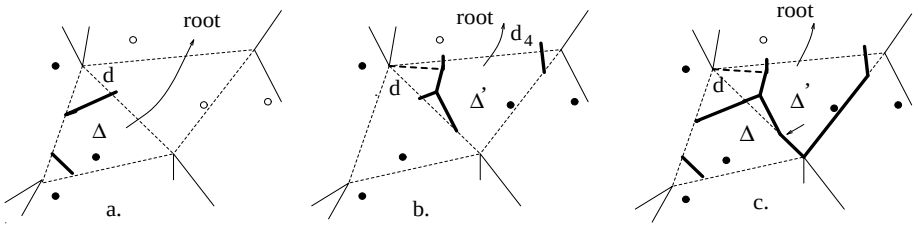

Fig. 4. a. $Vor_{\Delta(d)}(S(d))$, b. $Vor_{\Delta'(d)}(S(d_4))$, c. $SD(P) \cap \{\Delta, \Delta'\}$; border vertices are indicated by arrows.

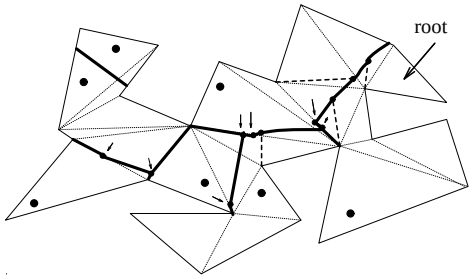

Fig. 5. $SD(P)$: Border vertices are indicated by arrows.

border vertices and *border* edges on d which are defined in exactly the same manner as for $SD(P)$ (see Figure 6c.) In $SD'(P)$, a border edge on a diagonal d implies that the site of the adjacent Voronoi cell in $\Delta(d)$ has not been considered in $P'(d)$ and that its Voronoi cell should expand in $P'(d)$ (see Figures 6c and 7).

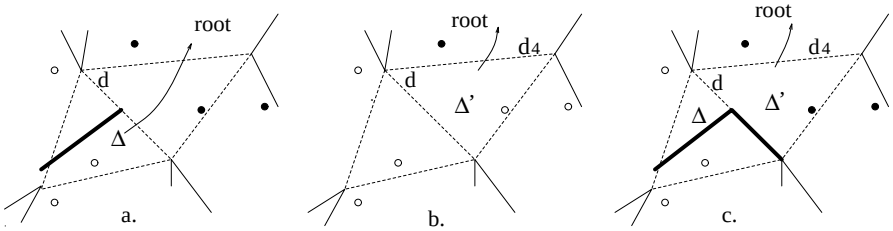

Fig. 6. a. $Vor_{\Delta(d)}(S'(d))$, b. $Vor_{\Delta'(d)}(S'(d_4))$, c. $SD'(P) \cap \{\Delta, \Delta'\}$.

We refer to faces of the two subdivisions as *regions*. Every region of $SD(P)$ and $SD'(P)$ has an *owner* (a site in S) and an *anchor* (a polygon vertex or site in S). Recall that the anchor of a region is the last vertex on the shortest path from the owner to every point in the region, and it is weighted by its geodesic distance from the owner of the region. We denote a region of anchor y, where $y \in V \cup S$, as $r(y)$. The union of regions that have owner $s \in S$ is denoted

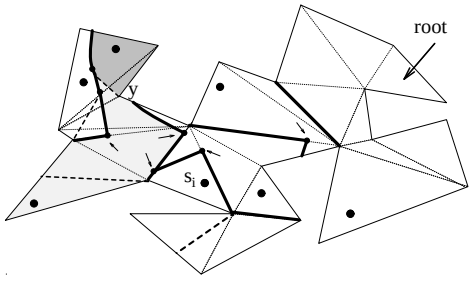

Fig. 7. $SD'(P)$. The shaded portion is $R(s_i)$. The face shaded darker is $r(y) \in R(s_i)$

as $R(s)$. In Figure 7, $R(s_i)$ is shown shaded; the face shaded darker indicates $r(y) \in R(s_i)$.

Given a diagonal d, $SD(d)$ denotes the part of $SD(P)$ "below" d excluding border vertices and edges on d *i.e.*, $SD(d) = SD(P) \cap \{\mathrm{P(d)} - d\}$. $SD'(d)$ denotes the part of $SD'(P)$ "above" d excluding border vertices and edges on d *i.e.*, $SD'(d) = SD'(P) \cap P'(d)$.

Lemma 1. *The number of border vertices of degree two in $SD(P)$ and $SD'(P)$ is $O(n+k)$.*

By lemma 1, Euler's formula, and the fact that the number of faces is $O(n+k)$, we have the following.

Lemma 2. *$SD(P)$ and $SD'(P)$ are planar subdivions of size $O(n+k)$.*

5 Constructing the subdivisions

Consider a polygon diagonal d, and the incident triangles $\Delta(d)$ and $\Delta'(d)$. For brevity, let Δ and Δ' denote $\Delta(d)$ and $\Delta'(d)$ respectively. The Voronoi diagram of $S(\Delta)$ in Δ, $Vor_{\Delta}(S(\Delta))$ can be easily computed in the ordinary divide-and-conquer way by ignoring the polygon boundary. Let $S_{d_i}(\Delta)$, $i = 1, 2$, and $S_d(\Delta)$ denote the ordered list of sites whose regions in $Vor_{\Delta}(S(\Delta))$ intersect d_i and d respectively. Consider the Voronoi diagrams, $Vor_{\Delta}(S(d_i))$, of $S(d_i)$, $i = 1, 2$ in Δ. $Vor_{\Delta}(S(d_i))$, $i = 1, 2$ can be obtained from $SD(d_i)$ using the extension procedure of [1].

In order to compute $SD(P)$, we need $Vor_{\Delta}(S(d))$, the Voronoi diagram of $S(d)$ in Δ. $Vor_{\Delta}(S(d))$ can be computed by merging $Vor_{\Delta}(S(d_1))$, $Vor_{\Delta}(S(\Delta))$, and $Vor_{\Delta}(S(d_2))$. Let's define a subdivision, $SD_1(d)$, to be used as an intermediate step in the construction of $SD(d)$. $SD_1(d)$ is defined as $SD(d)$ for $S(d_2) = \emptyset$ *i.e*, $SD_1(d) = SD(d_1) \cup Vor_{\Delta}(S(d_1) \cup S(\Delta)) \cup \{$*border edges on* $d_1\}$. Border edges on d_1 are induced by regions of sites in $S(\Delta)$ that are adjacent to d_1 in $Vor_{\Delta}(S(d_1)) \cup S(\Delta))$.

Let $\sigma(\Delta)$ denote the set of edges of $SD(d)$ that are shared by regions $R(s_i)$ and $R(s_j)$, for $s_i \in S(d_1) \cup S(\Delta)$ and $s_j \in S(d_2)$. $\sigma(\Delta)$ consists of $\{b(S(d_1) \cup$

$S(\Delta), S(d_2))\} \cap \Delta$ and the incident border edges on d_1 or d_2 (see Figure 8a). Let $\tau(\Delta)$ denote the set of edges of $SD_1(d)$ that are shared by regions $R(s_i)$ and $R(s_j)$, for $s_i \in S(d_1)$ and $s_j \in S(\Delta)$. Note that $\tau(\Delta)$ consists of $\{b(S(d_1), S(\Delta))\}$ $\cap\Delta$ and the incident border edges on d_1 (if any) (see Figure 8b). Under the general position assumption, $\sigma(\Delta)$ and $\tau(\Delta)$ have the following properties (see Figures 8, 9).

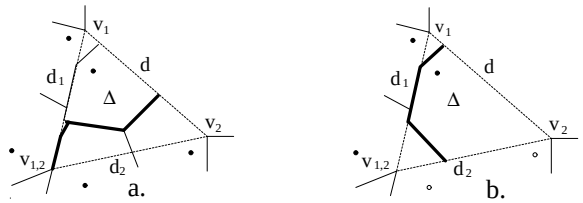

Fig. 8. a. $SD(d) \cap \Delta$; the thick curve is $\sigma(\Delta)$. b. $SD_1(d) \cap \Delta$; the thick curve is $\tau(\Delta)$.

Lemma 3. *If $S(\Delta) = \emptyset$, $\sigma(\Delta)$ is a simple path with one endpoint at vertex $v_{1,2}$ (the vertex incident to d_1 and d_2) and another on diagonal d; $\sigma(\Delta)$ has exactly one common point with d.*

Lemma 4. *If $S(\Delta) \neq \emptyset$, $\sigma(\Delta)$ is a subset of edges of $SD(d)$ with the following properties:*

- *$\sigma(\Delta)$ is a collection of disjoint simple paths, none of which is a cycle.*
- *There is one component of $\sigma(\Delta)$, referred to as the* initial *component, with one endpoint at vertex $v_{1,2}$ and another on d, which is the only point of the component common with d.*
- *The components of $\sigma(\Delta)$ other than the initial, have both their endpoints on d, and these are their only points common with d.*
- *Each component of $\sigma(\Delta)$ other than the initial, contains an edge which is shared by regions $R(s_i)$ and $R(s_j)$, for $s_i \in S_{d_2}(\Delta)$ and $s_j \in S(d_2)$.*

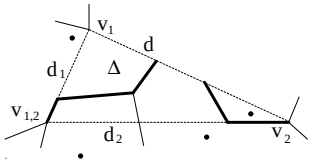

Fig. 9. $\sigma(\Delta)$.

Lemma 5. *$\tau(\Delta)$ is a subset of edges of $SD_1(d)$ with the following properties.*

- *$\tau(\Delta)$ is a collection of disjoint simple paths, none of which is a cycle.*

- *The endpoints of each component of $\tau(\Delta)$ lie on $\{d, d_2\}$, and these are the only points common with $\{d, d_2\}$.*
- *Each component of $\tau(\Delta)$ contains an edge which is shared by regions $R(s_i)$ and $R(s_j)$, for $s_j \in S_{d_1}(\Delta)$ and $s_i \in S(d_1)$.*

Lemmas 3, 4, and 5 imply that the number of connected components of $\sigma(\Delta)$ or $\tau(\Delta)$ is $O(1 + |S(\Delta)|)$.

$SD(P)$ is kept as a doubly-connected-edge-list (DCEL) [13]. For each diagonal d, we compute $SD(d)$ and $T(d)$, the ordered list of regions of $SD(d)$ intersecting d, arranged in a search tree that supports insertions, deletions, splitting, and merging in time logarithmic of its size (for example the red-black tree of [7]). As in [1], every triangle is associated with a bucket to hold bisector intersection points. The algorithm is as follows:

- For every triangle $\Delta = \Delta(d)$, compute the ordinary Voronoi diagram of $S(\Delta)$, $Vor(S(\Delta))$, ignoring the polygon boundary.
- Locate in the triangulation the Voronoi vertices of $Vor(S(\Delta))$ that do not lie in Δ, and store them in the associated buckets. Voronoi vertices lying in triangles of $P'(d)$ are considered for constructing $SD(P)$. Voronoi vertices lying in triangles of $P(d_1)$ and $P(d_2)$ are considered for constructing $SD'(P)$.
- Keep the part of $Vor(S(\Delta))$ that lies in Δ ($Vor_\Delta(S(\Delta))$). Identify $S_{d_1}(\Delta)$, $S_{d_2}(\Delta)$, $S_d(\Delta)$, the ordered list of regions in $Vor_\Delta(S(\Delta))$ intersecting d_1, d_2 and d respectively, and arrange them in search trees [7].
- Call procedure *Construct-SD*(d) for d being one of the two boundary edges of the root triangle. Procedure *Construct-SD*(d) does a postorder traversal of the dual tree.

Algorithm Construct-$SD(d)$
Input: A triangulated polygon P and a triangulation diagonal d.
Output: $SD(d)$ and $T(d)$.
begin

1. If d is a boundary edge such that there is no triangle *below* d then, if there is no site on d let $T(d) = \emptyset$. If d is a boundary edge and $S \cap d \neq \emptyset$ then initialize $T(d)$ to the ordered list of regions induced by the sites on d. Return $T(d)$, $SD(d) = d$.
2. Let $\Delta = \Delta(d)$.
   (a) Recursively call *Construct-SD*(d_1). Let $T(d_1)$ be the ordered list of regions of $SD(d_1)$ intersecting d_1 returned by the recursive call.
   (b) If $T(d_1) \neq \emptyset$, extend $T(d_1)$ into Δ using the algorithm of [1]. Split the extended tree at v_2 (the vertex shared by d and d_2). Let $T_1(d)$ be the part intersecting d, and $T'_1(d_2)$ be the part intersecting d_2. Store $T'_1(d_2)$ in order to be used in the construction of $SD'(P)$. While extending $T(d_1)$, compute $SD(d_1) \cup Vor_\Delta(S(d_1))$.
   (c) Recursively call *Construct-SD*(d_2). Let $T(d_2)$ be the ordered list of regions of $SD(d_2)$ intersecting d_2 returned by the recursive call.
   (d) If $T(d_2) \neq \emptyset$, extend $T(d_2)$ into Δ using the algorithm of [1]. Split the extended tree at v_1 (the vertex shared by d and d_1). Let $T_2(d)$ be the

part intersecting d, and $T_2'(d_1)$ be the part intersecting d_1. Store $T_2'(d_1)$ in order to be used in the construction of $SD'(P)$. While extending $T(d_2)$, compute $SD(d_2) \cup Vor_{\Delta}(S(d_2))$.

(e) If $S(\Delta) = \emptyset$, merge $\{SD(d_1) \cup Vor_{\Delta}(S(d_1))\}$ and $\{SD(d_2) \cup Vor_{\Delta}(S(d_2))\}$ to get $SD(d)$. While merging, update $T_1(d)$ and $T_2(d)$ every time a region is trimmed by $\sigma(\Delta)$. At the end, compute the intersection of the bisector of $\sigma(\Delta)$ intersecting d with its neighboring bisectors in $T_1(d)$ and in $T_2(d)$ respectively. Locate the intersections in the triangulation (without checking feasibility) and assign them to the corresponding buckets.

(f) If $S(\Delta) \neq \emptyset$, merge $\{SD(d_1) \cup Vor_{\Delta}(S(d_1))\}$ and $Vor_{\Delta}(S(\Delta))$ to get $SD_1(d)$. Merge $SD_1(d)$ and $\{SD(d_2) \cup Vor_{\Delta}(S(d_2))\}$ to get $SD(d)$. While merging, update $T_1(d)$, $S_d(\Delta)$ and $T_2(d)$ accordingly. Compute the points of intersection of the bisectors of $\tau(\Delta)$ and $\sigma(\Delta)$ intersecting d with the neighboring bisectors in $T_1(d)$, $S_d(\Delta)$, and $T_2(d)$, and locate them in the triangulation.

(g) Merge $T_1(d)$, $S_d(\Delta)$, and $T_2(d)$ to form $T(d)$. Return $T(d)$ and $SD(d)$.

end.

To extend $SD(d_1)$ and $SD(d_2)$ into Δ we use the *extension* procedure of [1], which given a triangulated polygon separated by a triangulation diagonal e into subpolygons P_1 and P_2, and the planar map induced by the Voronoi diagram of a set of sites in P_1, finds the planar map induced on P by the Voronoi diagram of the same set of points. We briefly review here the main ideas of this procedure.

Each triangle in the triangulation of P_2 is associated with a bucket to hold bisector intersection points. From the Voronoi diagram in P_1, extract the ordered list of regions adjacent to e, and arrange them in a search tree T. For each pair of bisectors bounding a single region in T compute their point of intersection (without checking feasibility), locate them in the triangulation of P_2, and store them in the associated buckets. (Here, we are given $T(d_1)$ and $T(d_2)$ thus, we do not extract the regions; intersections of neighboring bisectors have also been computed.) Move from triangle to neighboring triangle starting at the triangle adjacent to e. Upon entering a triangle through edge e, consider the extension segments of the endpoints of e from their owner in the diagram. (Here, we only move into Δ). Compute their point of intersection with the neighboring bisectors and locate them in the triangulation. Process the intersection points in the bucket of the current triangle in order of increasing distance from e. Process only the feasible intersection points. Each feasible intersection point corresponds to a region deletion and the generation of a new bisector. When construction inside the current triangle is complete, split T at the apex of the triangle giving two subtrees, one for each of the remaining edges of the current triangle. For more details see [1]. Slightly modifying the analysis of [1] we have the following.

Lemma 6. *The time to extend $T(d_i)$, $i = 1, 2$ into Δ (and therefore compute $SD(d_i) \cup \mathrm{Vor}_{\Delta}(S(d_i))$) is $O((|I_f(\Delta)|+1)\log|T(d_i)|+|I_{new}(\Delta)|\log n+|I_{inf}(\Delta)|)$, where $I_f(\Delta)$ denotes the feasible intersection points in the bucket associated with Δ, I_{new} denotes the intersection points generated while extending in Δ, and I_{inf} denotes the infeasible intersection points located in the bucket associated with Δ.*

To merge $SD(d_1) \cup Vor_{\Delta}(S(d_1))$ with $Vor_{\Delta}(S(\Delta))$, we construct $\tau(\Delta)$, split the diagrams along the components of $\tau(\Delta)$, and clean up dominated regions. Similarly, to merge $SD(d_2) \cup Vor_{\Delta}(S(d_2))$ with $SD_1(d)$ we construct $\sigma(\Delta)$, split along the components of $\sigma(\Delta)$, and clean up. Lemmas 3, 4, and 5, provide the means to identify a point on every component of the merge curves. The initial component of $\sigma(\Delta)$ has an endpoint at vertex $v_{1,2}$. For the other components of $\sigma(\Delta)$ (resp. $\tau(\Delta)$) we can identify a point as follows: Let s be the first point in $S_{d_2}(\Delta)$ (resp. $S_{d_1}(\Delta)$) that has not been considered while constructing the initial or other components. Let y be the anchor of the region where s lies in $Vor_{\Delta}(S(d_2))$ i.e., $s \in r(y)$ (resp. $Vor_{\Delta}(S(d_1))$). The region in $Vor_{\Delta}(S(d_i))$ where $s \in S(\Delta)$ lies can be easily determined in $O(\log |T(d_i)|)$ time during the extension procedure. Consider the bisector of s and y, $b(s,y)$. If $b(s,y)$ intersects segment $\overline{sy}$ in Δ then obviously the intersection point belongs to a component of $\sigma(\Delta)$ (resp. $\tau(\Delta)$). Otherwise, the point of intersection of $\overline{sy}$ and d_2 (resp. d_1) is closer to $S(\Delta)$ than to $S(d_2)$ (resp. $S(d_1)$) thus, it belongs to a border edge of $\sigma(\Delta)$ (resp. $\tau(\Delta)$). In any case, a point on a component of $\sigma(\Delta)$ (resp. $\tau(\Delta)$) can be identified in $O(\log |T(d_2)|)$ (resp. $O(\log |T(d_1)|)$ time.

Once a point on a component of $\sigma(\Delta)$ (resp. $\tau(\Delta)$) has been identified the component can be traced in a way similar to tracing the merge curve of the ordinary Voronoi diagram. When a component of $\sigma(\Delta)$ (resp. $\tau(\Delta)$) hits $d_i, i = 1, 2$, (resp. d_1) it continues along d_i (resp. d_1) tracing border edges, until it reaches a point on $b(S(d_2), S(d_1) \cup S(\Delta))$, (resp. $b(S(d_1), S(\Delta))$) in which case it continues in the interior of Δ as $b(S(d_2), S(d_1) \cup S(\Delta))$ (resp. $b(S(d_1), S(\Delta))$). Note that for a point x of $\sigma(\Delta)$ on a border edge of d_i, $gd(x, S(d_1) \cup S(\Delta)) \geq gd(x, S(d_2))$, and that for a point x on a border of $\tau(\Delta)$, $gd(x, S(\Delta)) \geq gd(x, S(d_1))$. By lemmas 3, 4, and 5, when $\sigma(\Delta)$ hits d and when $\tau(\Delta)$ hits $\{d, d_2\}$, the tracing of the component ends.

Lemma 7. *Given* $\mathrm{Vor}_{\Delta}(S(d_1))$, $\mathrm{Vor}_{\Delta}(S(\Delta))$, *and a point on every component of* $\tau(\Delta)$, *the time to trace* $\tau(\Delta)$ *is proportional to the size of* $\tau(\Delta)$ *plus* $|S(\Delta)|$.

Lemma 8. *Given* $SD_1(d)$, $\mathrm{Vor}_{\Delta}(S(d_2))$, *and a point on every component of* $\sigma(\Delta)$, *the time to trace sigma*(Δ) *is proportional to the size of* $\sigma(\Delta)$ *plus* $|S(\Delta)|$.

Lemmas 7 and 8, the fact that $\sigma(\Delta)$ and $\tau(\Delta)$ may have at most $O(1+|S(\Delta)|)$ components, and the fact that the size of $SD(P)$ is $O(n+k)$, imply that that the total time spent for tracing merge curves while constructing $SD(P)$ is $O(n+k)$. Since the number of components of $\sigma(\Delta)$ and $\tau(\Delta)$ is $O(1 + |S(\Delta)|)$, the total time spent to identify a point on components of $\sigma(\Delta)$ and $\tau(\Delta)$ throughout the procedure is $O(k \log(n+k))$. Considering also the time to clean up, generate and locate intersection points, update, split, and merge $T_2(d), T_1(d)$ and $S_d(\Delta)$ we derive the following lemma.

Lemma 9. *The total time spent for merging while constructing* $SD(P)$ *is* $O((n+k)\log(n+k))$.

The total time spent for extending Voronoi diagrams form triangle to triangle depends (by lemma 6) to the total number of intersection points generated

throughout the procedure. Note that bisector intersection points are generated due to $Vor(S(\Delta))$, the bisectors of $\sigma(\Delta)$ and $\tau(\Delta)$ crossing d, and the extension procedure.

Lemma 10. *The total number of bisector intersection points generated during the construction of $SD(P)$ is $O(n+k)$.*

By lemmas 6 and 10 we derive the following.

Lemma 11. *The total time spent for extending during the construction of $SD(P)$ is $O((n+k)\log(n+k))$.*

By the above lemmas, the time complexity to construct $SD(P)$ is $O((n+k)\log(n+k))$.

The construction $SD'(P)$ is similar, only easier, and hence we skip it from this abstract. We compute $SD'(P)$ in a preorder traversal of the rooted dual tree.

6 Merging the two Subdivisions

Once $SD(P)$ and $SD'(P)$ are available it is easy to construct the Voronoi diagram of S in P by merging the two diagrams. Let $\Delta = \Delta(d)$. Let $\phi(\Delta)$ denote the set of Voronoi edges in $Vor_{\Delta}(S)$ that are shared by pairs of Voronoi cells $V(s_i)$ and $V(s_j)$ for s_i in $S(d)$ and $s_j \in S'(d)$. Let $\phi(P)$ denote the union of $\phi(\Delta)$ for every triangle Δ in P. We have the following lemma.

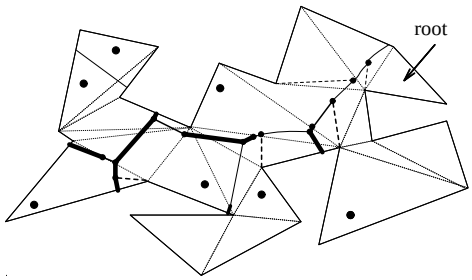

Fig. 10. Merging $SD(P)$ and $SD'(P)$. $\phi(P)$ is shown in thick lines.

Lemma 12. *$\phi(P)$ is a collection of Voronoi edges with the following properties:*

- *$\phi(P)$ consists of disjoint simple paths.*
- *The endpoints of a component of $\phi(P)$ must lie on the polygon boundary or at a border vertex of degree two of $SD(P)$ or $SD'(P)$.*
- *If an endpoint of a component of $\phi(P)$ is at a border vertex u of $SD(P)$ (resp. $SD'(P)$) then the bisector of $\phi(P)$ incident to u and the Voronoi edge in $SD(P)$ (resp. $SD'(P)$) incident to u, belong to the same bisector.*

Lemma 12 implies an algorithm for constructing $Vor_P(S)$ which can be briefly stated as follows: Trace the polygon boundary to identify points which are equidistant from their region owner in $SD(P)$ and their region owner in $SD'(P)$. For every such point, trace the corresponding component of $\phi(P)$ in the usual way [1, 13]. After all components starting at boundary points have been traced, trace the components (if any) with both endpoints at border vertices of degree two. Delete all border edges. Delete all Voronoi edges in $SD(P)$ and $SD'(P)$ incident to border vertices that are not endpoints of $\phi(\Delta)$. Finally, delete all border vertices and merge the two diagrams. It is not hard to see that merging can be done in $O(n+k)$ time.

7 Conclusion

We have presented an $O((n+k)\log(n+k))$-time algorithm for computing the geodesic Voronoi diagram of points in a simple polygon. Our algorithm is generalizable to an $O((n+k)\log(n+k))$-time algorithm for computing the geodesic Voronoi diagram of points in polygons with holes such that there exist *bridges* that partition the polygon into parts of at most one hole (see Figure 11). A *bridge* is a line segment with endpoints on the polygon boundary that is entirely contained in the polygon free space. We first generalize our algorithm for computing the geodesic Voronoi diagram of points in polygons with only one hole as follows. Given a polygon with a hole, transform it into a simple polygon by drawing a diagonal with one endpoint on the polygon boundary and the other on the boundary of the hole, and treating it as ordinary boundary. Compute the Voronoi diagram of points in the obtained simple polygon. Then remove the diagonal and modify the Voronoi diagram accordingly. Given a polygon with more than one hole such that holes are separated by bridges, we triangulate it using the bridges as triangulation diagonals. The dual graph is now a *tree of cycles*, where each cycle corresponds to hole. We call the dual graph *tree of cycles* because it becomes a tree (referred to as the dual *collapsed* tree) if each cycle is collapsed into a single node (referred to as a *collapsed* node). We can now apply the above algorithm using the dual *collapsed* tree as guide, and treating collapsed nodes as polygons with one hole.

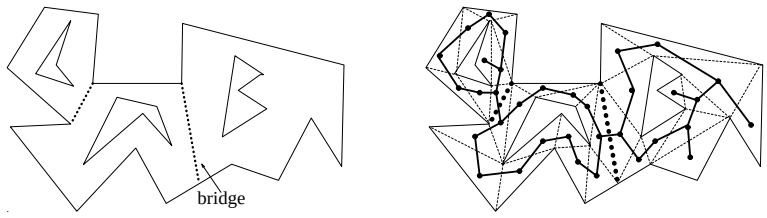

Fig. 11. A polygon with holes that can be partitioned into disjoint parts by *bridges*. The dual graph of the triangulation is a *tree of cycles*.

Furthermore, our approach can be slightly modified to compute the geodesic Voronoi diagram of point sites in polygonal domains where shortest paths are monotone with respect to a constant number of directions. Examples include a polygonal domain of n vertical line segments, where shortest paths are monotone with respect to the horizontal axis, and a polygonal domain of rectangular obstacles under the L_1 metric, where shortest paths are monotone with respect to either the horizontal or the vertical axis. In these cases we construct a subdivision of the plane (e.g., in the case of vertical line segments draw the vertical lines passing through the line segments) such that the dual graph can be transformed to a directed acyclic graph (e.g., in the case of vertical line segments assign a *right* direction to the edges of the dual graph). Sweeping is then guided by the topological sorting of the dual graph. Whether the approach of combining a sweep of the polygonal domain with the merging step of a divide-and-conquer strategy can be used for the geodesic Voronoi diagram of points in a general polygonal domain, is an open problem.

References

1. B. Aronov. "On the geodesic Voronoi diagram of point sites in a simple polygon", *Algorithmica*, 4 (1989), 109-140.
2. F. Aurenhammer, "Voronoi diagrams: A survey of a fundamental geometric data structure," *ACM Comput. Survey,* 23 1991, 345-405.
3. Ta. Asano and Te. Asano, "Voronoi diagrams for points in a polygon," in Discrete Algorithms & Complexity: Perspective in Computing, ed. D. S.Johnson, Academic Press, 1987, 51-64.
4. L. P. Chew, Constrained Delaunay triangulations, *Algorithmica*, 4 (1989), 97-108.
5. S. Fortune, A sweepline algorithm for Voronoi diagrams, *Algorithmica*, 2 (1987), 153-174.
6. L. J. 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 (1987), 209-233.
7. L. J. Guibas and R. Sedgewick, "A dichromatic framework for balanced trees", *Proc. 19th IEEE Symp. on Foundations of Computer Science*, 1978, 8-21.
8. J. Hershberger and S. Suri, "Efficient Computation of Euclidean Shortest Paths in the Plane", *34th Symp. on Foundations of Computer Science*, 1993, 508-517.
9. D. Kirkpatrick, "Optimal Search in Planar Subdivisions" *SIAM J. Computing*, Vol. 12, No 1, 1983, 28-35.
10. D. T. Lee and A. K. Lin, "Generalized Delaunay triangulations for planar graphs", Discrete Computational Geometry, 1 (1986),201-217.
11. D. T. Lee and F. P. Preparata, "Euclidean Shortest Paths in the Presence of Rectilinear Barriers", *Networks*, 14 1984, 393-410.
12. J. S. B. Mitchell, "Shortest paths among obstacles in the plane", *Proc. 9th ACM Symp. on Comput. Geometry*, May 1993, 308-317.
13. Preparata, F. P. and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, New York, NY 1985.
14. C. Wang and L. Schubert, "An optimal algorithm for constructing the Delaunay triangulation of a set of line segments", *Proc. 3rd ACM Symposium on Computational Geometry*, 1987, 223-232.

This article was processed using the LaTeX macro package with LLNCS style