Discrete Comput Geom 25:235–255 (2001)
DOI: 10.1007/s004540010086

Discrete & Computational
Geometry
© 2001 Springer-Verlag New York Inc.

An Efficient Algorithm for the Three-Dimensional Diameter Problem

S. Bespamyatnikh

Department of Computer Science, University of British Columbia,
Vancouver, British Columbia, Canada V6T 1Z2
besp@cs.ubc.ca

Abstract. We explore a new approach for computing the diameter of n points in $\mathbb{R}^3$ that is based on the restriction of the furthest-point Voronoi diagram to the convex hull. We show that the restricted Voronoi diagram has linear complexity. We present a deterministic algorithm with $O(n \log^2 n)$ running time. The algorithm is quite simple and is a good candidate to be implemented in practice. Using our approach the chromatic diameter and all-furthest neighbors in $\mathbb{R}^3$ can be found in the same running time.

1. Introduction

We address the well-known diameter problem:

The Diameter Problem. Let S be a set of n points in d-dimensional space. Compute the diameter of S, defined as the maximum distance between two points of S.

Let *diam*(S) denote the diameter of set S. In this paper we consider the diameter problem in $\mathbb{R}^3$. This problem was solved by Clarkson and Shor [12] by a randomized algorithm with optimal expected running time $O(n \log n)$. The diameter can be deterministically computed in $O(n^2)$ time by brute force, by simply comparing all the distances. Yao [30] solved this problem in $O((n \log n)^{1.8})$ time (in higher dimensions $d \geq 4$ he obtained $O(n^{2-a(d)} \log^{1-a(d)} n)$ time, where $a(d) = 2^{-(d+1)}$).

By computing a structure that allows point location in the (implicitly represented) furthest-point Voronoi diagram of S, it is possible to achieve running time $O(n^{4/3+\varepsilon})$ [20].

The further results in [3], [8], [11], [21], and [25] are based on the same approach. Clarkson and Shor [12] first transformed the three-dimensional diameter problem to the following problem.

The Ball Problem. Given n unit-balls and n points in $\mathbb{R}^3$, determine whether any point lies outside the common intersection of the balls.

The diameter of a set S is the smallest r such that the intersection of the balls of radius r centered at the points of S contains S. The algorithms of [11], [21], and [8] use Megiddo's parametric search technique [22]. For a specific value r, an oracle O decides whether $r < diam(S)$, $r = diam(S)$ or $r > diam(S)$ using an algorithm for the ball problem.

Chazelle et al. [11] obtained a deterministic $O(n^{1+\varepsilon})$ algorithm, where $\varepsilon > 0$ is an arbitrary small constant. Matoušek and Schwarzkopf [21] improved this running time to $O(n \log^c n)$ where $c > 0$ is a constant. Amato et al. [3], Brönninman et al. [8], and Ramos [25] gave $O(n \log^3 n)$ algorithms for computing the diameter. Ramos [26] improved the running time by a factor of $\log n$, using a one-dimensional lower envelope algorithm, and further insight into geometric optimization in an arrangement of surfaces. Very recently, Ramos [27] obtained a $O(n \log n)$ algorithm using derandomization and ε-nets that hide a constant in the asymptotic running time much larger than ours.

In this paper we study a new approach for computing the diameter by constructing a restriction of the furthest-point Voronoi diagram to the convex hull (in a special case). We present an algorithm with running time $O(n \log^2 n)$. Our algorithm uses practical data structures such as the lists (including the double-connected-edge-lists), the balanced binary search tree, and the hierarchy of Dobkin and Kirkpatrick [14]. The algorithm is quite simple and its performance should be good in practice. Using our approach the *chromatic diameter* in $\mathbb{R}^3$ and *all-furthest neighbors* of points on a convex polytope in $\mathbb{R}^3$ can be found in the same running time.

The Chromatic Diameter. For n colored points in $\mathbb{R}^3$, compute the largest distance between points of different colors.

The All-Furthest Neighbors. Given a set S of n points on a convex polytope in $\mathbb{R}^3$, for each point of S, compute its furthest neighbor in S.

The paper is organized as follows. In Section 2 we obtain basic geometric properties of the restricted furthest-point Voronoi diagram. In Section 3 we reduce the diameter problem of computing the diameter to the computation of the *red–blue diameter*. A red–blue diameter algorithm is described in Section 4. The details of the basic steps of the algorithm are given in Sections 5–7. In Section 8 we analyze the running time of the algorithm. In Section 9 we consider applications that can be solved by our approach. Section 10 contains some concluding remarks and open questions.

2. Geometric Preliminaries

We introduce some notation that is used throughout this paper. Denote the Euclidean distance between two points p and q by $dist(p, q)$. For a finite set $A \subset R^3$, $CH(A)$ denotes the convex hull of A, and $Vor(A)$ denotes the furthest-point Voronoi diagram of

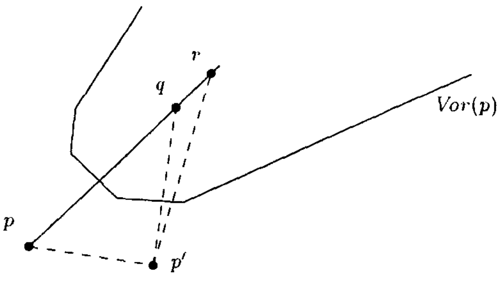

Fig. 1. The Voronoi cell of p contains each point q together with the ray qr.

A. For a point $p \in A$, $Vor(p)$ denotes the Voronoi polytope of p in the Voronoi diagram $Vor(A)$. The boundary of a polytope P is denoted by $bd(P)$.

The following lemmas are useful.

Lemma 1. *Let A be a finite set of points in $\mathbb{R}^d$, let p be a point of A, and let q be a point of the furthest-point Voronoi cell of p. Then the Voronoi cell $Vor(p)$ contains the ray starting at q in the direction from p to q.*

Proof. Let r be a point of the ray pq, see Fig. 1. Suppose that r does not belong to the Voronoi polytope $Vor(p)$. Hence, for a point $p' \in A$, the point $r \in V(p')$ and $dist(p, r) < dist(p', r)$. By the triangle inequality $dist(p', r) \leq dist(p', q) + dist(q, r) \leq dist(p, q) + dist(q, r) = dist(p, r)$. Contradiction. □

Lemma 2. *Let A be a finite set of points in $\mathbb{R}^3$ and let P be a convex polytope containing A. For a point $p \in A$, the intersection of the furthest-point Voronoi cell of p and the boundary of P is a connected set.*

Proof. Let q and r be any distinct points of the intersection of the Voronoi cell of p and the boundary of the polytope P, i.e. $q, r \in Vor(p) \cap bd(P)$. We can assume that the set A contains at least two points. Hence $p \notin Vor(p)$ and $p \neq q$ and $p \neq r$. Let π be the plane passing through the points p, q and r, see Fig. 2. The boundary of the

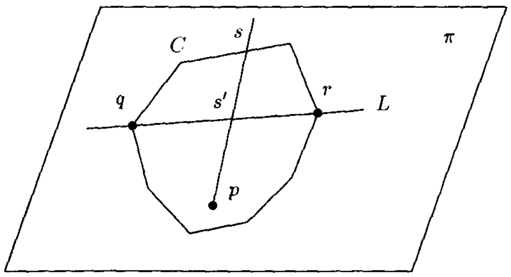

Fig. 2. Connectivity of $Vor(p) \cap bd(P)$.

convex polygon $Q = \pi \cap P$ contains q and r. Let L be the line passing through q and r. The points q and r divide the boundary of Q into two polygonal chains. Let C be the polygonal chain such that L separates p and C.

It is sufficient to show that the Voronoi cell of p contains the polygonal chain C. Consider any point $s \in C$. The furthest-point Voronoi cell of p is a convex set. Hence it contains the line segment $[q, r]$ and the point s' which is the intersection of the line segments $[q, r]$ and $[p, s]$. The Voronoi cell of p contains s by Lemma 1. □

Lemma 3. *Let S be the set of vertices of a convex polytope in $\mathbb{R}^3$. The furthest-point Voronoi diagram of S satisfies the following properties*:

(1) *For a point $p \in S$, the Voronoi cell of p is nonempty.*
(2) *For a point $p \in S$ and a plane α, $O(1)$ operations suffice to determine a side of α such that the corresponding halfspace and the Voronoi cell of p intersect (assuming that the list of edges/faces incident to p can be accessed in $O(1)$ time).*

Proof. (1) Let β be a support plane β touching the convex hull of S only at the point p, i.e. $\beta \cap CH(S) = \{p\}$. Let w be the ray starting at p, perpendicular to β and lying in the same halfspace formed by β as the set S. It is easy to show that the Voronoi cell of p intersects w.

(2) Using the previous case it is sufficient to find a support plane intersecting the convex hull of S only at p to determine a side of the plane α. Let f be a support face of the convex hull of S and let (p, q) and (p, r) be the consecutive edges of f. By a slight perturbation of the plane containing f (it suffices to rotate it around lines pq and pr) we can find a plane β such that

- β is still a support plane of the convex hull of S,
- β intersects the convex hull of S only at the point p, and
- β is not perpendicular to the plane α.

Compute the ray w as defined above. Note that w is not parallel to the plane α. The halfspace bounded by α that contains the tail of w can be determined in constant time. □

Our algorithm is based on the following theorem.

Theorem 4. *Let A be a finite set of points in $\mathbb{R}^3$ and let P be a convex polytope containing A. The restriction of the furthest-point Voronoi diagram of A to the boundary of P has linear size. In other words, the number of vertices, edges and faces of $Vor(A)$ intersecting the boundary of P is $O(|A|)$.*

Proof. By Lemma 2 the boundary of the polytope P is partitioned into at most $|A|$ connected areas with polygonal boundaries. We can map $bd(P)$ into a plane to obtain a planar subdivision with complexity $O(|A|)$. Note that an "edge" of the subdivision is a polygonal chain that separates two polygons and contributes 1 to the complexity. □

3. Reduction

First we reduce the problem of finding the diameter of a set to the problem of finding the furthest distance between points of two sets separated by a plane.

The Red–Blue Diameter Problem. Let R be a set of red points and let B be a set of blue points that are separated by a plane π. Find the *red–blue diameter* of R and B, defined as the largest distance between red and blue points.

Let $rb\_diam(R, B)$ denote the red–blue diameter of sets R and B. The reduction of the diameter problem to the red–blue diameter problem can be done easily. Compute the minimum axes-parallel box that contains data points. Construct three planes perpendicular to all coordinate axes such that each plane divides the covering box into two boxes of equal size. Consider one plane. It divides set S into two subsets. Solve the red–blue diameter problem for these sets. The diameter of S is the maximum of these diameters (the proof is given in Theorem 5). This reduction uses only three red–blue diameter subproblems. The pseudocode is as follows.

```
Algorithm Diam($S$)

Input. A set $S$ of $n$ points in $\mathbb{R}^3$.
Output. The diameter of $S$.

  1. Compute the bounding box $[l_x, r_x] \times [l_y, r_y] \times [l_z, r_z]$ of $S$
  2. Compute planes $\pi_x = \{p \mid p_x = (l_x + r_x)/2\}$, $\pi_y = \{p \mid p_y = (l_y + r_y)/2\}$, $\pi_z = \{p \mid p_z = (l_z + r_z)/2\}$
  3. return max (rb_diam($S \cap \pi_x^+, S \cap \pi_x^-, \pi_x$),
                 rb_diam($S \cap \pi_y^+, S \cap \pi_y^-, \pi_y$),
                 rb_diam($S \cap \pi_z^+, S \cap \pi_z^-, \pi_z$)),
     where rb_diam($R, B, \pi$) is a function that computes the red–blue
     diameter of the sets $R$ and $B$ separated by the plane $\pi$.
```

Theorem 5. *The reduction algorithm is correct.*

Proof. Consider the smallest axes-parallel box A containing S. Let a, b and c denote the lengths of its sides. It is clear that the diameter of S is at least $\max(a, b, c)$. Three planes partition the box A into eight equal boxes with sides $a/2$, $b/2$ and $c/2$. We show that any pair that determine the diameter of S cannot lie in the same subbox. Suppose that there exists a pair p, q of S in the same subbox such that $dist(p, q) = diam(S)$. It is clear that the distance between p and q is at most the length of the diagonal of A. Hence

$$\begin{aligned} diam(S) &= dist(p,q) \le \sqrt{\left(\frac{a}{2}\right)^2 + \left(\frac{b}{2}\right)^2 + \left(\frac{c}{2}\right)^2} \\ &\le \frac{\sqrt{3}}{2}\max(a,b,c) < \max(a,b,c) \le diam(S). \end{aligned}$$

Contradiction.

Consider any diametrical pair p, q of points in S, i.e. $dist(p, q) = diam(S)$. At least one of three planes separates p and q. Therefore the red–blue diameter corresponding to this plane is equal to the diameter of S. Hence our algorithm correctly computes the diameter of S. □

4. Red–Blue Diameter Algorithm

In this section we solve the red–blue diameter problem. Let R be a set of n_R points and let B be a set of n_B points. The plane π separates these sets. Without loss of generality we can assume that $n_R \geq n_B$. Two points $p \in R$ and $q \in B$ forming the red–blue diameter $dist(p, q) = diam(R, B)$ lie on the convex hull of the union of red and blue sets. First we construct the convex hull $H = CH(R \cup B)$ and remove points that lie inside it.

Let π_1 be a plane that

- is parallel to the plane π, and
- divides the set R into two subsets of (almost) equal size.

Let R_1 denote the subset of R which lies in the same halfspace defined by π_1 as the set B. Let R_2 denote another subset of R, i.e. $R_2 = R \backslash R_1$.

Construct the intersection of the furthest-point Voronoi diagram $Vor(B)$ and the plane π_1. The intersection has size $O(n)$. The algorithm for constructing $Vor(B) \cap \pi_1$ is described in Section 5. We also construct a polygon Q that is the intersection of the convex hull H and the plane π_1. It is clear that Q is a convex polygon.

For a point $p \in B$, there are four cases of layout $Vor(p)$, π_1 and Q. In each case we analyze whether $Vor(p)$ might contain points of R_1 and R_2.

Case 1. The Voronoi cell of p does not intersect the plane π_1.

There are two subcases depending on which side of π_1 the Voronoi cell is on. By Lemma 3, the side can be determined in $O(1)$ time.

Case 1.1. The Voronoi cell of p is on the same side of π_1 as the points of B. The Voronoi cell of p does not contain any point of R_2.

Case 1.2. The plane π_1 separates the Voronoi cell of p and the set B. The Voronoi cell of p does not contain any point of R_1.

In the following cases the Voronoi cell of p intersects the plane π_1. Denote this polygon by $P = Vor(p) \cap \pi_1$.

Case 2. The polygon P lies inside the polygon Q (see Fig. 3). We show that the Voronoi cell of p does not contain any point of R_1. Suppose to the contrary that the Voronoi cell of p contains a point $q \in R_1$. Consider a ray emanating from any point $r' \in P$ in the direction from p to r'. By Lemma 1 it is contained in $Vor(p)$. Let r be the point of intersection of the ray and the boundary of H. Clearly, $r \in Vor(p)$. By Lemma 2, there is a path in $Vor(p) \cap bd(H)$ from q to r. This path crosses the boundary of the polygon Q. Hence the Voronoi cell of p (and the polygon P) contains at least one point of the boundary of polygon Q. This contradicts the assumption $P \subset Q$.

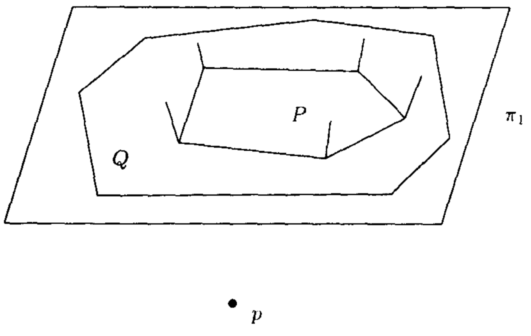

Fig. 3. Case 2: $P \subset Q$.

Case 3. The polygon P lies outside the polygon Q. Construct a line l_1 (see Fig. 4) such that

- l_1 separates the polygons P and Q, and
- l_1 is tangent to the polygon P.

To compute the line l_1 we can use an optimal $O(\log n)$ algorithm of Edelsbrunner [15] for finding the minimum distance between two convex polygons. Consider the tangent point of l_1 and edges of P incident to it. These edges share the Voronoi region of p and the Voronoi regions of two points, say q and r. It is clear that the Voronoi cell of p lies in the halfspace $\{t \mid dist(p,t) \le dist(q,t)\}$ and in the halfspace $\{t \mid dist(p,t) \le dist(r,t)\}$. The intersection of these halfspaces lies in the halfspace defined by a plane passing through the line l_1 and the line $\{t \mid dist(p,t) = dist(q,t) = dist(r,t)\}$. We denote this plane by π_2.

The plane π_2 (and the Voronoi cell of p) cannot intersect both parts of the convex hull H into which it is divided by the plane π_1. We want to determine a part that does not intersect the plane π_2. Find a vertex s of the polygon Q that is closest to the line l_1.

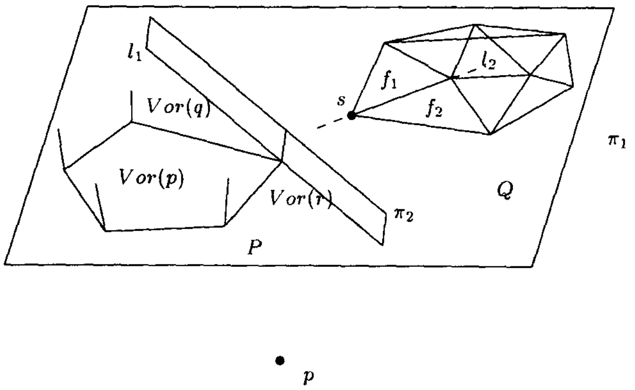

Fig. 4. Case 3: $P \cap Q = \emptyset$.

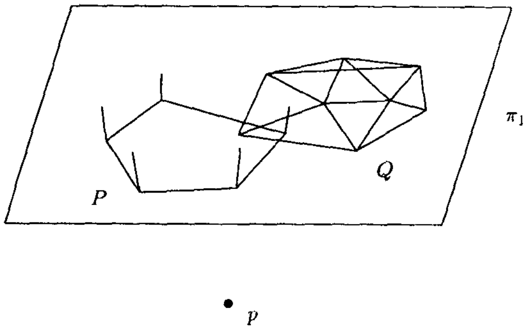

Fig. 5. Case 4: $P \cap Q \neq \emptyset$ and $P \backslash Q \neq \emptyset$.

Let f_1 and f_2 be the facets of the convex hull H that

- are incident to the point s, and
- intersect the plane π_1 (see Fig. 4).

Compute a line l_2 that is an intersection of planes passing through the facets f_1 and f_2. Consider the point of intersection of the line l_2 and the plane π_2. If the plane π_1 separates it and the point p, then the Voronoi cell of p does not contain any point of R_1. If these points lie at the same halfspace defined by π_1, then the Voronoi cell of p does not contain any point of R_2. If the line l_2 is parallel to the plane π_2, then the Voronoi cell of p does not contain any point of R.

Case 4. The polygon P intersects the boundary of the polygon Q (the vertices and the segments) (see Fig. 5). This is the last case. Unfortunately, the point p can be the furthest neighbor of points in both R_1 and R_2.

Thus, we partition the set B into three subsets $B = B_1 \cup B_2 \cup B_3$, where the points of B_1 (B_2) satisfy the property that their Voronoi cells do not contain any point of R_1 (R_2). The set $B_3 = B \backslash B_1 \cup B_2$ contains the points of Case 4. An algorithm to determine the sets B_1, B_2 and B_3 is given in Section 6.

The red–blue diameter *rb_diam*(R, B) is maximum of the red–blue diameters *rb_diam* (R_1, B_2), *rb_diam*(R_2, B_1) and *rb_diam*(R, B_3). The first two diameters are computed recursively. We show how to compute the red–blue diameter of R and B_3 in Section 7.

Algorithm `rb_diam`(R, B, π)

Input. Sets R and B are separated by a plane π, i.e. $R \subset \pi^+$ and $B \subset \pi^-$ (π might be an external variable because it is never changed.

Output. The red–blue diameter of R and B.

```
1. $n_R = |R|$ and $n_B = |B|$
2. if ($n_R < n_B$) then exchange $R$ and $B$
3. $H$ = convex_hull($R \cup B$)
4. compute a plane $\pi_1$ parallel to $\pi$ such that the sets $R_1 = R \cap \pi_1^-$ and
   $R_2 = R \cap \pi_1^+$ have at most $\lceil n_R/2 \rceil$ points.
```

```
5. build $P = H \cap \pi_1$ and the intersection of $\pi_1$ and the furthest-point
   Voronoi diagram of $B$
6. divide $B = B_1 \cup B_2 \cup B_3$ as described above in Cases 1–4
7. return max (rb_diam($R_1, B_2, \pi$),
               rb_diam($R_2, B_1, \pi$),
               rb_diam1($R, B_3, \pi_1$)),
   where rb_diam1($R, B, \pi$) is a function that computes the red–blue
   diameter of the sets $R$ and $B$ in a special case.
```

Lemma 6. *Let $T(N)$ be the running time of* `rb_diam()` *for total N red and blue points. Let $T_1(n, m)$ be the running time of* `rb_diam1()` *for n red and m blue points. Then*

$$T(N) = O(N \log N) + T(N_1) + T(N_2) + T_1(N, N_3)$$

for some $N_1, N_2, N_3 \geq 0$ such that $N = N_1 + N_2 + N_3$ and $N_1, N_2 \leq 3N/4 + 1$.

Proof. Steps 1 and 2 take $O(1)$ time. The convex hull can be built in $O(N \log N)$ time [23]. The plane π_1 in Step 4 can be found in $O(1)$ time using presorting of points. We show that Step 5 can be done in $O(N \log N)$ time in Section 5. In Section 6 we describe how to perform Step 6 in $O(N)$ time.

Two recursive calls at Step 7 add $T(N_1) + T(N_2)$ time to the total running time of the algorithm above, where $N_1 = |R_1| + |B_2|$ and $N_2 = |R_2| + |B_1|$. Note that

$$N_1 = |R_1| + |B_2| \leq \left(\frac{n}{2} + 1\right) + n_B = \frac{3n}{4} - \frac{m}{4} + m + 1 \leq \frac{3n}{4} - \frac{m}{4} + m + 1 = \frac{3N}{4} + 1.$$

Similarly, $N_2 \leq 3N/4 + 1$.

The call of `rb_diam1()` at Step 7 takes at most $T_1(N, N_3)$ time where $N_3 = |B_3| = N - N_1 - N_2$. □

5. Voronoi Diagram in Plane

In this section we consider the problem of constructing the intersection of a plane and the furthest-point Voronoi diagram in $\mathbb{R}^3$. We call it the *clipped* Voronoi diagram for brevity. This problem is related to *power diagrams* or *Laguerre diagrams* [24] or *Dirichlet cell complexes* [4]–[7], [17]. The power diagram is a generalization of the Voronoi diagram. Recall that the closest-point (furthest-point) Voronoi diagram of a set S is defined as the subdivision of the space into Voronoi cells and the Voronoi cell of a point p is defined as the locus of points q such that p is the closest (furthest) point of S to q. The order k Voronoi diagram is defined in similar way, the only difference is that the Voronoi cell of order k is defined by k points of S that are closest to the cell points. The order $(n - 1)$ Voronoi diagram is the furthest-point Voronoi diagram.

In the power diagram of S the distance from a point p in S to a point q is measured as *power*

$$pow(p, q) = d(p, q)^2 - w(p),$$

where $d(p, x)$ is the Euclidean distance between p and q and $w(p) \geq 0$ is a *weight* of a point p. The power $pow(p, q)$ can be considered as the square of the length of the tangent segment from q to the sphere $s = \{r, d(p, r) = \sqrt{w(p)}\}$. In Laguerre geometry a point (x, y, z) in $\mathbb{R}^3$ is made to correspond to a directed circle in the plane $z = 0$ and radius $|z|$, the circle being endowed with the direction of the sign of z. The Laguerre distance between two points corresponds to the length of the common tangent of the corresponding two circles. Hence the power $pow(p, q)$ in the plane $z = 0$ is the square of the Laguerre distance between points $(p_x, p_y, \sqrt{pow(p)})$ and $(q_x, q_y, 0)$ in $\mathbb{R}^3$.

The Voronoi diagram is a special case of the power diagram if all points have the power 0. Similarly, the order k power diagram and the furthest-point power diagram (the *maximal power diagram*) can be defined [5]).

Now we show a relation between the clipped Voronoi diagram CVD and the furthest-point power diagram. We can assume that the clipped plane π is $z = 0$ for simplicity. Consider a point $p = (p_x, p_y, p_z) \in S$. Let $p' = (p_x, p_y, 0)$ be the projection of p onto π. The cell of CVD corresponding to p contains points $q \in \pi$ such that $d(p, q)$ or $d(p', q)^2 + p_z^2$ is smallest. Assigning $w(p) = -p_z^2$ would reduce CVD to the power diagram except for the condition that $w(p)$ must be nonnegative. In order to overcome this shortcoming, we use the fact [17] that the power diagram does not change if we add any number to all powers. Let p_m be the maximum of p_x^2, $p \in S$. Assigning the weight $p_m - p_z^2 \geq 0$ to the point p produces the correct reduction of CVD to the power diagram.

To construct the furthest-point power diagram we can use the $O(n \log n)$ algorithm of Aurenhammer [5]. Aurenhammer reduced the problem to the convex hull problem in $\mathbb{R}^3$ using the *lifting map* technique. Also one can create a direct algorithm using the divide-and-conquer technique like an algorithm of Imai et al. [17] for the closest-point power diagram.

6. Partitioning the Set *B*

Recall that we computed the intersection of the plane π and the furthest-point Voronoi diagram $Vor(B)$. Denote this subdivision of π by $\mathcal{P}$. We also computed the convex polygon Q that is the intersection of π and the convex hull of the red and blue sets. We solve the following problem.

Problem. Given a convex polygon Q and a subdivision of the plane into k convex polygons $\mathcal{P}$ such that the total number of edges in Q and $\mathcal{P}$ is n, subdivide $\mathcal{P}$ into three groups of polygons that

- lie inside Q,
- lie outside Q,
- intersect the boundary of Q.

We restrict the problem to finding the last set $\mathcal{P}'$ of polygons (the remaining groups can be found by traversing the subdivision). We can find the intersection of edges of Q and the edges of polygons of $\mathcal{P}$. The number of intersections is bounded by $O(n)$ (in fact it is $O(k)$). We can apply the algorithm of Bentley and Ottmann [9] and compute the polygons of $\mathcal{P}'$ in $O(n \log n)$ running time.

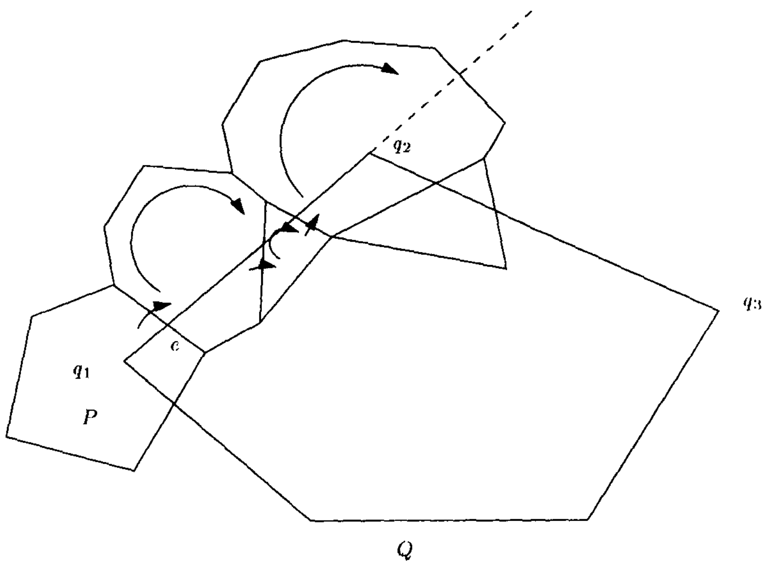

Fig. 6. Walk around boundaries of polygons intersecting the edge q_1, q_2.

We present a simple way to find $\mathcal{P}'$ in linear time. Denote the vertices of Q by $q_1, \ldots, q_m$ in clockwise order. We move the point p around the boundary Q and find the polygons of $\mathcal{P}$ that contain p. The moving starts at point $p = q_1$. Determine the polygon of $\mathcal{P}$ containing p by trying each polygon of $\mathcal{P}$. Denote it by P. It is clear that $P \in \mathcal{P}'$.

The point p moves in direction from q_1 to q_2 (see Fig. 6). Determine the side e of P that intersects the line through p and q. We distinguish three cases to process the point p.

Case 1. The edge e intersects the line segment $[p, q]$. The edge e is common to two polygons of $\mathcal{P}$. One of them is P. It is clear that the second polygon belongs to $\mathcal{P}'$. Add it into $\mathcal{P}'$ and assign it as P.

Case 2. The edge e does not intersect the line through p and q. (This case can occur after Case 1.) Take the next edge of P after e in clockwise order as e.

Case 3. The edge e intersects the line through p and q but does not intersect the line segment $[p, q]$. Move the point p into q and take the next point after p as q.

The proof of correctness of the algorithm is straightforward.

7. Processing the Set B_3

In this section we show how to find the red–blue diameter of the sets B_3 and R. Our approach is a modification of the following simple scheme. Construct the restriction of the Voronoi diagram of B_3 to the boundary of H. By Theorem 4, the restriction has linear complexity. Locate each red point p in the subdivision of $bd(H)$ to determine the furthest blue point from p in B_3. The diameter $diam(R, B_3)$ is the largest distance from a red point to its furthest neighbor.

Consider the furthest-point Voronoi diagram of B_3 in $\mathbb{R}^3$. By Lemma 2, the boundary of the convex hull H is partitioned into $|B_3|$ connected domains that we refer to as the *surface Voronoi cells*. For a point $p \in B_3$, let $V(p) = Vor(p) \cap H$ denote the surface Voronoi cell corresponding to p. The surface Voronoi cells are determined by faces of polytopes $Vor(B_3)$ that intersect the convex hull H. Each face lies in the *bisector* plane of two points in B_3.

It should be noted that the description of $V(p)$ does not include all intersection points of the boundary of Voronoi cell $Vor(p)$ with the edges of H (otherwise the total description may have quadratic size). It contains vertices and edges defined as follows. The vertices of $V(p)$ correspond to the intersections of edges of $Vor(B_3)$ and the boundary of the convex hull H. In other words, a point $q \in bd(H)$ is the vertex of $V(p)$ if q is equidistant from at least three points in B_3 (p is one of them). An edge of $V(p)$ connects two vertices of $V(p)$ and separates two surface Voronoi cells, one of them is $V(p)$. In fact any edge of $V(p)$ is a polygonal chain in the convex hull. In spite of this we store only endpoints of the edge and the pair of data points whose surface Voronoi cells are shared by the edge.

Recall that B_3 is the set of points $p \in B$ such that the interior of the Voronoi cell $Vor(p)$ intersects the polygon $Q = H \cap \pi_1$. We decompose the problem of constructing the restriction of the Voronoi diagram of B_3 into two subproblems according to the halfspaces bounded by the plane π_1. In the subproblems we construct the restriction of surface Voronoi cells to $bd(H) \cap \pi_1^+$ and $bd(H) \cap \pi_1^-$. (In order to obtain the general solution we can glue the edges of surface Voronoi cells that intersect the plane π_1. In fact we do not need to combine the solutions, see Section 7.2.)

The important condition making it possible to solve subproblems efficiently is that the complexity of the restricted Voronoi diagrams is still linear. Consider a point p in B_3. How many chains can the Voronoi cell $Vor(p)$ produce? The number of chains might be $|Q|$ in the worst case ($|Q|$ means a number of vertices of Q). It follows from the fact that the intersection of an edge of Q and the Voronoi cell $Vor(p)$ is a segment (if any). The total number of chains is $O(N_3)$ because there is $O(N_3)$ edges of the subdivision $Vor(B_3) \cap \pi_1$ and each edge intersects the convex boundary of Q at most twice. To be more precise it gives the bound $6N_3$ on the number of chains.

Let $c_1, \dots, c_k$ be the polygonal chains and let e_i and e_{i+1} be the endpoints of the chain c_i. Note that k is at least N_3 where $N_3 = |B_3|$. Let $p_i, i = 1, \dots, k$, be the point of B_3 whose surface Voronoi cell contains the polygonal chain c_i. One can show that $(p_1, \dots, p_k)$ represents an $(N_3, 2)$ Davenport–Schinzel sequence [1], [13]. It follows that $k \leq 2N_3 - 1$.

7.1. *Restricted Voronoi Diagram*

In this section we show how to compute the restriction of the furthest-point Voronoi diagram to the upper hull $bd(H) \cap \pi_1^+$ (the lower hull can be processed similarly). Its description includes a list of surface Voronoi cells restricted to the upper hull, a list of edges and a list of vertices of the subdivision. We assume that the list of incident

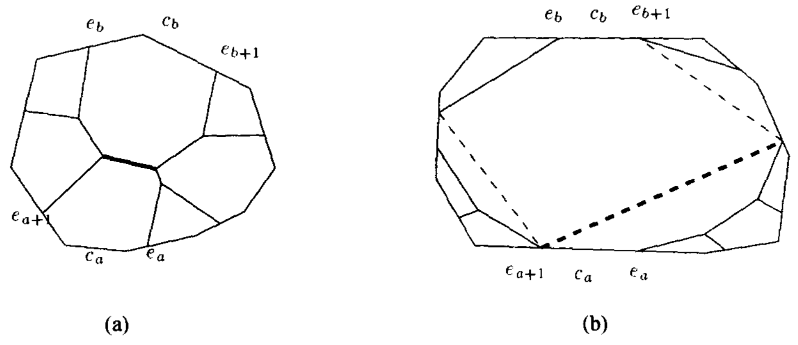

Fig. 7. (a) Bold edge is a centroid of T. (b) Dashed edges complete the binary tree T.

vertices/edges/faces to a vertex/edge/face can be accessed in $O(1)$ time. Note that a Voronoi cell can be split into many connected components in $bd(H) \cap \pi_1^+$, an edge can be split into at most two restricted edges producing new vertices. The lower hull can be processed similarly.

We construct the Voronoi diagram clipped by π_1 for the set B_3 using the algorithm described in Section 5. The polygon Q_e with vertices $e_1, e_2, \ldots, e_k$ is partitioned into cells corresponding to the points of B_3. Applying the divide-and-conquer approach we find two chains c_a and c_b, $a < b$, and split the problem into two subproblems corresponding to the point sets $P_1 = \{p_a, p_{a+1}, \ldots, p_b\}$ and $P_2 = \{p_b, p_{b+1}, \ldots, p_k, p_1, \ldots, p_a\}$. To balance the sizes of the subproblems we use *centroid decomposition* of Guibas et al. [16]. Without loss of generality we can assume that the vertices of the clipped Voronoi diagram have degree 3. If the edges of the Voronoi diagram clipped by the polygon Q_e are connected (in other words $k = N_3$) they define a binary tree T. In $O(N_3)$ time one can find a *centroid* of T using the algorithm [16]. The centroid edge decomposes T into two parts, each of size at least $\lfloor(|T| + 1)/3\rfloor$. The centroid can also be obtained by a polygon cutting theorem of Chazelle [10]. The Voronoi cells separated by the centroid edge define subproblems of balanced sizes, see Fig. 7(a). If there is a cell A with more than one chain we split it to complete the tree T, see Fig. 7(b).

For both subproblems P_1 and P_2, the chains around Q can be computed in linear time. There are two cases. If $p_a = p_b$, then the chain endpoints are $\{e_{a+1}, e_{a+2}, \ldots, e_b\}$ and $\{e_{b+1}, e_{b+2}, \ldots, p_k, p_1, \ldots, p_a\}$. Otherwise they are $\{e', e_{a+1}, e_{a+2}, \ldots, e_b\}$ and $\{e'', e_{b+1}, e_{b+2}, \ldots, p_k, p_1, \ldots, p_a\}$ where e' and e'' are the points of intersection of Q and the bisector of p_a and p_b.

The merging step works as follows. If $p_a = p_b$, then the edges of two surface Voronoi diagrams define the surface Voronoi diagram of $P_1 \cup P_2$. Suppose $p_a \neq p_b$. The surface Voronoi cells for $P_1 \cup P_2$ can be obtained by merging the edge structures similar to the Voronoi digram in the plane [18], [19], [28].

Lemma 7 (Merge Chain). *There is a path C of edges of the surface Voronoi cells for $P_1 \cup P_2$ such that*

- *every edge of C shares the surface Voronoi cells of points from different subsets P_1 and P_2*

- *the path C starts from e_a and ends in e_b, and*
- *the path C partitions the surface $H \cap \pi^+$ into two parts corresponding to P_1 and P_2.*

Proof. In fact C is the set of all edges sharing cells of points from different subsets. The points e_a and e_b are connected in C, otherwise there is a path connecting two points from the chains $C_{ab} = (e_a, e_{a+1}, \ldots, e_b)$ and $C_{ba} = (e_b, e_{b+1}, \ldots, e_a)$ avoiding C. Let $C' \subseteq C$ be the path connecting e_a and e_b. By Lemma 2 each point c in C can be connected to C_{ab}/C_{ba} by a path in a surface Voronoi cell of a point in P_1/P_2. Thus $c \in C'$ and $C = C'$. □

The problem now is to find the merge chain C. The computation of C can be viewed as the motion of a point q along C that discovers the endpoints of the edges on C. Let $\gamma(i, j)$ denote the bisector plane of the points p_i and p_j. Let P_{ij} denote the polygonal path $P_{ij} = \gamma(i, j) \cap bd(H) \cap \pi^+$. The algorithm starts with the point e_a that is the endpoint of an edge separating the surface Voronoi cells $V(p_a)$ and $V(p_{a-1})$. Consider the moment of time when we found an endpoint e that is the intersection of C and the edge separating two surface Voronoi cells, say $V(p_i)$ and $V(p_j)$, see Fig. 8. Without loss of generality we can assume that $p_i, p_j \in P_1$ and $i < j$. The path C enters the surface Voronoi cells $V(p_j)$. The edge of C incident to e separates $V(p_i)$ and the surface Voronoi cell of a point of P_2, say p_l. The point e is the first endpoint of new edge $E \in C$ separating the surface Voronoi cells $V(p_j)$ and $V(p_l)$.

To find the second endpoint e' of the edge E we apply a *chain shooting*. The edge E is a subchain of the polygonal chain P_{jl}. The chain shooting in $V(p_j)$ finds the first point t_j of the boundary of $V(p_j)$ in the way on P_{jl} starting at e, see Fig. 8. The path P_{jl} intersects the boundary of $V(p_j)$ in two points e and t_j. The chain shooting can be done by tracing the boundary of $V(p_j)$ in the direction opposite to the direction of polygonal

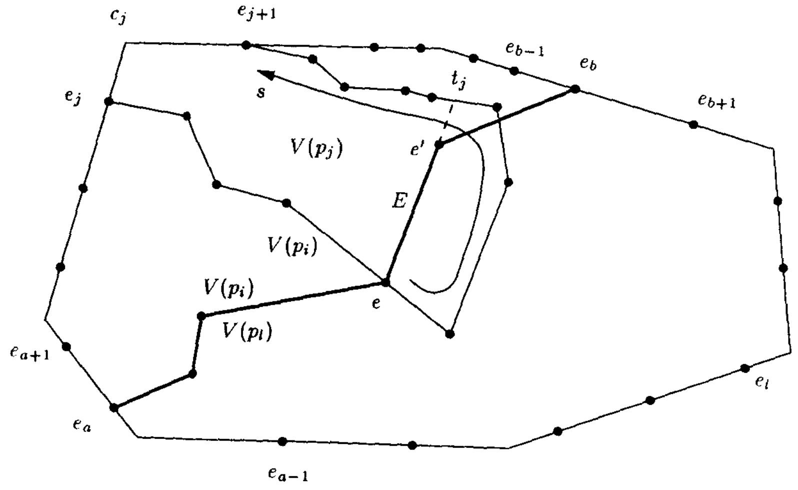

Fig. 8. Merge chain C.

chains $c_1, \ldots, c_k$ (in the projection onto the plane π). Let E' be the current edge of the boundary of $V(p_j)$ separating $V(p_j)$ and $V(p_s)$. To detect whether t_j belongs to E' we compute the intersection of the convex hull H and the line L that is the set of points equidistant from the points p_j, p_l and p_s. To do this we use the algorithm of Dobkin and Kirkpatrick [14]. Their algorithm takes

- $O(n \log n)$ time to preprocess the convex hull H, and
- $O(\log n)$ time to find the intersection of the line L and the convex hull H.

If the line L does not intersect the convex hull we pass the edge E' and take the next edge on the boundary $V(p_j)$. Suppose the line L intersects the boundary of the convex hull in two points. We choose the point r that is the first in the motion of e along the path P_{jl}.[1] The point r belongs to the bisector $\gamma(j,s)$. We can test whether it belongs to the edge E' in $O(1)$ time.[2] If the edge E' does not contain the point r, we pass it and take the next edge on the boundary $V(p_j)$.

We also apply the chain shooting in $V(p_l)$ to find the point t_l. The second endpoint e' of E is one of the points t_j and t_l that occurs first in the chain shooting. One of the cells $V(p_j)$ or $V(p_l)$ participates in the next edge of C. The tracing of its boundary can be continue starting with the last visited edge. This is important for the running time because we try to detect the intersection with an edge at most twice.

To prove the correctness of the algorithm we exploit the following lemmas.

Lemma 8. *For any points $p_j \in P_1$ and $p_l \in P_2$, the intersection of the surface Voronoi cell $V(p_j)$ (in the Voronoi diagram of P_1) and the bisector plane of p_j and p_l is a polygonal chain (if any).*

Proof. Let I denote the intersection of the surface Voronoi cell $V(p_j)$ and the bisector $\gamma(j,l)$ (or P_{jl}). Let u and v be any distinct points in I. Consider the subchain of P_{jl} connecting u and v. Suppose that there is a point c in this chain that does not belong to the surface Voronoi cell $V(p_j)$. There is a point $p_s \in P_1 \setminus \{p_j\}$ whose surface Voronoi cell $V(p_s)$ contains c. The plane passing through u, v and p_j gives a polygonal chain C_1 from u to v in $V(p_j)$ (see the proof of Lemma 2). Using the plane passing through u, v and p_l we construct a polygonal chain C_2 from u to v.

Consider the furthest-point Voronoi diagram of the set $P_1 \cup \{p_l\}$. The points c and e_s still lie in $V(p_s)$. However, the closed path $C_1 \cup C_2$ separates them ($C_1 \subset V(p_j)$ and $C_2 \subset V(p_l)$). Contradiction. □

Lemma 9. *Two consecutive chain shootings in $V(p_j)$ do not intersect.*

Proof. Let the shot by P_{jl} end in the point e' of an edge of $V(p_l)$ separating $V(p_l)$ and $V(p_s)$ (where $p_l, p_s \in P_2$). The next shot in $V(p_j)$ is defined by the chain P_{js}. Suppose it

[1] The direction of the motion in the chain P_{jl} can be defined using the halfplane $\{t \mid dist(t, p_j) = dist(t, p_l)$ and $dist(t, p_j) \le dist(t, p_i)\}$ for $e \ne e_a$ and the halfplane $\pi^+ \cap \{t \mid dist(t, p_1) = dist(t, p_k)\}$ for $e = e_1$.

[2] The edge E' does not contain the point r if and only if the line passing through endpoints of E' separates the points r and $(p_j + p_s)/2$ in the bisector $\gamma(j,s)$.

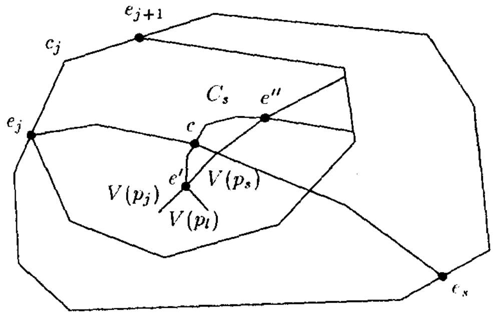

Fig. 9. Path e_jce_s separates e' and e''.

intersects the chain P_{jl} at the point $e'' \neq e'$. Consider the furthest-point Voronoi diagram of the set $P_1 \cup \{p_l, p_s\}$. It contains the edge C_s sharing the cells of p_j and p_s. Let c be any internal point of C_s. The points e_j and e_s can be connected by a path from e_j to c in $V(p_j)$ (by Lemma 2) and a path from c to e_s in $V(p_s)$. This path separates the points e' and e'', see Fig. 9. However, they are connected in the cell of $V(p_l)$ by the path in the plane passing through points e', e'' and p_l. Contradiction. □

Theorem 10. *The algorithm for building the surface Voronoi cells of B_3 in the upper cap of H is correct and takes $O(N_3 \log N_3 \log N)$ time where $N_3 = |B_3|$ and N is the number of vertices of H.*

Proof. The basic step of the algorithm is to find hit points t_j and t_l in the chain shooting in $V(p_j)$ and $V(p_l)$. Lemma 8 implies the existence of the points t_j and t_l. By Lemma 9, the endpoints t_j in the boundary of $V(p_j)$ computed by chain shooting in $V(p_j)$ form a monotone sequence in $bd(V(p_j))$. Note that chain shooting in $V(p_j)$ never crosses over the chain c_j. Hence the tracing of edges is correct.

Now we analyze the running time of constructing the surface Voronoi diagram. The input of the recursive procedure contains k chains in $bd(Q)$ and the convex hull H with at most N vertices. The reduction to the subproblems takes $O(k)$ time. The sizes of subproblems are at most $2k/3 + O(1)$. The merge chain has $O(k)$ vertices that can be found in $O(k \log N)$ time using a Dobkin–Kirkpatrick hierarchy [14]. The surface Voronoi diagram is constructed in $O(k \log k \log N)$ time. Hence the total running time is $O(N_3 \log N_3 \log N)$. □

7.2. *Furthest Neighbors*

In this section we explain how to find furthest neighbors. Specifically, for any point of R, we find its furthest neighbor among the points of B_3.

Theorem 11. *For n_R red points, the furthest neighbors among N_3 blue points of B_3 can be found in $O((n_R + N_3) \log N_3)$ time using the surface Voronoi diagram.*

Proof. We also divide the problem into two subproblems, for R_1 and R_2. The subproblems are similar, so we consider only the subproblem for R_2. The surface Voronoi cells induce a forest whose trees can be linked and form a binary tree T as in the previous section. Applying the divide-and-conquer approach we also use the centroid decomposition of Guibas et al. [16]. Let E be the centroid edge of T separating two chains c_a and c_b. There are two cases.

Case 1. $p_a \neq p_b$. Let e be an endpoint of E (it may be any point of E). The upper hull can be partitioned into two parts by a path $e_a e e_b$ as follows. The path $e_a e$ lies in the intersection of the convex hull and the plane α passing through points p_a, e_a and e. The plane β passing through points p_b, e_b and e defines the path ee_b. The points of R_2 are partitioned into two subsets by the path $e_a e e_b$. For a point $p \in R_2$ we determine a halfspace bounded by a bisector plane of p_a and p_b. If p_a is closer to p than p_b, then the location of p is defined by the plane α. Otherwise it depends on the plane β ($p \in \beta^+$ or $p \in \beta^-$).

Case 2. $p_a = p_b$. The partition algorithm is very simple. The points of R_2 are divided by the plane passing through p_a, e_a and e_b.

At the end of the recursion (if the number of data points is $O(1)$ or the number of vertices of a tree is $O(1)$) computing the furthest neighbors is straightforward.

The points of R_1 can be handled in similar way.

Now we estimate the running time. The algorithm is recursive. The centroid edge is found in $O(N_3)$ time. To split the red points the algorithm spends $O(n_R)$ time. Thus the algorithm takes linear time before recursive calls. Note that the total number of red and blue points in two recursive calls is $n_R + N_3 + 2$. The number of levels is $O(\log N_3)$ by choosing centroid edges. The total running time is $O((n_R + N_3)\log N_3)$. □

8. Analysis

We analyze the running time of the entire algorithm.

Theorem 12. *The diameter of a set of n points in $\mathbb{R}^3$ can be computed in $O(n \log^2 n)$.*

Proof. The running time of the diameter algorithm is $O(T(n))$ where $T(N)$ is the running time of computing the red–blue diameter of the total N red and blue points. Let $T'(N)$ be the running time except for the time for processing the set B_3. By Lemma 6,

$$T'(N) = O(N \log N) + T'(N_1) + T'(N_2), \tag{1}$$

producing $T'(N) = O(N \log^2 N)$. Note that each point might participate in B_3 at most once. The total running time for constructing Voronoi diagrams in the planes (Section 5) and for partitioning the set B (Section 6) is $O(n \log n)$. By Theorem 10 the total running time to construct all restricted Voronoi diagrams is $O(n \log^2 n)$. The furthest neighbors are computed in $O((n_R + N_3)\log N_3)$ time by Theorem 11. The part $O(N_3 \log N_3)$ is dominated by the time for computing the surface Voronoi diagram. The part $O(n_R \log N_3)$ is dominated by the first addend in (1). Hence the total running time $T(n) = O(n \log^2 n)$. □

9. Applications

In this section we discuss a few applications of our approach. Aggarwal and Kravets [2] gave a linear-time algorithm for finding all furthest neighbors in a convex polygon. We consider a three-dimensional analog, the all-furthest neighbors (AFN) for points on a convex polytope.

Theorem 13. *The all-furthest neighbors for points on a convex polytope in* $\mathbb{R}^3$ *can be computed in* $O(n \log^2 n)$ *time.*

Proof. We modify the diameter algorithm. The algorithm uses a reduction of the AFN problem to a *red–blue AFN* problem similar to one in Section 3. The red–blue AFN problem is to find furthest blue neighbors for all red points. We split the bounding box A of the size $a \times b \times c$ by nine planes, three planes per each axes, into 4^3 subboxes of equal size. A point p and its furthest neighbor q lie in different subboxes because otherwise

$$\begin{aligned} dist(p, q) &\le \sqrt{\left(\frac{a}{4}\right)^2 + \left(\frac{b}{4}\right)^2 + \left(\frac{c}{4}\right)^2} \\ &\le \frac{\sqrt{3}}{4} \max(a, b, c) < \frac{\max(a, b, c)}{2} \le dist(p, q). \end{aligned}$$

The last inequality follows from the fact that the distance from p to one of the points defining the longest side s of A is at least $|s|/2$. The red–blue AFN problem is asymmetric, so we add extra calls of `rb_diam()` with opposite side of the splitting planes. The total number of calls of `rb_diam()` is 9.

Note that all the points are in the convex polytope P. We do not exchange colors in Step 2 of `rb_diam()` and do not compute the convex hull H in Step 3. The algorithm uses the boundary of P as H and the precomputed Dobkin–Kirkpatrick hierarchy DK for P. It follows that the size of the polygon Q might be $\Omega(n)$. To avoid the walk around Q in the algorithm of Section 6 we apply edge queries to Q. To decide if an edge e intersects the boundary of Q we use DK hierarchy. The edge query can be answered in $O(\log n)$ time. Therefore the partition of B can be computed in $O(N_3 \log n)$ time.

The surface Voronoi cells and the furthest neighbors in Section 7 can be found in $O(N_3 \log^2 n + n_R \log n)$ time. The first part $O(N_3 \log^2 n)$ contributes $O(n_B \log^2 n)$ over all recursive calls, so we can ignore it in the runtime analysis. The algorithm `rb_diam()` has $O(\log n_R)$ levels of the recursion. At each level of the recursion it spends $O(n \log n)$ total time because each red and blue point participates in at most one call of `rb_diam()`. The total running time of the AFN algorithm is $O(n \log^2 n)$. □

Toussaint [29] gave a linear-time algorithm for finding the *symmetric all-furthest neighbors* in the plane. For a set S, the symmetric neighbors are defined as a set of pairs $p, q \in S$ such that p is the furthest neighbor of q and vice versa. Note that the symmetric neighbors are located in the convex hull of S. Theorem 13 implies the following result in $\mathbb{R}^3$.

Corollary 14. *The symmetric all-furthest neighbors in $\mathbb{R}^3$ can be found in $O(n \log^2 n)$ time.*

We consider two other applications, the colored versions of the above problems. In the *chromatic AFN* problem we are given a colored set S of n points in $\mathbb{R}^3$ and, for each point $p \in S$, we want to find its furthest neighbor of a different color.

Theorem 15. *For a colored set of n point on a convex polytope, the chromatic all-furthest neighbors can be found in $O(n \log^2 n \log k)$ time where k is number of colors.*

Proof. First we show how to solve the problem for two colors, i.e. $k = 2$. It suffices to solve the asymmetric version, which is to find the furthest blue neighbor for each red point. In contrast to Section 4 the points are not necessarily separated by a plane. We split the bounding box of blue points as in the proof of Theorem 13. In addition to the nine planes we consider six planes passing through the faces of A. Each of the six planes defines one call of `rb_diam()` (because there are no blue points on one side of the plane). Each of the nine planes defines two calls of `rb_diam()`. The rest is straightforward.

For arbitrary number of colors we apply divide-and-conquer technique. We partition the colors into two groups of at most $\lceil k/2 \rceil$ colors and color the points into two colors according to these groups. We solve the bichromatic AFN problem as described above and proceed with each group of colors recursively. The total running time is $O(n \log^2 n \log k)$. $\square$

Corollary 16. *The chromatic diameter of n points in $\mathbb{R}^3$ colored in k colors can be computed in $O(n \log^2 n \log k)$ time.*

Remark. The chromatic diameter can be computed in $O(T_d(n) \log k)$ time using a diameter algorithm with $O(T_d(n))$ running time. Hence it can be computed in $O(n \log n \log k)$ time using an algorithm of Ramos [27].

10. Conclusion

We presented a $O(n \log^2 n)$ algorithm for computing the diameter of n points in $\mathbb{R}^3$. The algorithm is based on the idea of restriction of the furthest-point Voronoi diagram to the convex hull (in a special case) reducing the complexity from $O(n^2)$ to $O(n)$. It would be interesting to find an optimal algorithm for computing the diameter using our approach. We applied our technique for finding the all-furthest neighbors for points on the convex hull, the symmetric all-furthest neighbors for any points and for chromatic versions of these problems in $\mathbb{R}^3$. We mention two open problems in $\mathbb{R}^3$ whether it possible in $O(n \log n)$ time to compute

- the restriction of the furthest-point Voronoi diagram to the convex hull, and
- all furthest neighbors.

We believe that our approach can also be applied to other problems in $\mathbb{R}^3$.

References

1. P. K. Agarwal and M. Sharir, *Davenport–Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, Cambridge, 1995.
2. A. Aggarwal and D. Kravets, A linear time algorithm for finding all farthest neighbors in a convex polygon, *Inform. Process. Lett.*, **31**(1) (1989), 17–20.
3. N. M. Amato, M. T. Goodrich, and E. A. Ramos, Parallel algorithms for higher-dimensional convex hulls, *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science*, pp. 340–347, 1994.
4. F. Aurenhammer, A criterion for the affine equality of cell complexes in $\mathbb{R}^d$ and convex polyhedra in $\mathbb{R}^{d+1}$, *Discrete Comput. Geom.*, **2** (1987), 49–64.
5. F. Aurenhammer, Power diagrams: properties, algorithms and applications, *SIAM J. Comput.*, **16** (1987), 78–96.
6. F. Aurenhammer, Improved algorithms for discs and balls using power diagrams, *J. Algorithms*, **9** (1988), 151–161.
7. F. Aurenhammer, Linear combinations from power domains, *Geom. Dedicata*, **28** (1988), 45–52.
8. H. Brönninman, B. Chazelle, and J. Matoušek, Product range spaces, sensitive sampling, and derandomization, *SIAM J. Comput.*, **28**(5) (1999), 1552–1575.
9. J. L. Bentley and T. A. Ottmann, Algorithm for reporting and counting geometric intersections, *IEEE Trans. Comput.*, **28** 1979, 643–647.
10. B. Chazelle, A theorem on polygon cutting with applications, *Proc. 23rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 339–349, 1982.
11. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Diameter, width, closest line pair, and parametric search, *Discrete Comput. Geom.*, **10** (1993), 145–158.
12. K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II. *Discrete Comput. Geom.*, **4** (1989), 387–421.
13. H. Davenport and A. Schinzel, A combinatorial problem connected with differential equations, *Amer. J. Math.*, **87** (1965), 684–689.
14. D. P. Dobkin and D. G. Kirkpatrick, Fast detection of polyhedral intersection, *Theoret. Comput. Sci.*, **27** (1983), 241–253.
15. H. Edelsbrunner, Computing the extreme distances between two convex polygons, *J. Algorithms*, **6** (1985), 213–224.
16. L. Guibas, J. Hershberger, D. Leven, M. Sharir and R. E. Tarjan, Linear time algorithms for visibility and shortest path problems inside triangulated simple polygons, *Algorithmica*, **2** (1987), 209-233.
17. H. Imai, M. Iri, and K. Murota, Voronoi diagrams in the Laguerre geometry and its applications, *SIAM J. Comput.*, **14** (1985), 93–105.
18. D. T. Lee, Proximity and Reachability in the Plane, Technical Report No. R-831, Coordinate Science Laboratory, University of Illinois at Urbana, IL, 1978.
19. D. T. Lee, Two-dimensional Voronoi diagrams in the L_p-metric, *J. Assoc. Comput. Mach.*, **27** (1980), 604–618.
20. J. Matoušek and O. Schwarzkopf, On ray shooting in convex polytopes, *Discrete Comput. Geom.*, **10** (1993), 215–232.
21. J. Matoušek and O. Schwarzkopf, A deterministic algorithm for the three-dimensional diameter problem, *Comput. Geom. Theory Appl.*, **6** (1996), 253–262.
22. N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. Assoc. Comput. Mach.*, **30** (1983), 852–865.
23. F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM*, **20**(2) (1977), 87–93.
24. F. P. Preparata and M. I. Shamos, *Computational Geometry, an Introduction*, 3rd edn., Springer-Verlag, New York, 1990.
25. E. A. Ramos, Intersection of unit-balls and diameter of a point set in $\mathbb{R}^3$, *Comput. Geom. Theory Appl.*, **6** (1996), 57–65.
26. E. A. Ramos, Construction of 1-d lower envelopes and applications, *Proc. 13th Annual ACM Symposium on Computational Geometry*, pp. 57–66, 1997.

27. E. A. Ramos, Deterministic algorithms for 3-D diameter and some 2-D lower envelopes, *Proc. 16th Annual ACM Symposium on Computational Geometry*, pp. 290–299, 2000.
28. M. I. Shamos and D. Hoey, Closest-point problems, *Proc. 16th Annual IEEE Symposium on Foundations of Computer Science*, pp. 151–162, 1975.
29. G. T. Toussaint, The symmetric all-furthest neighbor problem, *J. Comput. Math. Appl.*, **9**(6) (1983), 747–754.
30. A. C. Yao, On constructing minimum spanning trees in k-dimensional spaces and related problems, *SIAM J. Comput.*, **11** (1982), 721–736.

Received February 18, 2000, *and in revised form June* 27, 2000. *Online publication January* 17, 2001.