Improved Dynamic Geodesic Nearest Neighbor Searching in a Simple Polygon

Pankaj K. Agarwal[1]
Department of Computer Science, Duke University,
Durham, NC 27708, USA
pankaj@cs.duke.edu

Lars Arge[2]
MADALGO, Aarhus University,
Aarhus, Denmark
large@cs.au.dk

Frank Staals[3]
Dept. of Information and Computing Sciences, Utrecht University,
Utrecht, The Netherlands
f.staals@uu.nl

Abstract

We present an efficient dynamic data structure that supports geodesic nearest neighbor queries for a set S of point sites in a static simple polygon P. Our data structure allows us to insert a new site in S, delete a site from S, and ask for the site in S closest to an arbitrary query point $q \in P$. All distances are measured using the geodesic distance, that is, the length of the shortest path that is completely contained in P. Our data structure achieves polylogarithmic update and query times, and uses $O(n \log^3 n \log m + m)$ space, where n is the number of sites in S and m is the number of vertices in P. The crucial ingredient in our data structure is an implicit representation of a vertical shallow cutting of the geodesic distance functions. We show that such an implicit representation exists, and that we can compute it efficiently.

2012 ACM Subject Classification Computational Geometry

Keywords and phrases data structure, simple polygon, geodesic distance, nearest neighbor searching, shallow cutting

Digital Object Identifier 10.4230/LIPIcs.SoCG.2018.4

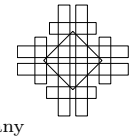
Related Version A full version of this paper is available at `http://arxiv.org/abs/1803.05765`

1 Introduction

Nearest neighbor searching is a classic problem in computational geometry in which we are given a set of point *sites* S, and we wish to preprocess these points such that for a query point q, we can efficiently find the site $s \in S$ closest to q. We consider the case where S is a *dynamic* set of points inside a simple polygon P. That is, we may insert a new site into S or delete an existing one. We measure the distance between two points p and q by the

[1] P.A. is supported by NSF under grants CCF-15-13816, CCF-15-46392, and IIS-14-08846, by ARO under grant W911NF-15-1-0408, and by the U.S.–Israel Binational Science Foundation under grant 2012/229.
[2] L.A. was supported by the Danish National Research Foundation under grant nr. DNRF84
[3] F.S. was supported by the Netherlands Organisation for Scientific Research under project no. 612.001.651.

34th International Symposium on Computational Geometry (SoCG 2018).
Editors: Bettina Speckmann and Csaba D. Tóth; Article No. 4; pp. 4:1–4:14



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

length of the *geodesic* $\Pi(p, q)$, that is, the shortest path connecting p and q that is completely contained in P. We refer to this distance as the *geodesic distance* $\pi(p, q)$.

Related work. It is well known that if we have only a fixed set S of n sites, we can answer nearest neighbor queries efficiently by computing the Voronoi diagram of S and preprocessing it for planar point location. This requires $O(n \log n)$ preprocessing time, the resulting data structure uses linear space, and we can answer queries in $O(\log n)$ time. Voronoi diagrams have also been studied in case the set of sites is restricted to lie in a simple polygon P, and we measure the distance between two points p and q by their geodesic distance $\pi(p, q)$ [4, 18, 24, 25]. The approach of Hershberger and Suri [18] computes the geodesic Voronoi diagram in $O((m + n) \log(m + n))$ time, where m is the total number of vertices in the polygon P, and is applicable even if P has holes. Very recently, Oh and Ahn [24] presented an $O(m + n \log n \log^2 m)$ time algorithm. When $n \leq m/\log^3 m$ this improves the previous results. All these approaches allow for $O(\log(n + m))$ time nearest neighbor queries. However, they are efficient only when the set of sites S is fixed, as inserting or deleting even a single site may cause a linear number of changes in the Voronoi diagram.

To support nearest neighbor queries, it is, however, not necessary to explicitly maintain the (geodesic) Voronoi diagram. Bentley and Saxe [6] show that nearest neighbor searching is a *decomposable search problem*. That is, we can find the answer to a query by splitting S into groups, computing the solution for each group individually, and taking the solution that is best over all groups. This observation has been used in several other approaches for nearest neighbor searching with the Euclidean distance [1, 8, 11]. However, even with this observation, it is hard to get both polylogarithmic update and query time. Chan [8] was the first to achieve this. His data structure can answer Euclidean nearest neighbor queries in $O(\log^2 n)$ time, and supports insertions and deletions in $O(\log^3 n)$ and $O(\log^6 n)$ amortized time, respectively. Recently, Kaplan et al. [19] extended the result of Chan to more general, constant complexity, distance functions.

Unfortunately, the above results do not directly lead to an efficient solution to our problem. The function describing the geodesic distance may have complexity $\Theta(m)$, and thus the results of Kaplan et al. [19] do not apply. Moreover, even directly combining the decomposable search problem approach with the static geodesic Voronoi diagrams described above does not lead to an efficient solution, since every update incurs an $\Omega(m)$ cost corresponding to the complexity of the polygon. Only the very recent algorithm of Oh and Ahn [24] can be made amendable to such an approach. This results in an $O(n + m)$ size data structure with $O(\sqrt{n}(\log n + \log m))$ query time and $O(\sqrt{n} \log n \log^2 m)$ updates. Independently from Oh and Ahn, we developed a different data structure yielding similar results [3]. The core idea in both data structures is to represent the Voronoi diagram implicitly. Moreover, both approaches use similar primitives. In this manuscript, we build on the ideas from our earlier work, and significantly extend them to achieve polylogarithmic update and query times.

Our results. We develop a fully dynamic data structure to support nearest neighbor queries for a set of sites S inside a (static) simple polygon P. Our data structure allows us to locate the site in S closest to a query point $q \in P$, to insert a new site s into S, and to delete a site from S. Our data structure supports queries in $O(\log^2 n \log^2 m)$ time, insertions in $O(\log^5 n \log m + \log^4 n \log^3 m)$ amortized expected time, and deletions in $O(\log^7 n \log m + \log^6 n \log^3 m)$ amortized expected time. The space usage is $O(n \log^3 n \log m + m)$.

Furthermore, we show that using a subset of the tools and techniques that we develop, we can build an improved data structure for when there are no deletions. In this insertion-

only setting, queries take worst-case $O(\log^2 n \log^2 m)$ time, and insertions take amortized $O(\log n \log^3 m)$ time. We can also achieve these running times in case there are both insertions and deletions, but the order of these operations is known in advance. The space usage of this version is $O(n \log n \log m + m)$.

2 An overview of the approach

As in previous work on geodesic Voronoi diagrams [4, 25], we assume that P and S are in general position. That is, (i) no two sites s and t in S (ever) have the same geodesic distance to a vertex of P, and (ii) no three points (either sites or vertices) are colinear. Note that (i) implies that no bisector b_{st} between sites s and t contains a vertex of P.

Throughout the paper we will assume that the input polygon P has been preprocessed for two-point shortest path queries using the data structure by Guibas and Hershberger [13] (see also the follow up note of Hershberger [17]). This takes $O(m)$ time and allows us to compute the geodesic distance $\pi(p, q)$ between any pair of query points $p, q \in P$ in $O(\log m)$ time.

Dynamic Euclidean nearest neighbor searching. We briefly review the data structures for dynamic Euclidean nearest neighbor searching of Chan [8] and Kaplan et al. [19], and the concepts they use, as we will build on these results.

Let F be a set of bivariate functions, and let $\mathcal{A}(F)$ denote the arrangement of the (graphs of the) functions in $\mathbb{R}^3$. In the remainder of the paper we will no longer distinguish between a function and its graph. A point $q \in \mathbb{R}^3$ has *level* k if the number of functions in F that pass strictly below q is k. The *at most k-level* $L_{\leq k}(F)$ is the set of all points in $\mathbb{R}^3$ for which the level is at most k, and the k-level $L_k(F)$ is the boundary of that region.

Consider a collection X of points in $\mathbb{R}^3$ (e.g. a line segment), and let F_X denote the set of functions from F intersecting X. We refer to F_X as the *conflict list* of X. Furthermore, let $\underline{X}$ denote the vertical (downward) projection of X onto the x, y-plane.

A *pseudo-prism* is a constant complexity region in $\mathbb{R}^3$ that is bounded from above by a function, unbounded from below, and whose sides are surfaces vertical with respect to the z-direction. A *a k-shallow $(1/r)$-cutting* $\Lambda_{k,r}(F)$ is a collection of such pseudo-prisms with pairwise disjoint interiors whose union covers $L_{\leq k}(F)$ and for which each pseudo-prism intersects at most n/r functions in F [23]. Hence, for each region (pseudo-prism) $\nabla \in \Lambda_{k,r}(F)$, the conflict list F_∇ contains n/r functions. The number of regions in $\Lambda_{k,r}(F)$ is the *size* of the cutting. Matoušek [23] originally defined a shallow cutting in terms of simplicies, however using pseudo-prisms is more convenient in our setting. In this case the parameter r cannot become arbitrarily large, however we are mostly interested in k-shallow $O(k/n)$-cuttings. In the remainder of the paper we simply refer to such a cutting $\Lambda_k(F)$ as *a k-shallow cutting*. Observe that each pseudo-prism in $\Lambda_k(F)$ is intersected by $O(k)$ functions.

Let $f_s(x)$ be the distance from x to s. The data structures of Kaplan et al. [19] and Chan [8] actually maintain the *lower envelope* $L_0(F)$ of the set of distance functions $F = \{f_s \mid s \in S\}$. To find the site s closest to a query point q they can simply query the data structure to find the function f_s that realizes $L_0(F)$ at q. The critical ingredient in both data structures, as well as our own data structure, is an efficient algorithm to construct a shallow cutting of the functions in F. Chan and Tsakalidis [9] show that if F is a set of linear functions (i.e. planes in $\mathbb{R}^3$), we can compute a k-shallow cutting $\Lambda_k(F)$ of size $O(n/k)$ in $O(n \log n)$ time. Kaplan et al. [19] show how to compute a k-shallow cutting of size $O(n \log^2 n/k)$, in time $O(n \operatorname{polylog} n)$ for a certain class of constant complexity algebraic functions F. Since the geodesic distance function $f_s(x) = \pi(s, x)$ of a single site s may already have complexity

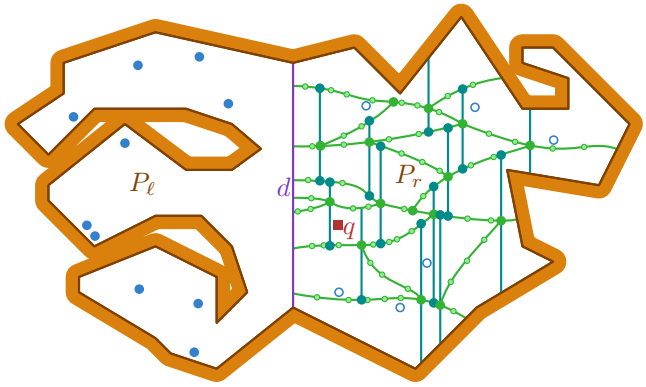


Figure 1 A schematic drawing of the downward projection of an implicit k-shallow cutting $\underline{\Lambda_k(F_\ell)}$ in P_r. The faces are pseudo-trapezoids. Neighboring pseudo-trapezoids share a vertical segment, or part of a bisector. We store only the degree one and three vertices (fat), and their topology.

$\Theta(m)$, any k-shallow cutting of such functions may have size $\Omega(m)$. To circumvent this issue, we allow the regions (pseudo-prisms) to have non-constant complexity. We do this in such a way that we can compactly represent each pseudo-prism, while still retaining some nice properties such as efficiently testing if a point lies inside it. Hence, in the remainder of the paper we will drop the requirement that the regions in a cutting need to have constant complexity.

The general approach. The general idea in our approach is to recursively partition the polygon into two roughly equal size sub-polygons P_ℓ and P_r that are separated by a diagonal. For the sites S_ℓ in the "left" subpolygon P_ℓ, we then consider their geodesic distance functions $F_\ell = \{f_s \mid s \in S_\ell\}$ restricted to the "right" subpolygon P_r, that is, $f_s(x) = \pi(s, x)$, for $x \in P_r$. The crucial part, and our main contribution, is that for these functions F_ℓ we can represent a vertical shallow cutting implicitly. See Fig. 1 for a schematic illustration. More specifically, in $O((n/k)\log^3 n(\log n + \log^2 m) + n\log^2 m + n\log^3 n \log m)$ expected time, we can build a representation of the shallow cutting of size $O((n/k)\log^2 n)$. We can then use this algorithm for building implicitly represented shallow cuttings in the data structure of Chan [8] and Kaplan et al. [19]. That is, we build and maintain the lower envelope $L_0(F_\ell)$. Symmetrically, for the sites in P_r, we maintain the lower envelope $L_0(F_r)$ that their distance functions F_r induce in P_ℓ. When we get a query point $q \in P_r$, we use $L_0(F_\ell)$ to find the site in P_ℓ closest to q in $O(\log^2 n \log m)$ time. To find the site in P_r closest to q, we recursively query in sub-polygon P_r. In total we query in $O(\log m)$ levels, leading to an $O(\log^2 n \log^2 m)$ query time. When we add or remove a site s we, insert or remove its distance function in $O(\log m)$ lower envelope data structures (one at every level). Every insertion takes $O(\log^5 n + \log^4 n \log^2 m)$ amortized expected time, and every deletion takes $O(\log^7 n + \log^6 \log^2 m)$ amortized expected time. Since every site is stored $O(\log m)$ times, this leads to the following main result.

▶ **Theorem 1.** *Let P be a simple polygon P with m vertices. There is a fully dynamic data structure of size $O(n\log^3 n \log m + m)$ that maintains a set of n point sites in P and allows for geodesic nearest neighbor queries in worst case $O(\log^2 n \log^2 m)$ time. Inserting a site takes $O(\log^5 n \log m + \log^4 n \log^3 m)$ amortized expected time, and deleting a site takes $O(\log^7 n \log m + \log^6 n \log^3 m)$ amortized expected time.*

The main complexity is in developing our implicit representation of the k-shallow cutting, and the algorithm to construct such a cutting. Once we have this algorithm we can directly plug it in into the data structure of Chan [8] and Kaplan et al. [19]. Our global strategy is similar to that of Kaplan et al. [19]: we compute an approximate k-level of $\mathcal{A}(F)$ –in our case an implicit representation of this approximate k-level– and then argue that, under certain conditions, this approximate k-level is actually a k-shallow cutting $\Lambda_k(F)$ of $\mathcal{A}(F)$. Our approximate k-level will be a t-level, for some appropriate t, on a random sample of the functions in F. So, that leaves us two problems: *i*) computing an implicit representation of a t-level, and *ii*) computing the conflict lists for all pseudo-prism in our cutting $\Lambda_k(F)$.

For problem *i*), representing the t-level implicitly, we use the connection between the t-level and the t^{th}-order Voronoi diagram. In Section 3 we describe a small, implicit representation of the t^{th}-order Voronoi diagram that still allows us to answer point location queries efficiently. Initially, we use the recent algorithm of Oh and Ahn [24] to construct this representation. In Section 5 we then design an improved algorithm for our particular use case.

For problem *ii*), computing the conflict lists, we will use a data structure developed by Chan [7] together with the implicit Voronoi diagrams that we developed in Section 5. We describe these results in more detail in Section 6. In Section 7, we then show in detail how we can combine all the different parts into a fully dynamic data structure for nearest neighbor queries. Omitted details are in the full version of this paper [2].

3 Implicit representations

Let $F = \{f_s \mid s \in S\}$ denote the set of geodesic distance functions inside the entire polygon P. Our implicit representation of a k-shallow cutting $\Lambda_k(F)$ is based on an implicit representation of the k-level in $\mathcal{A}(F)$. To this end, we first study higher order geodesic Voronoi diagrams.

Higher order Voronoi diagrams. Consider a set of n sites S, a domain $\mathcal{D}$, and a subset $H \subseteq S$ of size k. The *k^{th}-order Voronoi region* $V_k(H, S)$ is the region in $\mathcal{D}$ in which the points are closer to (a site in) H, with respect to some distance metric, than to any other subset $H' \subseteq S$ of size k. The *k^{th}-order Voronoi diagram* $\mathcal{V}_k(S)$ is the partition of $\mathcal{D}$ into such maximal regions $V_k(H, S)$ over all subsets $H \subseteq S$ of size k [21]. Liu et al. [22] study the *geodesic* k^{th}-order Voronoi diagram. They show that $\mathcal{V}_k(S)$ has complexity $O(k(n-k)+km)$. In particular, it consists of $O(k(n-k))$ degree one and degree three vertices, and $O(km)$ degree two vertices (and by our general position assumption, there are no vertices of degree more than three).

Consider a Voronoi region $V_k(H, S)$. Let $e_1, .., e_\ell$, be the edges bounding $\partial V_k(H, S)$, let H_j be the set of sites defining the other Voronoi region incident to e_j, and observe that $H_j \setminus H$ contains a single site q_j [21]. Let $Q = \{q_j \mid j \in [1, \ell]\}$ be the set of sites *neighboring* $V_k(H, S)$. Observe that these results imply that two adjacent regions $V_k(H, S)$ and $V_k(H_j, S)$ in $\mathcal{V}_k(S)$ are separated by a part of a bisector b_{st}, where $s = H \setminus H_j$ and $t = q_j$.

By combining the above two observations we can represent $\mathcal{V}_k(S)$ implicitly. That is, we store only the locations of these degree one and degree three vertices, the adjacency relations between the regions, and the pair of labels (s, t) corresponding to each pair (R_s, R_t) of neighboring regions. See also the recent result of Oh and Ahn [24]. It follows that the size of this representation is linear in the number of degree one and degree three vertices of $\mathcal{V}_k(S)$. We refer to this as the *topological complexity* of $\mathcal{V}_k(S)$.

Representing the k-level. Consider the partition of P into maximally connected regions in which all points in a region have the same k^{th} nearest site in S. Observe that this partition corresponds to the downward projection $\underline{L_k(F)}$ of the k-level $L_k(F)$. As we argue next, this partition is closely related to the k^{th}-order Voronoi diagram defined above.

Lee observes that there is a relation between the i^{th}-order Voronoi diagram and the $(i+1)^{\text{th}}$-order Voronoi diagram [21]. In particular, he shows that we can partition each i^{th}-order Voronoi region $V_i(H,S)$ into $(i+1)^{\text{th}}$ order Voronoi regions by intersecting $V_i(H,S)$ with the (first order) Voronoi diagram of the set of sites neighboring $V_i(H,S)$. More specifically:

▶ **Observation** 2. Let $V_i(H,S)$ be a geodesic i^{th}-order Voronoi region, and let Q be the sites neighboring $V_i(H,S)$. For any point $p \in V_i(H,S)$, the $(i+1)$-closest site from p is the site $s \in Q$ for which $p \in \mathcal{V}(s,Q)$.

▶ **Lemma 3.** *The topological complexity of* $\underline{L_k(F)}$ *is* $O(k(n-k))$.

Proof. By Observation 2, we can obtain $\underline{L_k(F)}$ from $\mathcal{V}_{k-1}(S)$ by partitioning every region $V_{k-1}(H,S)$ using the Voronoi diagram of the sites Q neighboring $V_{k-1}(H,S)$, and merging regions that have the same k^{th} nearest neighbor in the resulting subdivision. Thus, the vertices in $\underline{L_k(F)}$ appear either in $V_{k-1}(H,S)$ or in one of the newly created Voronoi diagrams.

There are $O((k-1)(n-k+1)) = O(k(n-k))$ degree one and degree three vertices that are also vertices in $V_{k-1}(H,S)$. Since the (first order) geodesic Voronoi diagram has $O(k)$ degree one and degree three vertices, a region $V_{k-1}(H,S)$ creates $|Q|$ new degree one and three vertices. Summing over all faces in $\mathcal{V}_{k-1}(H,S)$ this then sums to $O(k(n-k))$. ◀

As with $\mathcal{V}_k(S)$ we can represent $\underline{L_k(F)}$ implicitly by storing only the locations of the degree one and three vertices and the topology of the diagram. Note that if we can efficiently locate the region R_s of $\underline{L_k(F)}$ that contains a query point q, we can also easily compute the z-coordinate of $L_k(F)$ at q, simply by computing the geodesic distance $\pi(s,q)$. Since we preprocessed P for two-point shortest path queries this takes only $O(\log m)$ time (in addition to locating the region of $\underline{L_k(F)}$ containing q).

Representing a vertical decomposition. For every degree three and degree one vertex in $\underline{L_k(F)}$ we now extend a vertical segment up and down until it hits another edge of $\underline{L_k(F)}$. Observe that the topological complexity of the resulting *implicit vertical decomposition* $\underline{L_k^{\nabla}(F)}$ that we obtain still has topological complexity $O(k(n-k))$. Furthermore, each region in $\underline{L_k^{\nabla}(F)}$ is a *pseudo-trapezoid* $\underline{\nabla}$ that is bounded on the left and right either by a vertical segment or a piece of polygon boundary, and on the top and bottom by pieces of bisectors or a piece of polygon boundary. We refer to the four degree one or degree three vertices on the boundary of $\underline{\nabla}$ as the *corners* of $\underline{\nabla}$. See Fig. 2 for an illustration. In the remainder of the paper, we will no longer distinguish between $\underline{L_k^{\nabla}(F)}$ and its implicit representation. In Section 6, we will use such an implicit vertical decomposition to obtain an implicit representation of a shallow cutting.

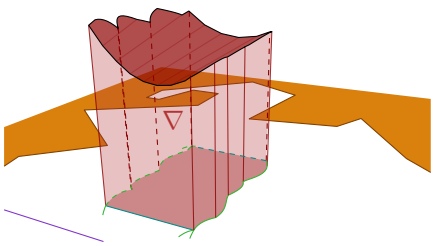


Figure 2 A pseudo prism ∇ and its projection $\underline{\nabla}$ (darker red) onto P_r.

Computing implicit representations. We can use the algorithm of Oh and Ahn [24] to compute the implicit representation of $\mathcal{V}_k(S)$ in $O(k^2 n \log n \log^2 m)$ time. To compute (the representation of) $\underline{L_k(F)}$ we first construct $\mathcal{V}_{k-1}(S)$, and for each region $V_{k-1}(H,S)$ in

$\mathcal{V}_{k-1}(S)$, we again use their algorithm to compute the Voronoi diagrams $\mathcal{V}(Q)$ of the set neighbors Q. We then clip these diagrams to $V_{k-1}(H, S)$. This clipping can be done by a breadth first search in $\mathcal{V}(Q)$, starting with one of the vertices that is also in $\partial V_{k-1}(H, S)$. This takes $O(|Q| \log |Q| \log^2 m)$ time in total. Summing over all faces in $\mathcal{V}_{k-1}(S)$ gives us again a running time of $O(k^2 n \log n \log^2 m)$. Finally, to compute $\underline{L_k^{\nabla}(F)}$ we need to insert two vertical extension segments at each vertex of $\underline{L_k(F)}$. We can find the other endpoint of each extension segment using a point location query. Thus, we obtain the following result.

▶ **Lemma 4.** *An implicit representation $\underline{L_k^{\nabla}(F)}$ of the k-level $L_k(F)$ that uses $O(k(n-k))$ space, can be computed in $O(k^2 n \log n \log^2 m)$ time. Using this representation, the pseudo-prism containing a query point (if it exists) can be determined in $O(\log n + \log m)$ time.*

In Section 5 we will show that if the sites defining the functions in F lie in one half of the polygon and we restrict the functions to the other half we can improve these results. Moreover, we can then compute an implicit representation of a k-shallow cutting of F.

4 Approximating the k-level

An x, y-monotone surface Γ is an *ε-approximation* of $L_k(F)$ if and only if Γ lies between $L_k(F)$ and $L_{(1+\varepsilon)k}(F)$. Following the same idea as in Kaplan et al. [19] we construct an ε-approximation of $L_k(F)$ as follows. We choose a random sample R of F of size $r = (cn/k\varepsilon^2) \log n$ and consider the t-level of $\mathcal{A}(R)$ for some randomly chosen level t in the range $[(1+\varepsilon/3)h, (1+\varepsilon/2)h]$. Here c and c' are some constants, $h = c'/\varepsilon^2 \log n$, and $\varepsilon \in [0, 1/2]$ is the desired approximation ratio. We now argue that $L_t(R)$ is an ε-approximation of $L_k(F)$.

Consider the range space $\mathcal{S} = (F, \mathcal{R})$, where each range in $\mathcal{R}$ is the subset of functions of F intersected by a downward vertical ray in the $(-z)$-direction. See Har-Peled [15] for details on range spaces. An important concept is the *VC-dimension* of $\mathcal{S}$, defined as the size of the largest subset $F' \subseteq F$ for which the number of sets in $\{F' \cap F_\rho \mid F_\rho \in \mathcal{R}\}$ is $2^{|F'|}$.

▶ **Lemma 5** (Lemma 2.3.5 of Aronov et al. [5])**.** *Let s, t, and u be three sites in P. Their bisectors b_{st} and b_{tu} intersect in at most a single point.*

▶ **Lemma 6.** *The VC-dimension of the range space $\mathcal{S}$ is finite.*

Proof. The range space $(F, \mathcal{R})$ is equivalent to $(S_\ell, \mathcal{D})$ where $\mathcal{D} \subseteq 2^{S_\ell}$ is the family of subsets of S_ℓ that lie within some geodesic distance of some point $p \in P$. That is $\mathcal{D} = \{\{s \in S_\ell \mid \pi(p, s) \leq z\} \mid p \in P, z \geq 0\}$. We now observe that any three points s, t, and u, define at most one geodesic disk (Lemma 5). Namely, the disk centered at the intersection point $p = b_{st} \cap b_{tu}$ and radius $\pi(p, s) = \pi(p, t) = \pi(p, u)$. This means the same argument used by as Har-Peled [15, Lemma 5.15] now gives us that the shattering dimension of $\mathcal{S}$ (see [15]) is constant (three). It then follows that the VC-dimension of $\mathcal{S}$ is finite. ◀

Since $\mathcal{S}$ has finite VC-dimension (Lemma 6), and R has size $r = (cn/k\varepsilon^2) \log n \geq \left(\frac{c}{\varepsilon^2 p}\left(\log \frac{1}{p} + \log \frac{1}{q}\right)\right)$, for $p = \frac{k}{2n}$ and $q = 1/n^b$ for some sufficiently large constant b, it follows that with high probability, R is a *relative* $(p, \frac{\varepsilon}{3})$*-approximation* for $\mathcal{S} = (F, \mathcal{R})$ [16]. So, for every range $H \in \mathcal{R}$, we have (whp.) that

$$\left|\frac{|H|}{|F|} - \frac{|H \cap R|}{|H|}\right| \leq \begin{cases} \frac{\varepsilon}{3}\frac{|H|}{|F|}, & \text{if } |H| \geq p|F|, \text{ and} \\ \frac{\varepsilon}{3}p & \text{if } |H| < p|F|. \end{cases} \tag{1}$$

Using exactly the same argument as Kaplan et al. [19] we then obtain the following result.

▶ **Lemma 7.** *The level $L_t(R)$ is an ε-approximation of the k-level $L_k(F)$.*

What remains is to show that the (expected) topological complexity of the t-level $L_t(R)$ is small.

▶ **Lemma 8.** *The expected topological complexity of level $L_t(R)$ is $O((n/k\varepsilon^5)\log^2 n)$.*

Proof. The lower envelope $L_0(R)$ of R has topological complexity $O(r)$, so by Clarkson and Shor [10] the total topological complexity of all levels in the range $[(1+\varepsilon/3)h, (1+\varepsilon/2)h]$ is $O(rh^2)$. Hence, the expected topological complexity of a level $L_t(R)$, with t randomly chosen from this range, is $O(rh^2/h\varepsilon) = O(rh/\varepsilon)$. Substituting $r = (cn/k\varepsilon^2)\log n$ and $h = c'/\varepsilon^2 \log n$, we get $O(nk/\varepsilon^5 \log^2 n)$ as claimed. ◀

Again as in Kaplan et al. [19], if $k < (1/\varepsilon^2)\log n$, we can skip the sampling of the set R, and directly take a random level t in $\mathcal{A}(F)$ in the range $[k, k(1+\varepsilon)]$. This level has also an expected topological complexity of at most $O((n/k\varepsilon^5)\log^2 n)$. Using Lemma 4 (and restarting the computation if the size of the cutting exceeds $O((n/k\varepsilon^5)\log^2 n)$) we then get:

▶ **Lemma 9.** *An ε-approximation of the k-level of $\mathcal{A}(F)$ that has topological complexity $O((n/k\varepsilon^5)\log^2 n)$ can be computed in expected $O((n/k\varepsilon^6)\log^3 n \log^2 m)$ time.*

The main difference between our approach and that of Kaplan et al. [19] is the range space used. In our approach, the ranges are defined by downward vertical rays, whereas in Kaplan et al. the ranges are defined by more general objects. For example, their range space includes a range consisting of the functions intersected by some other constant complexity algebraic function. This allows them to directly turn their approximate level into a shallow cutting since the boundaries of the pseudo-prisms correspond to ranges. Unfortunately, this idea does not directly extend to the geodesic setting, as the VC-dimension of such a range space may again depend on the complexity of the polygon. Therefore, we will use a different approach in Section 6.

5 Computing implicit representations in subpolygon P_r

Consider a diagonal d that splits the polygon P into two subpolygons P_ℓ and P_r, and assume without loss of generality that d is vertical and that P_r lies right of d. We consider only the sites S_ℓ in P_ℓ, and we restrict their functions $F = \{f_s \cap (P_r \times \mathbb{R}) \mid s \in S_\ell\}$ to P_r. We now present a more efficient algorithm to compute the implicit representation of $L_k(F)$ in this setting. To this end, we show that the two-point shortest path query data structure of Guibas and Hershberger [13] essentially gives us an efficient way of accessing the bisector b_{st} between a pair of sites without explicitly computing it. See the full version [2]. We use this to compute an implicit representation of the Voronoi diagram of S_ℓ in P_r. Building on these results, we can compute an implicit representation of the k^{th}-order Voronoi diagram $\mathcal{V}_k(S_\ell)$ in P_r, the k-level $L_k(F)$ of F in $P_r \times \mathbb{R}$, and finally an implicit vertical decomposition $\underline{L_k^\nabla}$ of $\underline{L_k(F)}$ in P_r. We sketch some of our results here. Refer to the full version [2] for the details.

Computing an implicit Voronoi diagram. Our main idea for constructing $\mathcal{V} = \mathcal{V}(S_\ell)$ in P_r, is that we can consider it as a *Hamiltonian abstract Voronoi diagram*. Such a Voronoi diagram can be constructed in $O(Xn)$ time [20], where X is the time required for certain geometric primitives. We can implement these primitives to run in $O(\log^2 m)$ time, resulting in a $O(n \log^2 m)$ time algorithm to compute $\mathcal{V}$.

A Voronoi diagram is *Hamiltonian* if there is a curve –in our case the diagonal d– that intersects all regions exactly once, and furthermore this holds for all subsets of the sites [20]. Let T_ℓ be the subset of sites from S_ℓ whose Voronoi regions intersect d, and thus occur in $\mathcal{V}$.

▶ **Lemma 10.** *The Voronoi diagram $\mathcal{V}(T_\ell)$ in P_r is a Hamiltonian abstract Voronoi diagram.*

To construct $\mathcal{V} = \mathcal{V}(S_\ell) = \mathcal{V}(T_\ell)$ we need the set of sites T_ℓ whose Voronoi regions intersect d, and the order in which they do so. The following lemma is the key insight that allows us to extract T_ℓ from S_ℓ in $O(n \log^2 m)$, assuming that we maintain the sites in S_ℓ in a balanced binary search tree ordered on increasing distance from the bottom endpoint of d.

▶ **Lemma 11.** *Let $s_1, .., s_n$ denote the sites in S_ℓ ordered by increasing distance from the bottom-endpoint p of d, and let $t_1, .., t_z$ be the subset $T_\ell \subseteq S_\ell$ of sites whose Voronoi regions intersect d, ordered along d from bottom to top. For any pair of sites $t_a = s_i$ and $t_c = s_j$, with $a < c$, we have that $i < j$.*

Once we have the set of sites T_ℓ, we construct $\mathcal{V}$ in $O(n \log^2 m)$ time, using the algorithm of Klein and Lingas [20]. Refer to the full version [2] for the details.

▶ **Lemma 12.** *For $s, t \in S_\ell$, the part of the bisector $b^*_{st} = b_{st} \cap P_r$ that lies in P_r is x-monotone.*

It follows from Lemma 12 that we can use the approach of Edelsbrunner, Guibas, and Stolfi [12] to preprocess $\mathcal{V}$ for planar point location queries, and obtain the following result.

▶ **Lemma 13.** *Given a set of n sites S_ℓ in P_ℓ, ordered by increasing distance from the bottom-endpoint of d, the forest $\mathcal{V}$ representing the Voronoi diagram of S_ℓ in P_r can be computed in $O(n \log^2 m)$ time. Given $\mathcal{V}$, finding the site $s \in S_\ell$ closest to a query point $q \in P_r$ requires $O(\log n \log m)$ time.*

Computing $\mathcal{V}_k(S_\ell)$, $L_k(F)$, and an implicit vertical decomposition of the k-level. By combining the algorithm from Lemma 13 with the iterative approach of Lee [21] we can construct an implicit representation of the k^{th}-order Voronoi diagram $\mathcal{V}_k(S_\ell)$ in P_r, or similarly the k-level of F. We then decompose the downward projection $\underline{L_k(F)}$ into *pseudo*-trapezoids, giving us an implicit vertical decomposition.

▶ **Lemma 14.** *A representation $\underline{L^\nabla_k}$ of the k-level $L_k(F)$ consisting of $O(k(n-k))$ pseudo-trapezoids can be computed in $O(k^2 n(\log n + \log^2 m))$ time. Given a query point $q \in P_r$, the k-nearest site in S_ℓ can be reported in $O(\log n \log m)$ time.*

6 An implicit shallow cutting of the geodesic distance function

Let F again denote the set of geodesic distance functions that the sites S_ℓ in P_ℓ induce in P_r. We now argue that we can compute an implicit k-shallow cutting $\Lambda_k(F)$ for these functions.

As in Section 4, let R be our random sample of size r, and let $L_t(R)$ be our approximate k-level of $\mathcal{A}(F)$. Let $\underline{L^\nabla_t(R)}$ be the vertical decomposition of $L_t(R)$. We now raise every pseudo-trapezoid in $\underline{L^\nabla_t(R)}$ to the t-level. Denote the result by Λ. Let $F_p = F_{\rho(p)}$ denote the conflict list of $p \in \mathbb{R}^3$, i.e., the functions intersecting the vertical downward half-line $\rho(p)$ starting in p.

▶ **Lemma 15.** *Let ∇ be a pseudo prism in Λ. The conflict list F_∇ of ∇ is the union of the conflict lists of its corners W, i.e. $F_\nabla = \bigcup_{v \in W} F_v$.*

Proof. Let f_s be the function defining the ceiling of ∇. We have that $F' = \bigcup_{v \in W} F_v \subseteq F_\nabla$ by definition, so we focus on proving $F_\nabla \subseteq F'$. Assume by contradiction that $f_t \in F_\nabla$, but $f_t \notin F'$. So, there is a point $q \in \underline{\nabla}$ for which $\pi(t,q) < \pi(s,q)$, but $\pi(s,v) < \pi(t,v)$ for all corners $v \in W$. Hence, all four corners lie on the "s-side" of b^*_{st}, whereas p lies on the "t-side" of b^*_{st}. Assume without loss of generality that s is closer to the points above b^*_{st} (and thus all corners lie above b^*_{st}). See Fig. 3. Since b^*_{st} is x-monotone (Lemma 12) it must intersect the bottom edge of $\underline{\nabla}$ twice. This bottom edge is part of a single bisector b^*_{su}, for some $f_u \in F$. However, by Lemma 5 b^*_{st} and b^*_{su} intersect at most once. Contradiction. ◀

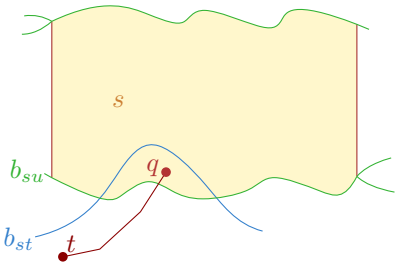


Figure 3 Since the bisectors restricted to P_r are x-monotone it follows that if a site t conflicts with a prism ∇, it must conflict with a corner of ∇.

▶ **Theorem 16.** *Λ is a vertical k-shallow $(k(1+\varepsilon)/n)$-cutting of $\mathcal{A}(F)$ whose topological complexity, and thus its size, is $O((n/k\varepsilon^5)\log^2 n)$. Each pseudo-prism in Λ intersects at least k and at most $4k(1+\varepsilon)$ functions in F.*

Proof. By Lemma 8 Λ consists of $O((n/k\varepsilon^5)\log^2 n)$ regions. Note that all regions are pseudo-prisms. Lemma 15 then gives us that the conflict list of each pseudo-prism is contained in the conflict lists of its at most four corners. ◀

6.1 Computing the conflict lists

Using Lemma 14 we can construct an implicit representation of the k-shallow cutting $\Lambda = \Lambda_k(F)$. So, all that remains is to compute the conflict lists of the pseudo-prisms. By Lemma 15 it is sufficient to compute the conflict lists of the four corner points of each pseudo-prism. Next, we show how to do this in $O(n(\log^3 n \log m + \log^2 m))$ expected time.

We use the same approach as used by Chan [7]. That is, we first build a data structure on our set of functions F so that for a vertical query line ℓ (in $\mathbb{R}^3$) and a value k, we can report the lowest k functions intersected by ℓ in $O((\log n + k)\log m)$ expected time. We then extend this to report only the functions that pass strictly below some point $q \in \mathbb{R}^3$. To compute the conflict lists of all corners in $\Lambda_k(F)$ we repeatedly query this data structure.

The data structure. Our data structure consists of a hierarchy of the lower envelopes of random samples $R_0 \subset R_1 \subset .. \subset R_{\log n}$, where $|R_i| = 2^i$. For each set R_i we store an implicit vertical decomposition representing the (the downward projection of the) lower envelope $L_{0,i} = L_0(R_i)$. This decomposes the space below $L_{0,i}$ into pseudo-prisms. For each such pseudo-prism ∇ we store its conflict list F_∇ with respect to F, i.e. the functions from (the entire set) F that intersect ∇. The following lemma shows that for each R_i, the expected amount of space used is $O(n)$. The total expected space used is thus $O(n \log n)$.

▶ **Lemma 17.** *Let $r \in [1,n]$ and consider a random sample R of F of size r. (i) The expected value of $\sum_\nabla |F_\nabla|$ over all pseudo-prisms below $L_0(R)$ is $O(n)$, and (ii) For any vertical line ℓ, the expected value of $|F_\nabla|$, where ∇ is the pseudo-prism of $L_0(R)$ intersected by ℓ, is $O(n/r)$.*

Proof. The first statement follows directly from a Clarkson and Shor style sampling argument. More specifically, from what Har-Peled [15] calls the "Bounded moments theorem" (Theorem 8.8). The second statement then follows directly from the first statement. ◀

Building the data structure. For each set R_i, we use the algorithm from Section 5 to construct an implicit vertical decomposition of $\underline{L_{0,i}}$. To this end, we need to order the (sites corresponding to the) functions in R_i on increasing distance to the bottom endpoint of the diagonal d. For $R_{\log n} = F$ we do this in $O(n(\log n + \log m))$ time. For R_{i-1} we do this by filtering the ordered set R_i in linear time. Since the sizes of R_i are geometrically decreasing, it follows that we spend $O(n(\log n + \log^2 m))$ time in total.

▶ **Lemma 18.** *Let $f_s \in F \setminus R$ be a function that intersects a pseudo-prism of $L_0(R)$, let T be the set of sites whose functions contribute to $L_0(R)$, ordered on increasing distance from the bottom endpoint of d, and let t and u be the predecessor and successor of s in T, respectively. The vertex $v \in \underline{L_0(R)}$ that represents $d \cap b_{tu}$ is closer to s than to t and u.*

Proof. If f_s intersects a pseudo prism of $L_0(R)$ then there is a point $q \in P_r$ for which s is closer than all other sites in T. It follows that there must be a point on the diagonal d that is closer to s than to all other sites in T. Lemma 11 then gives us that the Voronoi region of s (with respect to $R \cup \{s\}$) on d must lie in between that of t and u (if these still contribute a Voronoi region). Therefore, t and u no longer have a vertex of $\mathcal{V}(R \cup \{s\})$ on d. Since t and u were the closest sites to v in R, this implies that v must lie in the Voronoi region of s, hence s is closer to v than t and u. ◀

By Lemma 18 we can now compute the conflict lists of the cells in $L_{0,i}$ as follows. For each function $f_s \in F \setminus R_i$ we find the vertex v defined in Lemma 18. If s is further from v than the sites defining it, then f_s does not conflict with any pseudo-prism in $L_{0,i}$. Otherwise, we find *all* (degree one or degree three) vertices of $\underline{L_{0,i}}$ that conflict with s. Since Voronoi regions are simply connected, we can do this using a breadth first search in $\underline{L_{0,i}}$, starting from vertex v. When we have this information for all functions in $F \setminus R_i$, we actually also know for every vertex v in $L_{0,i}$ which functions $F \setminus R_i$ pass below it. That is, we have the conflict lists for all vertices v. The conflict list of a pseudo-prism in $L_{0,i}$ is then simply the union of the conflict lists of its four corners (Lemma 15).

Given the ordering of all sites in S on increasing distance to the bottom endpoint of d, we can find the initial vertices for all functions in $F \setminus R_i$ in $O(|R_i| \log m))$ time. For every other reported conflict we spend $O(\log m)$ time, and thus computing the conflict lists for all cells in $L_{0,i}$ takes $O(\sum_{\nabla \in L_{0,i}} |F_\nabla| \log m)$ time. By Lemma 17 this sums to $O(n \log m)$ in expectation. Summing over all $O(\log n)$ random samples, it follows that we spend $O(n \log n \log m)$ expected time to compute all conflict lists. The total expected time to build the data structure is thus $O(n(\log^2 m + \log n \log m))$.

Querying. The query algorithm is exactly as in Chan [7]. The main idea is to use a query algorithm that may fail, depending on some parameter δ, and then query with varying values of δ until it succeeds. The query algorithm locates the cell ∇ in $L_0(R_i)$ stabbed by the vertical line ℓ, for $i = \lceil \log \lceil n\delta/k \rceil \rceil$. If $|F_\nabla| > k/\delta^2$ or $|F_\nabla \cap \ell| < k$ the query algorithm simply fails. Otherwise it reports the k lowest functions intersecting ℓ. Since computing the intersection of a function f_s with ℓ takes $O(\log m)$ time, the running time is $O((\log n + k/\delta^2) \log m)$. Using three independent copies of the data structure, and querying with $\delta = 2^{-j}$ for increasing j gives us an algorithm that always succeeds in $O((\log n + k) \log m)$ time. Refer to Chan [7] for details. We can now also report all functions that pass below a point q by repeatedly querying with the vertical line through q and doubling the value of k. This leads to a query time of $O((\log n + k) \log m)$, where k is the number of functions passing below q.

▶ **Theorem 19.** *There is a data structure of size $O(n \log n)$ that allows reporting the k lowest functions in $\mathcal{A}(F)$ intersected by a vertical line through a query point $q \in P_r$, that is, the k-nearest neighbors of a query point q, or all k functions that pass below q, in $O((\log n + k) \log m)$ time. Building the data structure takes $O(n(\log n \log m + \log^2 m))$ expected time.*

Computing a shallow cutting. To construct a shallow cutting we now take a random sample R of size r, build an implicit representation of the t-level in this sample, and then construct the above data structure to compute the conflict lists. By Lemma 14 constructing the implicit representation of $L_t(R)$ takes $O(t^2 r(\log r + \log^2 m))$ time. Plugging in $r = (cn/k\varepsilon^2) \log n$, $t = \Theta(1/\varepsilon^2 \log n)$, and $\varepsilon = 1/2$, this takes $O((n/k) \log^3 n(\log n + \log^2 m))$ expected time.

Constructing the query data structure takes $O(n(\log n \log m + \log^2 m))$ time. We then query it with all degree three and degree one vertices in Λ. The total size of these conflict lists is $O(n \log^2 n)$ (Theorem 16). So, this takes $O(n \log^3 n \log m)$ time in total. We conclude:

▶ **Theorem 20.** *A k-shallow cutting $\Lambda_k(F)$ of F of topological complexity $O((n/k) \log^2 n)$ can be computed in $O((n/k) \log^3 n(\log n + \log^2 m) + n \log^2 m + n \log^3 n \log m)$ expected time.*

7 Putting everything together

Kaplan et al. [19] essentially prove the following result, which, combined with Theorem 20 gives us an efficient data structure to answer nearest neighbor queries when sites are in P_ℓ and the query points are in P_r.

▶ **Lemma 21** (Kaplan et al. [19])**.** *Given an algorithm to construct a k-shallow cutting Λ of size $S(n, k)$ on n functions in $T(n, k)$ time, and such that locating the cell ∇ in Λ containing a query point q takes $Q(n, k)$ time, we can construct a data structure of size $O(S(n, k) \log n)$ that maintains a dynamic set of at most n functions F and can report the function that realizes the lower envelope $L_0(F)$ at a query point q in $O(Q(n, 1) \log n)$ time. Inserting a new function in F takes $O((T(n, 1)/n) \log n)$ amortized time, and deleting a function from F takes $O((T(n, 1)/n) \log^3 n)$ amortized time.*

Our main data structure is a balanced binary tree, corresponding to a balanced decomposition of P into sub-polygons [14], in which each node stores two copies of the data structure from Lemma 21. A node in the tree corresponds to a subpolygon P' of P, and a diagonal d that splits P' into two roughly equal size subpolygons P_ℓ and P_r. One copy of our data structure associated with this node stores the sites in S_ℓ and can answer queries in P_r. The other copy stores the sites in S_r and can answer queries in P_ℓ. Since the balanced hierarchical decomposition consists of $O(\log m)$ layers, every site is stored $O(\log m)$ times. This results in an $O(n \log^3 n \log m + m)$ size data structure. To answer a query q, we query $O(\log m)$ data structures, one at every level of the tree, and we report the site that is closest over all.

▶ **Theorem 1.** *Let P be a simple polygon P with m vertices. There is a fully dynamic data structure of size $O(n \log^3 n \log m + m)$ that maintains a set of n point sites in P and allows for geodesic nearest neighbor queries in worst case $O(\log^2 n \log^2 m)$ time. Inserting a site takes $O(\log^5 n \log m + \log^4 n \log^3 m)$ amortized expected time, and deleting a site takes $O(\log^7 n \log m + \log^6 n \log^3 m)$ amortized expected time.*

Proof. Theorem 20 gives us $T(n, k) = O((n/k) \log^3 n(\log n + \log^2 m) + n \log^2 m + n \log^3 n \log m)$, $S(n, k) = O(n \log^2 n)$, and $Q(n, k) = O(\log n \log m)$, where m is the size of our polygon. Therefore, $T(n, 1)/n = O(\log^4 n + \log^3 n \log^2 m)$. Plugging in these results in Lemma 21 and using that the balanced decomposition consists of $O(\log m)$ levels completes the proof. ◀

In case there are only insertions and no deletions, or the full sequence of updates is known in advance, we can design an alternative data structure, using a subset of our results. The main idea is still to recursively partition the polygon subpolygons P_ℓ and P_r. Instead of building a dynamic lower envelope data structure of the sites $S_\ell = S \cap P_\ell$ in P_r, we further split the sites S_ℓ into subsets $S_1, .., S_k$. For each subset we use the algorithm from Section 5 to build (an implicit representation of) the Voronoi diagram they induce in P_r. To answer a query (of a query point in P_r) we simply query *all* k (implicit) Voronoi diagrams. To update the data structure we simply rebuild the Voronoi diagram(s) of the affected subset(s). By appropriately splitting S_ℓ we get $O(\log^2 n \log^2 m)$ query time and $O(\log n \log^3 m)$ amortized update time, using $O(n \log n \log m + m)$ space. See the full version [2] for the details.

References

1. Pankaj K. Agarwal and Jiří Matoušek. Dynamic Half-Space Range Reporting and its Applications. *Algorithmica*, 13(4):325–345, 1995.
2. Pankaj Agarwal K., Lars Arge, and Frank Staals. Improved dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1803.05765, 2018.
3. Lars Arge and Frank Staals. Dynamic geodesic nearest neighbor searching in a simple polygon. *CoRR*, abs/1707.02961, 2017.
4. Boris Aronov. On the Geodesic Voronoi Diagram of Point Sites in a Simple Polygon. *Algorithmica*, 4(1):109–140, 1989.
5. Boris Aronov, Steven Fortune, and Gordon Wilfong. The furthest-site geodesic voronoi diagram. *Discrete & Computational Geometry*, 9(3):217–255, Mar 1993.
6. Jon Louis Bentley and James B Saxe. Decomposable searching problems I. Static-to-dynamic transformation. *Journal of Algorithms*, 1(4):301–358, 1980.
7. Timothy M. Chan. Random Sampling, Halfspace Range Reporting, and Construction of ($\leq k$)-levels in Three Dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.
8. Timothy M. Chan. A Dynamic Data Structure for 3-D Convex Hulls and 2-D Nearest Neighbor Queries. *Journal of the ACM*, 57(3):16:1–16:15, March 2010.
9. Timothy M. Chan and Konstantinos Tsakalidis. Optimal Deterministic Algorithms for 2-d and 3-d Shallow Cuttings. In *Proc. 31st International Symposium on Computational Geometry*, volume 34 of *Leibniz International Proceedings in Informatics*, pages 719–732. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015.
10. Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.
11. David Dobkin and Subhash Suri. Maintenance of Geometric Extrema. *Journal of the ACM*, 38(2):275–298, April 1991.
12. Herbert Edelsbrunner, Leo J. Guibas, and Jorge Stolfi. Optimal Point Location in a Monotone Subdivision. *SIAM Journal on Computing*, 15(2):317–340, May 1986.
13. Leonidas J. Guibas and John Hershberger. Optimal Shortest Path Queries in a Simple Polygon. *Journal of Computer and System Sciences*, 39(2):126 – 152, 1989.
14. Leonidas J. Guibas, John Hershberger, Daniel Leven, Micha Sharir, and Robert E. Tarjan. Linear-Time Algorithms for Visibility and Shortest Path Problems Inside Triangulated Simple Polygons. *Algorithmica*, 2(1):209–233, 1987.
15. Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173. American mathematical society Boston, 2011.
16. Sariel Har-Peled and Micha Sharir. Relative (p, ϵ)-approximations in Geometry. *Discrete & Computational Geometry*, 45(3):462–496, Apr 2011.
17. John Hershberger. A new data structure for shortest path queries in a simple polygon. *Information Processing Letters*, 38(5):231–235, June 1991.

18 John Hershberger and Subhash Suri. An Optimal Algorithm for Euclidean Shortest Paths in the Plane. *SIAM Journal on Computing*, 28(6):2215–2256, 1999.

19 Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic Planar Voronoi Diagrams for General Distance Functions and their Algorithmic Applications. In *Proc. 28th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2017.

20 Rolf Klein and Andrzej Lingas. *Hamiltonian abstract Voronoi diagrams in linear time*, pages 11–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 1994.

21 Der-Tsai Lee. On k-nearest neighbor voronoi diagrams in the plane. *IEEE Transactions on Computers*, C-31(6):478–487, June 1982.

22 Chih-Hung Liu and D. T. Lee. Higher-order geodesic voronoi diagrams in a polygonal domain with holes. In *Proc. 24th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 1633–1645, 2013.

23 Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry Theory and Applications*, 2(3):169–186, 1992.

24 Eunjin Oh and Hee-Kap Ahn. Voronoi Diagrams for a Moderate-Sized Point-Set in a Simple Polygon. In *Proc. 33rd International Symposium on Computational Geometry*, volume 77 of *Leibniz International Proceedings in Informatics*, pages 52:1–52:15. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2017.

25 Evanthia Papadopoulou and Der-Tsai Lee. A New Approach for the Geodesic Voronoi Diagram of Points in a Simple Polygon and Other Restricted Polygonal Domains. *Algorithmica*, 20(4):319–352, 1998.