Computing Farthest Neighbors on a Convex Polytope[*]

Otfried Cheong[†] Chan-Su Shin[‡] Antoine Vigneron[§]

March 28, 2002

Abstract

Let N be a set of n points in convex position in R^3. The farthest-point Voronoi diagram of N partitions R^3 into n convex cells. We consider the intersection $G(N)$ of the diagram with the boundary of the convex hull of N. We give an algorithm that computes an implicit representation of $G(N)$ in expected $O(n \log^2 n)$ time. More precisely, we compute the combinatorial structure of $G(N)$, the coordinates of its vertices, and the equation of the plane defining each edge of $G(N)$. The algorithm allows us to solve the all-pairs farthest neighbor problem for N in expected time $O(n \log^2 n)$, and to perform farthest-neighbor queries on N in $O(\log^2 n)$ time with high probability.

1 Introduction

Let N be a set of n points in three dimensions. Its diameter is the maximum distance between any two points in N. The problem of computing the diameter has been intensively studied in the past two decades. Indeed, back in 1985 Preparata and Shamos [6] called it a source of frustration to many workers. After Clarkson and Shor [4] gave a simple randomized algorithm that runs in optimal expected $O(n \log n)$ time, most work on the problem has concentrated on finding a matching deterministic algorithm. After considerable efforts by several researchers, Ramos [7] and Bespamyatnikh [3] achieved deterministic algorithms that run in $O(n \log^2 n)$ time. Finally, Ramos [8] solved the problem in optimal time $O(n \log n)$.

The all-pairs farthest neighbors problem for a set N of n points in three dimensions is to compute, for each point p in N, the point of N farthest from p. This natural generalization of the diameter problem has several applications [1]. While all-pairs *nearest* neighbors in fixed dimension d can be computed in optimal $O(n \log n)$ time [9], no algorithm with similar efficiency is known for the all-pairs *farthest* neighbors. Agarwal et al. [1] showed that 3-dimensional all-pairs farthest neighbors can be computed in $O(n^{4/3} \log^{4/3} n)$ time, and pose closing the gap between this and the only lower bound of $\Omega(n \log n)$ as a challenging open problem.

Progress on this problem was made by Bespamyatnikh [3], who considered the special case where the points form the vertices of a convex polytope. Compare this with the fact that two-dimensional all-pairs farthest neighbors can be computed in linear time if the points are the vertices of a given convex polygon, even though the problem has complexity $\Omega(n \log n)$ for arbitrary points [2]. Bespamyatnikh gaven an $O(n \log^2 n)$ time deterministic algorithm to solve the all-pairs

[*]This research was partially supported by the Hong Kong Research Grants Council. Part of it was done when the first two authors were at HKUST.

[†]Institute of Information and Computing Sciences, Utrecht University, Netherlands. otfried@cs.uu.nl

[‡]Department of Computer Science, KAIST, Korea. cssin@jupiter.kaist.ac.kr

[§]Department of Computer Science, The Hong Kong University of Science and Technology, China. antoine@cs.ust.hk

farthest neighbor problem in this case. The algorithm relies on the fact that the intersection of the 3-dimensional farthest-point Voronoi diagram of a set of n points N with the boundary $\partial\mathcal{P}$ of a convex polytope $\mathcal{P}$ containing N is a linear-complexity subdivision of $\partial\mathcal{P}$. More precisely, this subdivision, called the *restricted Voronoi diagram*, has a linear number of simply-connected faces, vertices, and "edges," where each "edge" is in fact a polygonal chain on $\partial\mathcal{P}$.

However, Bespamyatnikh's algorithm does not actually compute the restricted Voronoi diagram. Instead, it reduces the all-pairs farthest neighbor problem to several instances of the bi-chromatic farthest neighbor problem (defined below), which are then solved by divide-and-conquer.

We show how to compute the restricted Voronoi diagram with a relatively simple randomized incremental algorithm. Given a convex polytope $\mathcal{P}$ with m vertices, and n sites lying on its surface, we compute an implicit representation of the intersection of the farthest-point Voronoi diagram of these sites with the surface of $\mathcal{P}$ in expected $O(m \log m + n \log n \log m)$ time. We also obtain a point-location data structure for the diagram, which allows us to perform farthest-neighbor queries for points on the boundary of $\mathcal{P}$ with query time $O(\log^2 n)$.

As a direct application of this data structure we can compute all-pairs farthest neighbors for n points in convex position in expected time $O(n \log^2 n)$, using a practical algorithm that we believe to be simpler than Bespamyatnikh's.

All-pairs farthest neighbors have a number of interesting applications. We cite a few applications that follow directly from Agarwal et al.'s results [1]:

- *Bi-chromatic farthest neighbors*: Given a set R of n "red" points and another set B of m "blue" points in 3 dimensions such that $R \cup B$ is in convex position, we find for each red point $r \in R$ the farthest blue point from r in expected $O((n + m) \log^2(n + m))$ time.

- *External farthest neighbors*: Given a set N of n points in 3 dimensions in convex position and its partition $N_1, N_2, \ldots, N_m$ into m subsets, we compute in expected $O(n \log^3 n)$ time for each point p in N, a farthest point in $N \backslash N_i$, where $p \in N_i$.

- *Euclidean maximum spanning tree*: Given a set N of n points in 3 dimensions in convex position, we compute in expected $O(n \log^4 n)$ time a spanning tree of N whose edges have the maximum total length among all spanning trees, where the length of an edge is the Euclidean distance between its endpoints. From this tree we can compute a minimum diameter 2-clustering of N in linear time.

2 Preliminaries

Given a set N of points *sites* in R^3 and a point site s not necessarily in N, we define the (farthest-point) *Voronoi cell* $\mathrm{Vor}(s|N)$ of s with respect to N as the set of points $x \in R^3$ such that the Euclidean distance $d(x, s)$ is larger than the distance $d(x, s')$ to any site $s' \in N$ with $s' \neq s$. Voronoi cells are convex, and may be empty. The (farthest-point) *Voronoi diagram* of N is the partition of R^3 into the Voronoi cells $\mathrm{Vor}(s|N)$, for $s \in N$.

Let now $\mathcal{P}$ be the *boundary* of a convex polytope in three dimensions. Let N be a set of point sites lying on $\mathcal{P}$, and s a site on $\mathcal{P}$ not necessarily in N. The Voronoi cell $\mathrm{Vor}(s|N)$ intersects $\mathcal{P}$ in a two-dimensional, possibly empty, *Voronoi face* $\mathrm{Vor}_{\mathcal{P}}(s|N)$. Bespamyatnikh [3] observed that Voronoi faces are simply connected. We include a proof for completeness.

Lemma 1 ([3]) *Let $\mathcal{P}$ be the boundary of a 3-dimensional polytope, N a set of point sites on $\mathcal{P}$, and $s \in N$. The Voronoi face $\mathrm{Vor}_{\mathcal{P}}(s|N)$ is simply connected, that is, its boundary is a simple closed curve.*

Proof: Let p, q be two points in $\mathrm{Vor}_{\mathcal{P}}(s|N)$. Let $C_s(pq)$ be the two-dimensional cone with apex s spanned by pq, and let $L_s(pq)$ be the intersection $C_s(pq) \cap \mathcal{P}$. $L_s(pq)$ is a path on $\mathcal{P}$ connecting p and q. We prove that $L_s(pq)$ lies entirely in $\mathrm{Vor}_{\mathcal{P}}(s|N)$.

In fact, let $x \in pq$. Since $x \in \mathrm{Vor}(s|N)$, N lies in the sphere S with center x and passing through s. If we enlarge S by moving its center along the ray sx and keeping s on the sphere, N will remain inside the enlarged sphere. It follows that the entire portion of $C_s(pq)$ not in the triangle spq is contained in $\mathrm{Vor}(s|N)$. Since $L_s(pq)$ lies in this portion, we have $L_s(pq) \subset \mathrm{Vor}_{\mathcal{P}}(s|N)$.

Moreover, if we centrally project $L_s(pq)$ from s, the result is a line segment. Since for any two points p, q in $\mathrm{Vor}_{\mathcal{P}}(s|N)$ we have $L_s(pq) \subset \mathrm{Vor}_{\mathcal{P}}(s|N)$, this implies that the central projection of $\mathrm{Vor}_{\mathcal{P}}(s|N)$ from s is convex, and therefore simply connected. $\square$

It follows that a set of sites N on $\mathcal{P}$ partitions $\mathcal{P}$ into simply connected faces, defining a planar graph that we denote as $G(N) = G_{\mathcal{P}}(N)$. A face of $G(N)$ is a Voronoi face, a vertex of $G(N)$ is a point of equal distance from three sites, and therefore the intersection of an edge of the three-dimensional Voronoi diagram with $\mathcal{P}$. There can be at most two vertices defined by the same three sites (and this case can indeed arise). An edge of $G(N)$ separates two Voronoi faces $\mathrm{Vor}_{\mathcal{P}}(s|N)$ and $\mathrm{Vor}_{\mathcal{P}}(s'|N)$, and therefore lies on the bisecting plane of the sites s and s'.

Theorem 1 ([3]) *Let N be a set of n sites on a polytope $\mathcal{P}$. Then $G(N) = G_{\mathcal{P}}(N)$ has $O(n)$ vertices, edges, and faces.*

Proof: It follows from Lemma 1 that the number of faces is at most n. A vertex has degree at least three. Euler's formula now implies the linear bound on the number of vertices and edges. $\square$

The embedding of an edge e of $G(N)$ in $\mathcal{P}$ is a polyline whose vertices are the intersections between the embedding of e and the edges of $\mathcal{P}$. If $\mathcal{P}$ has m edges, the embedding of e consists of at most m segments. The overall complexity of the embedding of $G(N)$ is therefore $O(nm)$. This bound is tight, as the example of the modified n-Camembert in Fig. 1 shows.

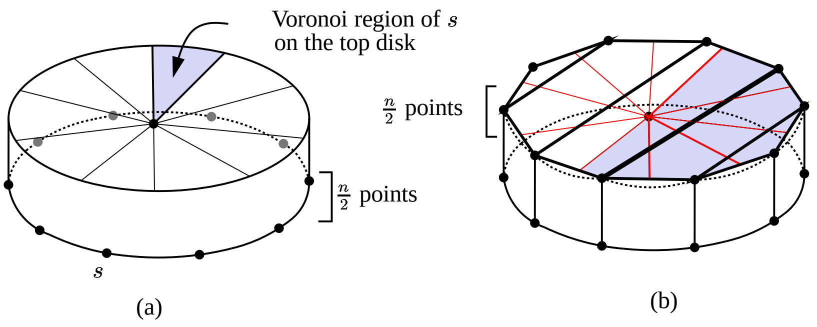


Figure 1: An example of $G(N)$ with complexity of $\Omega(n^2)$. $\mathcal{P}$ is the convex hull of N. (a) If one puts $n/2$ points equidistantly on the boundary of the bottom disk, then their Voronoi cells partition the top disk equally. (b) Add a convex roof slightly above the top disk which consists of the remaining $n/2$ points. The number of intersections of the Voronoi cells of the points on the bottom disk with edges of the roof becomes $\Omega(n^2)$. The fat edge of the roof intersects the shaded Voronoi cells.

To achieve subquadratic time, we cannot work with the explicit embedding of $G(N)$ into $\mathcal{P}$. Instead, we will use a linear size representation of $G(N)$. The representation stores the adjacency relations between vertices, edges, and faces of $G(N)$. By Theorem 1 this has linear complexity. In addition, we record for each face the defining site, and for each vertex the coordinates of its embedding into $\mathcal{P}$.

We need the following lemma.

Lemma 2 *Let N be a set of sites on a polytope $\mathcal{P}$, and let v be a vertex of $G(N) = G_{\mathcal{P}}(N)$. If the sites defining the faces around v are known in the correct order, then the coordinates of v's embedding into $\mathcal{P}$ can be computed using a single ray shooting query on $\mathcal{P}$.*

Proof: The sites define a line of equal distance, which intersects $\mathcal{P}$ in two points. The two points differ in the order of sites. The order of sites thus orients the line, and we can find the correct intersection point with $\mathcal{P}$ using a ray shooting query from infinity. $\square$

As a final observation, note that this approach to computing the farthest-point Voronoi diagram would fail if we tried to apply it to the *nearest-point* Voronoi diagram instead. In fact, the intersection of the nearest-point Voronoi diagram of n sites N with the boundary of the convex hull of N can have a quadratic number of faces; such an example can be easily designed.

In the following we will assume N to be in *general position*, which means that no five sites lie on a sphere and no four points lie on a circle.

3 The strategy to compute $G(N)$

Given a convex polytope $\mathcal{P}$ with m vertices and a set N of n point sites on the surface $\mathcal{P}$, we show how to compute $G(N) = G_{\mathcal{P}}(N)$ in expected time $O(m \log m + n \log n \log m)$.

The first step of the algorithm is to compute, in $O(m \log m)$ time, a data structure that permits ray shooting queries in $\mathcal{P}$ with query time $O(\log m)$.

We then compute $G(N)$ by randomized incremental construction. We choose a random permutation $s_1, s_2, s_3, \ldots, s_n$ of N, and insert the sites in this order.

Let $N^i = \{s_1, s_2, ...s_i\}$ be the set of the first i inserted sites. The algorithm maintains the implicit representation of $G(N^i)$ while sites are added one by one, resulting in $G(N^n) = G(N)$. Note that the polytope $\mathcal{P}$ defining $G(N^i)$ does not change during the course of the algorithm.

Our algorithm is mostly a straightforward implementation of the randomized incremental paradigm using a conflict graph, and most of the lemmas below are analogous to those proven, say, in Mulmuley's book [5]. We do, however, need to cope with an unpleasant aspect of our diagram concerning the maintenance of the conflict graph. As we will see, conflicts can "jump" to another edge. We need $O(\log m)$ time to check a conflict between a site and an edge, which results in overall $O(n \log n \log m)$ expected time for the computation of $G(N)$.

4 Conflicts

A vertex v of $G(N^i)$ is said to be in *conflict* with a site $s \in N \setminus N^i$ if s is farther from v than any of the sites that define v, that is, the sites whose faces are adjacent to v. This is equivalent to $v \in \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. Similarly, an edge e of $G(N^i)$ is said to be in *conflict* with a site $s \in N \setminus N^i$ if $e \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i) \neq \emptyset$.

In addition to a representation of $G(N^i)$, our algorithm maintains a conflict list: for each not-yet-inserted site $s \in N \setminus N^i$, we keep a bidirectional pointer to a single vertex $X(s)$ of $G(N^i)$ in conflict with s. If no vertex of $G(N^i)$ is in conflict with s, we set $X(s)$ to a single edge of $G(N^i)$ in conflict with s. If no edge of $G(N^i)$ is in conflict with s either, then $X(s) := \emptyset$.

Lemma 3 *Let $s \in N \setminus N^i$. If $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ is not empty, then the vertices and edges of $G(N^i)$ in conflict with s form a connected subgraph of $G(N^i)$.*

Proof: Suppose the vertices and edges are not connected. Then we can separate them using a curve γ contained in $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ that cuts $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ into two non-empty components without intersecting vertices or edges of $G(N^i)$. This means that γ is entirely contained in a face $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$

of $G(N^i)$. Since $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$ is simply connected by Lemma 1, γ cannot be a closed curve. The endpoints of γ lie in $\mathrm{Vor}_{\mathcal{P}}(s'|N^i) \setminus \mathrm{Vor}_{\mathcal{P}}(s|N^i) = \mathrm{Vor}_{\mathcal{P}}(s'|N^i \cup \{s\})$. Since this set is connected by Lemma 1, there is a path γ' connecting the endpoints of γ through $\mathrm{Vor}_{\mathcal{P}}(s'|N^i) \setminus \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. The concatenation of γ and γ' is a closed curve contained in $\mathrm{Vor}_{\mathcal{P}}(s'|N^i)$. It separates the edges and vertices of $G(N^i)$, a contradiction to Lemma 1. ☐

We consider now the insertion of the $i+1$'st site s_{i+1} into the data structure storing $G(N^i)$.

Lemma 4 *During the insertion of s_{i+1}, one of the three following situations occurs (see Fig. 3):*

(i) *If s_{i+1} has no conflicting vertex and no conflicting edge in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) = \emptyset$ and $G(N^{i+1}) = G(N^i)$.*

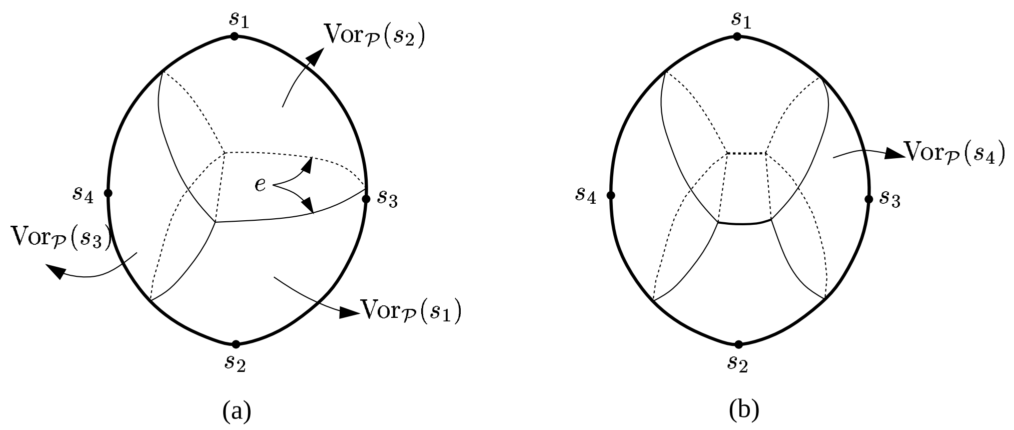
(ii) *If s_{i+1} has only one conflicting edge e and no conflicting vertex in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap G(N^i)$ is a connected portion pq of e.*

(iii) *If s_{i+1} has at least one conflicting vertex in $G(N^i)$, then at least one endpoint of each edge in conflict with s_{i+1} is also in conflict with s_{i+1}.*

Proof: If s_{i+1} has no conflicting vertex and no conflicting edge in $G(N^i)$, then $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ must be empty; otherwise, $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ would lie entirely within some face of $G(N^i)$, which is impossible since the face is simply connected in $G(N^{i+1})$. Thus if $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$, then there must be a vertex or an edge of $G(N^i)$ in conflict with s_{i+1}.

Suppose now that an edge of $G(N^i)$ conflicts with s_{i+1}, but that none of its endpoints conflicts with s_{i+1}. Then Lemma 3 directly implies that no other vertex or edge of $G(N^i)$ is in conflict with s_{i+1}.

The third case trivially holds because the vertices and edges in conflict with s_{i+1} form a connected subgraph of $G(N^i)$ by Lemma 3. ☐



s_1
$\mathrm{Vor}_{\mathcal{P}}(s_2)$
e
s_4
s_3
$\mathrm{Vor}_{\mathcal{P}}(s_3)$
$\mathrm{Vor}_{\mathcal{P}}(s_1)$
s_2
(a)
s_1
$\mathrm{Vor}_{\mathcal{P}}(s_4)$
s_4
s_3
s_2
(b)

Figure 2: (a) $G(N^3)$. No vertex of $G(N^3)$ is in conflict with s_4, but edge e conflicts with s_4. (b) $G(N^4)$.

The algorithm implements the three cases of Lemma 4. Consider the insertion of s_{i+1} into the data structure for $G(N^i)$. We first follow the conflict pointer of s_{i+1} to find the conflict $X(s_{i+1}) \in G(N^i)$ in constant time.

If $X(s_{i+1}) = \emptyset$, we have case (i) of the lemma, and nothing needs to be done.

If $X(s_{i+1})$ is an edge, no vertex of $G(N^i)$ is in conflict with s_{i+1}. We have therefore case (ii) of the lemma, and there is no other conflict of s_{i+1} at all. We can update our data structure to represent

$G(N^{i+1})$ by removing a portion of e and replacing it with an eye-like subgraph that consists of two edges induced by the two bisector planes between s_{i+1} and each of the two sites defining e in N^i. See Fig. 3. Updating the adjacency relations takes constant time. However, we also need to compute the coordinates of the two new vertices. By Lemma 2, this can be done by two ray shooting queries on $\mathcal{P}$ in time $O(\log m)$.

If $X(s_{i+1})$ is a vertex, we have case (iii) of the lemma. By Lemma 3, the portion of $G(N^i)$ lying inside $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is a connected subgraph G of $G(N^i)$. We identify G by traversing $G(N^i)$. This takes time linear in the size of G, as we only need to test conflicts between vertices and s_{i+1}, which takes constant time per test. The extremal edges of G lie on the boundary of the new face $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. We shorten these edges by creating new vertices. After generating the new boundary by connecting these vertices, we finally delete G. If the complexity of G is k, all this can be done in time $O(k \log m)$ using Lemma 2. See Fig. 3.

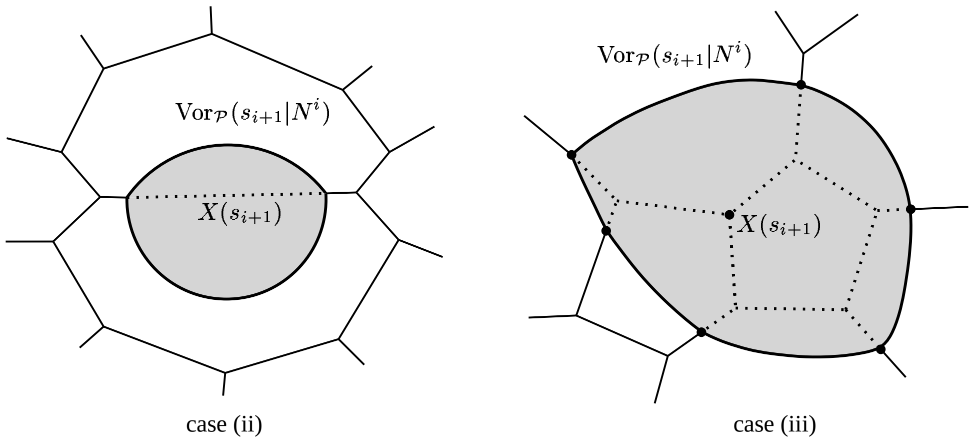

$\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$
$X(s_{i+1})$
case (ii)
$\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$
$X(s_{i+1})$
case (iii)

Figure 3: Illustrating two cases of Lemma 4.

5 Conflict maintenance

It remains to see how we update the conflict list during the insertion of a new site s_{i+1}. Recall that we maintain for every site $s \in N \setminus N^i$ a bidirectional pointer to a vertex or edge of $G(N^i)$ in conflict with s, if there is one. More precisely, if there are vertices of $G(N^i)$ in conflict with s, then $X(s)$ is one of them. If no vertex of $G(N^i)$ is in conflict with s and $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \neq \emptyset$, then there must be an edge $e \subset G(N^i)$ in conflict with s by Lemma 4 (ii). In this case, we set $X(s) = e$. Notice that if $X(s)$ is an edge, then no vertex in $G(N^i)$ conflicts with s (see Lemma 4 (ii)).

When inserting s_{i+1}, a new face $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is defined. If $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$, then the vertices and edges in conflict with s_{i+1} will be destroyed in $G(N^{i+1})$. These are exactly the vertices and edges of $G(N^i)$ that intersect $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$.

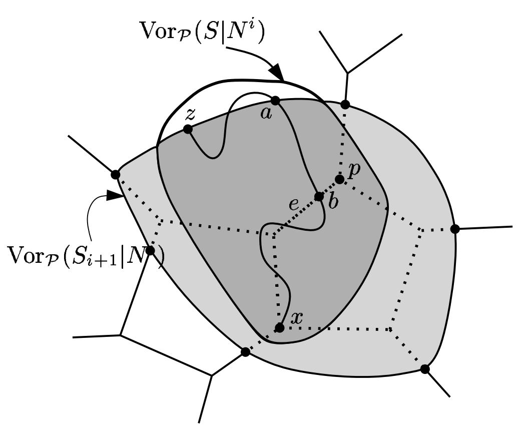
Let s be a non-inserted site in $N \setminus N^{i+1}$. If $X(s)$ intersects $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then $X(s)$ is not defined in $G(N^{i+1})$ any more, and we need to update $X(s)$ by finding a vertex or edge of $G(N^{i+1})$ in conflict with s. Otherwise, namely if $X(S)$ does not intersect $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then we do not need to update $X(s)$. The next lemmas will be used for such conflict update. The boundary of a set R is denoted by ∂R.

Lemma 5 *Suppose that* $s \in N \setminus N^{i+1}$ *and* $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$. *If* $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ *then* $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1}) = \emptyset$. *Otherwise* $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$.

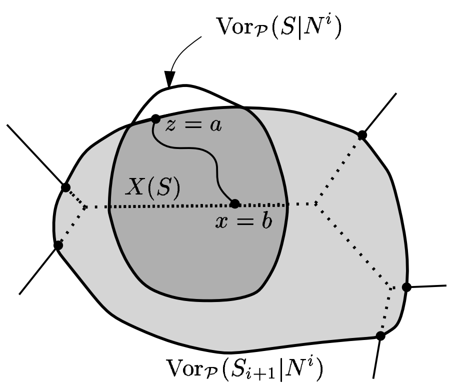
Proof: If $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \subset \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, then all the vertices and edges of $G(N^i)$ lying inside $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ would be destroyed at the end of step $i+1$. It is equivalent to say that no vertex and no edge is in conflict with s over $G(N^{i+1})$. By Lemma 4 (i) it follows that $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$ is empty.

Let now $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Since $x \in \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, there is some $s' \in N^i$ such that $d(x, s_{i+1}) = d(x, s')$. However, since $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^i)$, we have $d(x,s) > d(x,s')$, and so $d(x,s) > d(x, s_{i+1})$. This implies $x \in \mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$. $\square$

A point p is said to be *visible* from a point q within $G(N^i)$ if there is a path connecting q to p whose interior does not intersect any vertex or edge of $G(N^i)$. Similarly, an edge e is said to be *visible* from a point q within $G(N^i)$ if there is a point on e that is visible from q.


$\mathrm{Vor}_{\mathcal{P}}(S|N^i)$
z
a
p
e
b
$\mathrm{Vor}_{\mathcal{P}}(S_{i+1}|N^i)$
x

(a)


$\mathrm{Vor}_{\mathcal{P}}(S|N^i)$
$z = a$
$X(S)$
$x = b$
$\mathrm{Vor}_{\mathcal{P}}(S_{i+1}|N^i)$

(b)

Figure 4: The proof of Lemma 6 (ii). (a) $X(s)$ is a vertex. (b) $X(s)$ is a portion of an edge. In this example, $z = a$ and $x = b$.

Lemma 6 *Suppose that s is a non-inserted site in $N \setminus N^{i+1}$ and $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \neq \emptyset$. If $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1}) \neq \emptyset$, then one of these two cases must occur:*

(i) *s has a conflicting vertex $v \in \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ in $G(N^{i+1})$.*

(ii) *S has no conflicting vertex in $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, but has a single conflicting edge $e \in \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ such that e is visible from a point $p \in G(N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i)$ within $G(N^i)$.*

Proof: By Lemma 5, each point of $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ is in conflict with s. Define $C = \mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Then C is a connected chain due to Lemma 1. If C contains a vertex of $G(N^{i+1}$, case (i) is true.

If C is a portion of an edge e of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, we need to prove that e is visible from some point $p \in G(N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s|N^i)$. For the following description, see Fig. 4.

Let z be a point of C. Let x be a point of $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Note here that if $X(s)$ is a vertex, then $x = X(s)$; if $X(s)$ is an edge, x is any point of $X(s)$ which belongs to $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. Consider an arbitrary simple path γ connecting x to z within $\mathrm{Vor}_{\mathcal{P}}(s|N^i)$ (not within $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$). Since x lies in $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ and z does not lie in the interior of $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$, γ must intersect $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ at least once, possibly in z. We denote by a the first such intersection on the way from x to z. Let e be the edge of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ containing a.

Let b be the last intersection of $G(N^i)$ with γ on the way from x to a. If $b = x$, then it means e is an edge bounding some region in $G(N^i)$ incident to x, so e is visible from x, that is, $p = x$. If

$b \neq x$, Lemma 4 (iii) implies that an endpoint p of the edge of $G(N^i)$ containing b conflicts with s. It means that a is visible from p, so e is visible from p, and e intersects γ. Hence e conflicts with s in $G(N^{i+1})$. $\square$

Lemma 7 *For a site $s \in N \setminus N^i$ and a given vertex or edge of $G(N^i)$, we can decide in $O(\log m)$ time whether they are in conflict.*

Proof: A vertex $w \in G(N^i)$ is in conflict with s if and only if w is farther from s than the three sites in N^i that define w. This can be checked in constant time because we have access to the coordinates of w and the sites defining the adjacent faces.

Suppose now that we want to check whether an edge e is in conflict with a site s. Let $s', s'' \in N^i$ be the sites defining the faces adjacent to e, and let π be the bisecting plane of s', s''. The edge e is embedded in π, and is in conflict with s if and only if it intersects the halfplane $\pi' := \{x \in \pi \mid d(x,s) > d(x,s')\}$. We test this by ray shooting on $\mathcal{P}$ in time $O(\log m)$. $\square$

We are now ready to describe how the conflict list is updated during the insertion of a new site s_{i+1}. We first collect all not-yet-inserted sites s whose conflict is destroyed by the insertion of s_{i+1}. This can be done during the exploration of the subgraph. Since $X(s)$ is being deleted in $G(N^{i+1})$, we need to either find a new conflicting object in $G(N^{i+1})$, or to find out that there is none, which means that $\mathrm{Vor}_{\mathcal{P}}(s|N^j) = \emptyset$ for all $j \geq i+1$ by Lemma 5.

Suppose that $\mathrm{Vor}_{\mathcal{P}}(s|N^{i+1})$ is not empty. By Lemma 6, we can find a new vertex or edge in conflict with s by exploring the subgraph of $G(N^i)$ belonging to $\mathrm{Vor}_{\mathcal{P}}(s|N^i) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ as follows. If $X(s)$ is an edge, it may have a conflict with a new vertex of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ that is defined in the interior of $X(s) \cap \partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ or a new edge that is visible from any point in $X(s) \cap \mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. In the latter case, there can be at most two such edges, thus we can check if the edges conflict with s. If they do not, then s can be discarded since its Voronoi region is empty.

On the other hand, if $X(s)$ is a vertex, then we first look for a new conflicting vertex by walking among the vertices of $G(N^i)$ in conflict with s until we reach a vertex of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. If we do not find such a vertex, we still have to check if there is a new conflicting edge of $\partial\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$ which is visible from some vertex visited so far (see Lemma 6 (ii)). If we fail to find a conflicting edge, then s can be discarded.

6 Analysis

Two of our primitive operations—creating a Voronoi vertex and detecting conflict with an edge—require ray shooting query on $\mathcal{P}$, and therefore take $O(\log m)$ time. All other primitive operations take constant time.

We will show that, after the initial preprocessing in time $O(m \log m)$, our algorithm performs an expected number of $O(n \log n)$ primitive operations. This implies a running time of $O(m \log m + n \log n \log m)$.

The cost of updating the Voronoi diagram is proportional to the number of created edges and vertices plus the number of destroyed edges and vertices. Since each of them can only be created and destroyed once, the amortized cost of this operation is just the number of created objects, which is proportional to the number of edges of $\mathrm{Vor}_{\mathcal{P}}(s_{i+1}|N^i)$. This quantity will be denoted by $m(s_{i+1}, N^{i+1})$. We proceed by backward analysis. Consider that N^{i+1} is fixed and the last inserted site s is chosen at random. The expected update cost is proportional to

$$\frac{1}{i+1} \sum_{s \in N^{i+1}} m(s, N^{i+1})$$

Since an edge is adjacent to exactly two faces, the sum adds up to twice the number of edges. By Theorem 1, this is $O(i)$, and so the expected update time of the Voronoi diagram is $O(1)$.

Let's now analyze the cost of maintaining the conflict list. Suppose we reach a vertex (a Voronoi edge or vertex) $X(s)$ for some s while walking within the subgraph of the vertices in conflict with s_{i+1}. If $X(s)$ is an edge, then we only need to check at most two edges of the boundary that are visible from $X(s)$. If $X(s)$ is a vertex, we first walk among vertices that are conflicting both s and s_{i+1}, then we check the boundary edges that have been visible during this walk, which only adds a constant factor. By the same amortization argument as above, it suffices to count the total number of conflicts created in each stage.

We will proceed by backwards analysis. Let $c(s, N^i)$ denote the number of edges of $G(N^i)$ in conflict with s. Assume that N^i is fixed and s_i is taken randomly. The number of conflicts created during the insertion of s_i is, in expectation:

$$\frac{2}{i} \sum_{s \in N \setminus N^i} c(s, N^i) = \frac{2(n-i)}{i} E[c(s_{i+1}, N^i)].$$

Note that $E[c(s_{i+1}, N^i)]$ is simply the number of edges destroyed at step i, so the above quantity summed over all values of i is smaller than:

$$\sum_{i=1}^{n-1} \frac{2}{i} E[m(s_{i+1}, N^{i+1})] = \sum_{i=1}^{n-1} \frac{2}{i} O(1) = O(n \log n).$$

Theorem 2 *Given a polytope $\mathcal{P}$ with m vertices, and n points N on the surface of $\mathcal{P}$, we can compute $G(N) = G_{\mathcal{P}}(N)$ in expected time $O(m \log m + n \log n \log m)$.*

7 Point location queries and farthest neighbors

Once we have computed the implicit representation of the farthest-point Voronoi diagram of N on $\mathcal{P}$, we can easily find a point in the interior of each face $\mathrm{Vor}_{\mathcal{P}}(s|N)$. For example, take two non-adjacent vertices in an ordinary face, or the two vertices of an eye, and shoot a ray from s towards their midpoint. Then, run the same algorithm again, but now computing a radial triangulation of $G(N)$ (see pp. 109 in [5]). Each of its faces is the intersection of $\mathcal{P}$ with a cone (ss', sv, sv') where $s \in N$, s' is the interior point for $\mathrm{Vor}_{\mathcal{P}}(s|N)$, and v, v' are consecutive vertices of $\mathrm{Vor}_{\mathcal{P}}(s|N)$. During the second pass, we maintain for each site, whether it has been inserted or not, a pointer to the face of the radial triangulation that contains it. One can see easily that this does not hurt our time bound. It allows to find for each site the face of $G(N)$ it belongs to. Still following [5] we can build a point location data structure for our radial triangulation that answer queries in $O(\log^2 n)$ time with high probability. The reason why we need the radial triangulation for these two results is a configuration space argument, namely we want the faces of our graph to have bounded degree.

Theorem 3 *Given a set N of points in 3-dimensional convex positions, all-pairs farthest neighbors can be computed in expected $O(n \log^2 n)$ time. A data structure that answers farthest-neighbor queries in $O(log^2 n)$ time with high probablility can be build within the same time bounds.*

References

[1] P. K. Agarwal, J. Matoušek, and S. Suri. Farthest neighbors, maximum spanning trees and related problems in higher dimensions. *Comput. Geom. Theory Appl.*, 1(4):189–201, 1992.

[2] A. Aggarwal and D. Kravets. A linear time algorithm for finding all farthest neighbors in a convex polygon. *Information Processing letters*, 31(1):17–20, 1989.

[3] S. N. Bespamyatnikh. An efficient algorithm for the three-dimensional diameter problem. *Discrete Comput. Geom.*, 25:235–255, 2001.

[4] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, 4:387–421, 1989.

[5] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice Hall, Englewood Cliffs, NJ, 1994.

[6] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, New York, NY, 1985.

[7] E. Ramos. Construction of 1-d lower envelopes and applications. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pp. 57–66, 1997.

[8] E. Ramos. An optimal deterministic algorithm for computing the diameter of a 3-d point set. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, page to appear, 2000.

[9] P. M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete Comput. Geom.*, 4:101–115, 1989.