Computing the Diameter of a Point Set

Grégoire Malandain, Jean-Daniel Boissonnat

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Computing the Diameter of a Point Set

Grégoire Malandain — Jean-Daniel Boissonnat

N° 4233

July 27, 2001

THÈMES 3 et 2

ISSN 0249-6399 ISRN INRIA/RR--4233--FR+ENG

INRIA
SOPHIA ANTIPOLIS

Computing the Diameter of a Point Set

Grégoire Malandain, Jean-Daniel Boissonnat

Thèmes 3 et 2 — Interaction homme-machine,
images, données, connaissances — Génie logiciel
et calcul symbolique
Projets Epidaure et Prisme

Rapport de recherche n° 4233 — July 27, 2001 — 29 pages

Abstract: Given a finite set of points $\mathcal{P}$ in $\mathbb{R}^d$, the diameter of $\mathcal{P}$ is defined as the maximum distance between two points of $\mathcal{P}$. We propose a very simple algorithm to compute the diameter of a finite set of points. Although the algorithm is not worst-case optimal, it appears to be extremely fast for a large variety of point distributions.

Key-words: Computational geometry, diameter.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles, BP 93, 06902 Sophia Antipolis Cedex (France)
Téléphone : +33 4 92 38 77 77 — Télécopie : +33 4 92 38 77 65

Calculer le diamètre d'un ensemble de points

Résumé : Etant donné un ensemble fini $\mathcal{P}$ de points de $\mathbb{R}^d$, le diamètre de $\mathcal{P}$ est la distance maximale entre deux points de $\mathcal{P}$. On propose un algorithme très simple pour calculer le diamètre d'un ensemble fini de points. Bien que l'algorithme ne soit pas optimal dans le cas le pire, il s'avère extrêmement rapide pour une grande variété de distributions de points.

Mots-clés : Géométrie algorithmique, diamètre.

1 Introduction

Given a set $\mathcal{P}$ of n points in $\mathbb{R}^d$, the *diameter* of $\mathcal{P}$ is the maximum Euclidean distance between any two points of $\mathcal{P}$.

Computing the diameter of a point set has a long history. By reduction to set disjointness, it can be shown that computing the diameter of n points in $\mathbb{R}^d$ requires $\Omega(n \log n)$ operations in the algebraic computation-tree model [PS90]. A trivial $O(n^2)$ upper-bound is provided by the brute-force algorithm that compares the distances between all pairs of points. In dimensions 2 and 3, better solutions are known. In the plane, it is easy to solve the problem optimally in $O(n \log n)$ time. The problem becomes much harder in $\mathbb{R}^3$. Clarkson and Shor gave a randomized $O(n \log n)$ algorithm [CS89]. This algorithm involves the computation of the intersection of n balls (of the same radius) in $\mathbb{R}^3$ and the fast location of points with respect to this intersection. This makes the algorithm less efficient in practice than the brute-force algorithm for almost any data set. Moreover this algorithm is not efficient in higher dimensions since the intersection of n balls of the same radius has size $\Theta(n^{\lfloor \frac{d}{2} \rfloor})$. Recent attempts to solve the 3-dimensional diameter problem led to $O(n \log^3 n)$ [AGR94, Ram97b] and $O(n \log^2 n)$ deterministic algorithms [Ram97a, Bes98]. Finally Ramos found an optimal $O(n \log n)$ deterministic algorithm [Ram00]. All these algorithms use complex data structures and algorithmic techniques such as 3-dimensional convex hulls, intersection of balls, furthest-point Voronoi diagrams, point location search structures or parametric search. We are not aware of any implementation of these algorithms. We suspect that they are very slow in practice compared to the brute-force algorithm, even for large data sets.

Some of these algorithms could be extended in higher dimensions. However, this is not worth trying since the data structures they use have sizes that depend exponentially on the dimension: e.g. the size of the convex hull of n points of $\mathbb{R}^d$ can be as large as $\Omega(n^{\lfloor \frac{d}{2} \rfloor})$.

Our algorithm works in any dimension. Moreover, it does not construct any complicated data structure; in particular, it does not require that the points are in convex position and therefore does not require to compute the convex hull of the points. The only numerical computations are dot product computations as in the brute-force algorithm.

The algorithm is not worst-case optimal but appears to be extremely fast under most circumstances, the most noticeable exception occuring when the points are distributed on a domain of constant width, e.g. a sphere. We also propose an approximate algorithm.

Independently, Har-Peled has designed an algorithm which is similar in spirit to our algorithm [Har01]. We compare both methods and also show that they can be combined so as to take advantage of the two.

2 Definitions, notations and geometric preliminaries

We denote by n the number of points of $\mathcal{P}$, by h the number of vertices of the convex hull of $\mathcal{P}$, and by D the diameter of $\mathcal{P}$. $\delta(\cdot,\cdot)$ denotes the Euclidean distance, and $\delta^2(\cdot,\cdot)$ the squared Euclidean distance.

A pair of points of $\mathcal{P}$ is called a *segment*. The length of a segment pq is the euclidean distance $\delta(p,q)$ between p and q. A segment of length D is called *maximal*.

For $p \in \mathcal{P}$, $FP(p)$ denotes the subset of the points of $\mathcal{P}$ that are at maximal distance from p. The segment joining two points p and q is called a *double normal* if $p \in FP(q)$ and $q \in FP(p)$. If pq is a maximal segment, pq is a double normal. The converse is not necessarily true.

Observe that the endpoints of a maximal segment or of a double normal belong to the convex hull of $\mathcal{P}$. Observe also that, if the points are in *general position*, i.e. there are no two pairs of points at the same distance, the number of double normals is at most $h/2$.

$B(p,r)$ denotes the ball of radius r centered at p, $\Sigma(p,r)$ its bounding sphere. The ball with diameter pq is denoted by $B[pq]$ and its boundary by $\Sigma[pq]$.

Since the distance between any two points in $B[pq]$ is at most than $\delta(p,q)$, we have:

Lemma 1 *If $p, q \in \mathcal{P}$ and if pq is not a maximal segment, any maximal segment must have at least one endpoint outside $B[pq]$.*

As a corollary, we have:

Lemma 2 *If $p, q \in \mathcal{P}$ and if $\mathcal{P} \setminus B[pq] = \emptyset$, pq is a maximal segment of $\mathcal{P}$ and $\delta(p,q)$ is the diameter of $\mathcal{P}$.*

3 Computation of a double normal

Algorithm 1 below repeatidly computes a furthest neighbour of a point of $\mathcal{P}$ until a double normal DN is found. To find a furthest neighbour of $p \in \mathcal{P}$, we simply compare the distances between p and all the other points in $\mathcal{P}$ (FP scan). Point p is then removed from $\mathcal{P}$ and won't be considered in further computations.

```
 1: procedure DoubleNormal( p, P )    // p is a point of P
 2: Δ²_0 = 0   i = 0
 3: repeat    // FP scan
 4:    increment i
 5:    Δ²_i = Δ²_{i-1}
 6:    P = P \ {p}    // remove p from P from any further computation
 7:    find q ∈ FP(p), i.e. one of the furthest neighbours of p
 8:    if δ²(p,q) > Δ²_i then
 9:       Δ²_i = δ²(p,q) and DN = pq
10:       p = q
11: until (Δ²_i = Δ²_{i-1})
12: return DN
```

Algorithm 1: Computes a double normal.

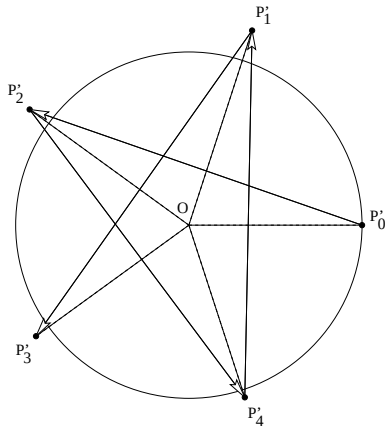


Figure 1: Proof of lemma 5.

Lemma 3 *Algorithm 1 terminates and returns a double normal.*

Proof. Δ_i can only take a finite number of different values and strictly increases: this ensures that the algorithm terminates. After termination (after I iterations) we have $q \in FP(p)$ and all the points of $\mathcal{P}$ belong to $B(p, \delta(p,q))$. Since $\Delta_{I-1} = \delta(p,q)$, all the points of $\mathcal{P}$ belong also to $B(q, \delta(p,q))$ and therefore $p \in FP(q)$. □

After termination of Algorithm 1, the original set $\mathcal{P}$ has been replaced by a strictly subset $\mathcal{P}'$ since some points have been removed from $\mathcal{P}$ (line 6 of algorithm 1). By construction, the returned segment pq is a double normal of the reduced set $\mathcal{P}'$ (lemma 3), and it is also a double normal of the original set $\mathcal{P}$.

Lemma 4 *The only numerical operations involved in Algorithm 1 are comparisons of squared distances.*

Lemma 5 *Algorithm 1 performs at most h FP scans and takes $\Theta(nh)$ time.*

Proof. The upper bound is trivial since all the points q that are considered by Algorithm 1 belong to the convex hull of $\mathcal{P}$ and all points q are distinct. As for the lower bound, we give an example in the plane, which is sufficient to prove the bound. Consider a set of $2n+1$ points $p_0, \ldots, p_{2n}$ placed at the vertices of a regular polygon P (in counterclockwise order). For $i > 0$, we slightly move the p_i outside P along the ray Op_i by a distance ε^i for some small $\varepsilon < 1$. Let p'_i be the perturbed points (see figure 1). It is easy to see that the farthest point from p'_i is always $p'_{i+n \bmod (2n+1)}$ except for p'_{n+1}. Therefore, the algorithm will perform FP scans starting successively at $p_{\sigma_0}, \ldots, p_{\sigma_{2n+1}}$ where $\sigma_i = i \times n$ (modulo $2n+1$). □

Although tight in the worst-case, the bound in lemma 5 is very pessimistic for many point distributions. This will be corroborated by experimental results.

4 Iterative computation of double normals

Assume that Algorithm 1 has been run and let $\mathcal{Q} = \mathcal{P} \setminus B[pq]$. If $\mathcal{Q} = \emptyset$, pq is a maximal segment and $\delta(p,q)$ is the diameter of $\mathcal{P}$ (lemma 1). Otherwise, we have to determine whether pq is a maximal segment or not. Towards this goal, we run Algorithm 1 again, starting at a point in $\mathcal{Q}$ rather than in $\mathcal{P}$, which is sufficient by lemma 2. Although any point in $\mathcal{Q}$ will be fine, experimental evidence has shown that choosing the furthest point from $\Sigma[pq]$ is usually better.

Algorithm 2 below repeats this process further until either $\mathcal{Q}$ becomes empty or the current maximal distance Δ does not increase.

```
 1: Δ² = 0   stop = 0
 2: pick a point m ∈ P
 3: repeat   // DN scan
 4:   DoubleNormal( m, P )   // yields a double normal pq of length δ(p,q)
 5:   if δ²(p,q) > Δ² then
 6:     Δ² = δ²(p,q) and DN = pq
 7:     Q = P \ B[pq]
 8:     if Q ≠ ∅ then
 9:       find m ∈ P a furthest point from Σ[pq]
10:   else
11:     stop = 1   // terminates with Q ≠ ∅.
12: until Q = ∅ or stop = 1
13: return DN = pq, Δ² = δ²(p,q)
```

Algorithm 2: Iterated search for double normals.

Lemma 6 *Algorithm 2 can be implemented so that the only numerical computations are comparisons of dot products of differences of points.*

Lemma 7 *Algorithm 2 performs $O(h)$ DN scans. Its overall time-complexity is $O(nh)$.*

Proof. The first part of the lemma comes from the fact that the algorithm enumerates (possibly all) double normals by strictly increasing lengths.

Let us prove now the second part of the lemma. Each time Algorithm 1 performs a FP scan starting at a point p (loop 3-11), p is removed from further consideration (line 6). Moreover, except for the first point p to be considered, all these points belong to the convex hull of $\mathcal{P}$. It follows that the total number of FP scans is at most $h+1$. Since each FP scan takes $O(n)$ time, we have proved the lemma. □

5 Diameter computation

Assume that Algorithm 2 terminates after I iterations. Since, at each iteration, a new double normal is computed, the algorithm has computed I double normals, noted p_iq_i, $i = 1, \ldots, I$, and we have $\delta(p_1, q_1) < \ldots < \delta(p_{I-1}, q_{I-1})$.. Each time Algorithm 1 is called, some points are removed from the original data set. We rename the original data set $\mathcal{P}^{(0)}$ and denote by $\mathcal{P}^{(j)}$ the set of points that remain after the j-th iteration, i.e. the one that computes p_jq_j. Hence set $\mathcal{P}^{(i)}$ is strictly included in $\mathcal{P}^{(i-1)}$. Moreover, each segment p_iq_i is a double normal for all the sets $\mathcal{P}^{(j)}$, $j = i-1, \ldots, I$.

It is easily seen that, at each iteration j, the length of the computed double normal p_jq_j is strictly greater than the distances $\delta(x, FP(x))$ computed so far, or equivalently, than the lengths of all the segments in $\mathcal{P} \setminus \mathcal{P}^{(j)} \times \mathcal{P}$ since Algorithm 1 removed the corresponding x from $\mathcal{P}$.

When Algorithm 2 terminates, we are in one of the two following cases :

Case 1 : $\delta(p_I, q_I) > \delta(p_{I-1}, q_{I-1})$ and $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_Iq_I] = \emptyset$.

In this case, p_Iq_I is a maximal segment of $\mathcal{P}$: by lemma 2, it is a maximal segment of $\mathcal{P}^{(I)}$, and, as mentionned above, no segment with an endpoint in $\mathcal{P} \setminus \mathcal{P}^{(I)}$ can be longer.

Case 2 : $\delta(p_I, q_I) \leq \delta(p_{I-1}, q_{I-1})$.

In this case, $\mathcal{P}^{(I-1)} \setminus B[p_{I-1}\, q_{I-1}]$ was not empty *before* the computation of $[p_I, q_I]$. We have to determine whether $p_{I-1}q_{I-1}$ is a maximal segment or not. Thanks to lemma 1, if a longer double normal exists, one of its endpoints lies in $\mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}]$. If this last set is empty, which is checked by Algorithm 3, $p_{I-1}q_{I-1}$ is a maximal segment of $\mathcal{P}$.

Required: $\mathcal{P}^{(I)}$ and $p_{I-1}q_{I-1}$ (provided by Algorithm 2)
1: $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B\,[p_{I-1}\, q_{I-1}]$
2: **if** $\mathcal{Q} = \emptyset$ **then**
3: $\quad p_{I-1}q_{I-1}$ is a maximal segment of $\mathcal{P}$

Algorithm 3: Checks whether $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}] = \emptyset$.

If $\mathcal{Q} = \mathcal{P}^{(I)} \setminus B[p_{I-1}\, q_{I-1}] \neq \emptyset$, we have to check whether there exists a maximal segment with an endpoint in this set. To search for such maximal segments, we propose two methods. For clarity purpose, we will write $\mathcal{P}$ instead $\mathcal{P}^{(I)}$ in the following.

5.1 Exhaustive search over $\mathcal{Q} \times \mathcal{P}$

The first method (Algorithm 4) simply considers all segments in $\mathcal{Q} \times \mathcal{P}$.

Required: Δ^2 (provided by Algorithm 2) and $\mathcal{Q}$ (provided by Algorithm 3)
1: **if** $\mathcal{Q} \neq \emptyset$ **then** // `Exhaustive search with an endpoint in` $\mathcal{Q}$
2: **for all** points $p_i \in \mathcal{Q}$ **do**
3: **for all** points $p_j \in \mathcal{P}$ **do**
4: **if** $\delta^2(p_i, p_j) > \Delta^2$ **then**
5: $\Delta^2 = \delta^2(p_i, p_j)$
6: **return** Δ^2

Algorithm 4: Exhaustive search over $\mathcal{Q} \times \mathcal{P}$.

5.2 Reduction of $\mathcal{Q}$

As it might be expected and is confirmed by our experiments (see section 7), the observed total complexity is dominated by the exhaustive search of the previous section. It is therefore important to reduce the size of $\mathcal{Q}$. For that purpose, we propose to reuse all the computed segments $p_i q_i$, $i = 1, \ldots, I-2$, and $p_I q_I$.

5.2.1 Principle

Assume that we have at our disposal an approximation Δ of the diameter of set $\mathcal{P}$ and a subset $\mathcal{Q} \subset \mathcal{P}$ that contains an endpoint of each maximal segment longer than Δ (plus possibly other points). To find such endpoints in $\mathcal{Q}$, we may, as in Algorithm 4, exhaustively search for a maximal segment over $\mathcal{Q} \times \mathcal{P}$.

Under the assumption that the diameter of $\mathcal{P}$ is larger than Δ, we know, from lemma 1, that any maximal segment will have at least one endpoint outside any ball of radius $\Delta/2$.

Consider such a ball B' of radius $\Delta/2$. The exhaustive search over $\mathcal{Q} \times \mathcal{P}$ can then be reduced to two exhautive searches associated to a partition of $\mathcal{Q}$ into $\mathcal{Q} \cap B'$ and $\mathcal{Q} \setminus B'$. More precisely, if $p \in \mathcal{Q}$, searching for a point q such that $\delta(p, q) > \Delta$ reduces to searching q in $\mathcal{P} \setminus \mathcal{Q} \setminus B'$ if p belongs to B', and searching q in $\mathcal{P}$ otherwise.

This way, instead of searching over $\mathcal{Q} \times \mathcal{P}$, we search over $(\mathcal{Q} \cap B') \times (\mathcal{P} \setminus \mathcal{Q} \setminus B')$ and $(\mathcal{Q} \setminus B') \times \mathcal{P}$, therefore avoiding searching a maximal segment in $(\mathcal{Q} \cap B') \times (\mathcal{P} \cap B')$.

B' should be chosen so as to maximize the number of points in $\mathcal{P} \cap B'$, which reduces the cost of searching over $(\mathcal{Q} \cap B') \times (\mathcal{P} \setminus \mathcal{Q} \setminus B')$. The idea is to reuse the already found segments $p_i q_i$ (which are double normals of $\mathcal{P}$) and to iteratively center the balls of radius $\Delta/2$ at the points $\frac{p_i + q_i}{2}$.

Required: $\Delta^2 = \delta^2(p_{max}, q_{max})$ and $\mathcal{S}$ provided by Algorithm 2
Required: $\mathcal{Q}^{(0)} = \mathcal{Q}$ provided by Algorithm 3
1: **for all** segments $p_i q_i \in \mathcal{S}$, $i = 1 \ldots |S|$ **do**
2: $B' = B\left(\frac{p_i+q_i}{2}, \Delta/2\right)$
3: $d^2 = \max \delta^2(p,q) \quad (q,p) \in \left(\mathcal{Q}^{(i-1)} \cap B'\right) \times \left(\mathcal{P} \setminus \mathcal{Q}^{(i-1)} \setminus B'\right)$
4: **if** $d^2 > \Delta^2$ **then** // `A better diameter estimation was found`
5: $\Delta^2 = d^2$
6: Add segment pq to set $\mathcal{S}$
7: $\mathcal{Q}^{(i)} = \mathcal{Q}^{(i-1)} \setminus B'$ // `new set` $\mathcal{Q}$
8: **if** $\mathcal{Q}^{(i)} = \emptyset$ **then**
9: return Δ^2 // `diameter has been found`

Algorithm 5: Iterative reduction of $\mathcal{Q}$ by successive examination of all segments $p_i q_i$.

5.2.2 Algorithm

Assume that Algorithm 2 terminates under case 2, yielding the segment $p_{max}q_{max}$ (i.e. $p_{I-1}q_{I-1}$) of length $\Delta = \delta(p_{max}, q_{max})$ which is considered as an estimation of the diameter. Moreover, we assume that the set $\mathcal{Q}$ computed by Algorithm 3 is not empty.

All the double normals $p_i q_i$ that have been found by Algorithm 2, except $p_{I-1}q_{I-1}$, are collected into a set $\mathcal{S}$.

If Algorithm 5 terminates with $\mathcal{Q}^{(|S|)} \neq \emptyset$, one still must run Algorithm 4 with $\mathcal{Q} = \mathcal{Q}^{(|S|)}$, i.e. the exhaustive search over $\mathcal{Q}^{(|S|)} \times \mathcal{P}$.

6 Diameter approximation

Our algorithm provides a lower bound $\Delta \stackrel{\text{def}}{=} \Delta_{\min}$ on the diameter. It also provides an upper bound $\Delta_{\max} = \Delta_{\min}\sqrt{3}$. Indeed, let pq be the double normal whose length is $\Delta_{\min}$. All the points of $\mathcal{P}$ belong to the intersection of the two balls of radius $\Delta_{\min}/2$ centered at p and q.

With only slight modifications, our algorithm can also be used to compute a better approximation of the diameter. More precisely, for any given ε, we provide an interval $[\Delta_{\min}, \Delta_{\max}]$ of length $\leq \varepsilon$ that contains the true diameter.

Since the algorithm provides a lower bound Δ, we simply need to ensure that $\Delta + \varepsilon$ is an upper bound of the true diameter.

We will just indicate where the necessary modifications must take place.

First, during the iterative search of double normals (line 9 in Algorithm 2) the ball centered at $\frac{p+q}{2}$ and passing through the furthest point m contains all the points of $\mathcal{P}$. The diameter $\Delta_{\max}$ of that ball is given by

$$\Delta^2_{\max} = 4\,\overrightarrow{mp}.\overrightarrow{mq} + \Delta^2$$

where $\Delta = \delta(p,q)$. Therefore, when $\Delta^2_{\max} \leq (\Delta+\varepsilon)^2$, we have found an ε-approximation of the diameter and we stop.

Second, the intermediate step (Algorithm 3) checks if $\mathcal{P}^{(I)}$ contains the endpoint of some potential maximal segment. Here the set $\mathcal{Q}$ has to be replaced by $\mathcal{P}^{(I)} \setminus B\left(\frac{p_{I-1}+q_{I-1}}{2}, \frac{\Delta+\varepsilon}{2}\right)$.

A better estimate than $\Delta+\varepsilon$ of the upper bound $\Delta_{\max}$ is obviously

$$2 \times \max_{q \in \mathcal{R}} \delta\left(\frac{p_{I-1}+q_{I-1}}{2}, q\right) \quad \text{with} \quad \mathcal{R} = \mathcal{P}^{(I)} \cap \left\{ B\left(\frac{p_{I-1}+q_{I-1}}{2}, \frac{\Delta+\varepsilon}{2}\right) \setminus B\left(\frac{p_{I-1}+q_{I-1}}{2}, \frac{\Delta}{2}\right) \right\}.$$

If $\mathcal{Q}$ is empty, we stop.

The exhaustive search over $\mathcal{Q} \times \mathcal{P}$ described in Algorithm 4 will possibly update both Δ and $\Delta_{\max}$.

Required: Δ^2 provided by algorithm 2,
Required: $\mathcal{Q}$, $\Delta^2_{\max}$ and provided by modified algorithm 3
1: **if** $\mathcal{Q} \neq \emptyset$ **then** // `Exhaustive search with an endpoint in` $\mathcal{Q}$
2: **for all** points $p_i \in \mathcal{Q}$ **do**
3: **for all** points $p_j \in \mathcal{P}$ **do**
4: **if** $\delta^2(p_i,p_j) > \Delta^2$ **then**
5: $\Delta^2 = \delta^2(p_i,p_j)$
6: **if** $\delta^2(p_i,p_j) > \Delta^2_{\max}$ **then**
7: $\Delta^2_{\max} = \delta^2(p_i,p_j)$
8: **return** Δ^2 and $\Delta^2_{\max}$

Algorithm 6: Modified exhaustive search over $\mathcal{Q} \times \mathcal{P}$.

Finally, in Algorithm 5 (line 2), we will use $\Delta_{\max}$ instead of Δ and update both Δ^2 and $\Delta^2_{\max}$ when necessary (lines 4-6).

7 Experimental results

In this section, we present experimental results with both the exact and the approximate algorithms. In all cases, we run the method with and without the reduction of $\mathcal{Q}$ described in section 5.2.

7.1 Point distributions in $\mathbb{R}^d$

In our experiments, we use several point distributions. For each of them, the limit value of the diameter (for an infinite number of points) is 1. O denotes the origin.

Volume based distributions

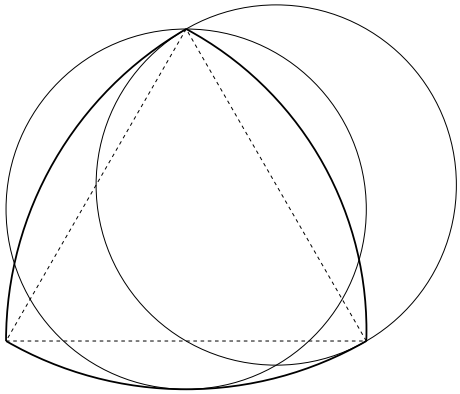


Figure 2: In bold, a 2D shape of constant width D (a Reuleaux triangle) and two circles of diameter D.

Points in a cube. The points are chosen independently from a uniform distribution in $\left[-1/\left(2\sqrt{d}\right), 1/\left(2\sqrt{d}\right)\right]^d$.

For a large number of points, the maximal segment of the point set is close to a maximal segment of the cube. The number of maximal segments of the cube is 2^{d-1} and therefore increases dramatically with the space dimension.

Points in a ball. The points are chosen independently from a uniform distribution in the ball of radius $1/2$ centered at O.

Points in a set of constant width ($d = 2$). The points are chosen independently from a uniform distribution in a domain of constant width 1 centered at O.

Such a domain is built from a regular polygon with $2p+1$ vertices (see figure 2) inscribed in a circle of radius $1/\sqrt{2\left(1-\cos\frac{2\pi p}{2p+1}\right)}$.

Surface based distributions

Points on a sphere. The points are chosen independently from a uniform distribution on the sphere of radius $1/2$ centered at O.

Points on an ellipsoid. The points are chosen independently from a uniform distribution on the surface of an ellipsoid centered at O. The ellipsoid axes are aligned with the coordinate axes.

The largest axis of the ellipsoid is set to $1/2$ while the others are chosen independently from a uniform distribution in $[1/10, 1/2]$.

Points on a regular ellipsoid. The points are chosen independently from a uniform distribution on the surface of an ellipsoid centered at O. The ellipsoid axes are aligned with the coordinate axes.

The largest axis $r_0 = r_{max}$ is set to $1/2$, the smallest one $r_{n-1} = r_{min}$ is set to $1/10$, while the other axes r_i for $i = 1, \ldots, (n-2)$, are chosen independently from a uniform distribution in $\left[r_{max} - \left(i + \frac{1}{2}\right) \frac{r_{max} - r_{min}}{n-1}, r_{max} - \left(i - \frac{1}{2}\right) \frac{r_{max} - r_{min}}{n-1}\right]$. Notice that the axes are taken from non-overlapping intervals.

These ellipsoids cannot have an infinite number of maximal double normals. When the number of points is large enough, the maximal segment of the point set is close to the main axis of the ellipsoid.

7.2 Performance evaluation

The only numerical operations used by our algorithms are dot products. We therefore evaluate the complexity of all the proposed methods by measuring the ratio between the number of dot products and the number of points. As an example, the complexity of the brute-force algorithm that considers all $n(n-1)/2$ segments is $(n-1)/2$.

We also give CPU times in seconds. The program is written in C, compiled with `cc` using the `-O` option and runs on a DEC station PWS (500 MHz).

7.3 Exact computation

7.3.1 Point sets in $\mathbb{R}^3$

Figures 3 and 4 shows the complexity of the method with and without reduction of $\mathcal{Q}$ for various 3D point distributions. Several remarks can be made.

- The algorithm performs quite well (it is almost linear) for the cube and the regular ellipsoid.
- It performs a little bit worse for the general ellipsoid, with a more chaotic complexity. See the discussion in section 7.3.1.
- The complexity is bad for balls and spheres. In the case of points distributed on a sphere, it is close (but still better) to the complexity of the brute-force method. This is due to the fact that the first set $\mathcal{Q}$ contains about 50% of the points of $\mathcal{P}$.
- Interestingly, computing double normals is always very fast. The overall computing time is dominated by the exhaustive examination of all segments of $\mathcal{Q} \times \mathcal{P}$.

Table 1 gives CPU times (in seconds) for the same examples as in figure 4.

	Method with reduction of $\mathcal{Q}$					Brute-force [a]
	3D Points distribution [b]					
	Volume		Surface			
n	cube	sphere	reg. ellipsoid	ellipsoid	sphere	
10000	0.010	0.039	0.006	0.012	2.74	4.65
20000	0.019	0.110	0.013	0.031	10.97	19.25
30000	0.029	0.246	0.021	0.043	25.6	43.7
40000	0.038	0.322	0.028	0.059	47.58	78
50000	0.048	0.413	0.035	0.092	71.77	122.1
60000	0.058	0.625	0.043	0.097	104.53	176.1
70000	0.068	0.916	0.050	0.099	144.28	239.3
80000	0.077	1.022	0.058	0.129	184.77	313
90000	0.088	1.026	0.065	0.128	234.01	396.9
100000	0.095	1.380	0.074	0.160	286.84	492
1000000	1.873		1.562	3.482		

[a] For $10,000 \leq n \leq 30,000$, average times have been estimated with 100 trials, for $40,000 \leq n \leq 90,000$, average times have been estimated with 10 trials, only one trial was performed for $n = 100,000$.

[b] 1000 (resp. 100) trials were used to estimate the average time for the cube and ellipsoids (resp. spheres) distributions.

Table 1: Exact computation of the diameter with reduction of both $\mathcal{Q}$ and $\mathcal{P}$ compared to the brute-force algorithm for various 3D point distributions.

The ellipsoid case (in $\mathbb{R}^3$)

The fact that the complexity varies widely for point distributed on both types of ellipsoids can be explained using more detailed experiments. Instead of randomly picking the axes of the ellipsoids, we let them vary continuously. More precisely, the largest axis $R_{\max}$ being set to 1/2 (see section 7.1), we study the behaviour of our method with respect to the ratio $R_{\text{med}}/R_{\max}$ and $R_{\min}/R_{\text{med}}$ where $R_{\min}$ and R_{med} are respectively the smallest and the medium axes.

We observe that, for almost all $R_{\text{med}}/R_{\max}$ and $R_{\min}/R_{\text{med}}$, the size of $\mathcal{Q}$ remains small and comparable to what we get in the case of the regular ellipsoid. The only exception is for points on a sphere (i.e., when the three axes are equal) and when $R_{\text{med}}/R_{\max}$ is close to 1.

Interestingly, the number of FP scans needed to compute a double normal increases with the ratio $R_{\text{med}}/R_{\max}$, but falls down for $R_{\text{med}}/R_{\max} = 1$. This is related to the construction described in the proof of lemma 5 where we had points close but not on a circle, leading to a large number of FP scans.

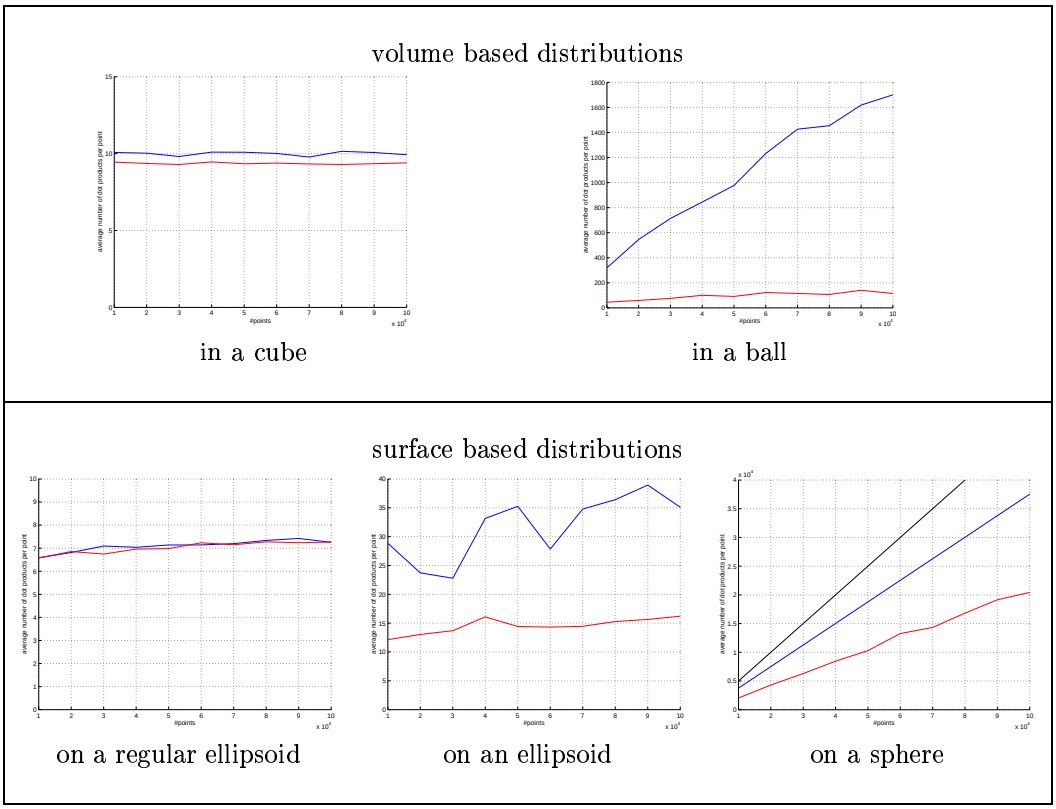


Figure 3: Exact computation of the diameter : complexity measures for various distributions in $\mathbb{R}^3$. Blue line: with no reduction of $\mathcal{Q}$. Red line: with reduction of $\mathcal{Q}$. Black line (when displayed): complexity of the brute-force method (i.e. $(n-1)/2$). For the ellipsoids and the cube (resp. the ball and the sphere) each plotted point was obtained by averaging 1000 trials (resp. 100 trials).

Sets of constant width (in $\mathbb{R}^2$)

Objects of constant width are easily constructed from regular polygons with an odd number of vertices (see figure 2). The object obtained from a regular triangle is the celebrated Reuleaux triangle. We select points from a uniform distribution inside such objects and compare the complexity with the one obtained for points distributed in a 2D ball (see figure 6).

Assume that the longest segment returned by Algorithm 2 is pq. In case of points distributed in a ball, $B[pq]$ covers most of the set $\mathcal{P}$ while, in case of a Reuleaux triangle, a larger part of $\mathcal{P}$ is not covered by $B[pq]$ (see figure 2).

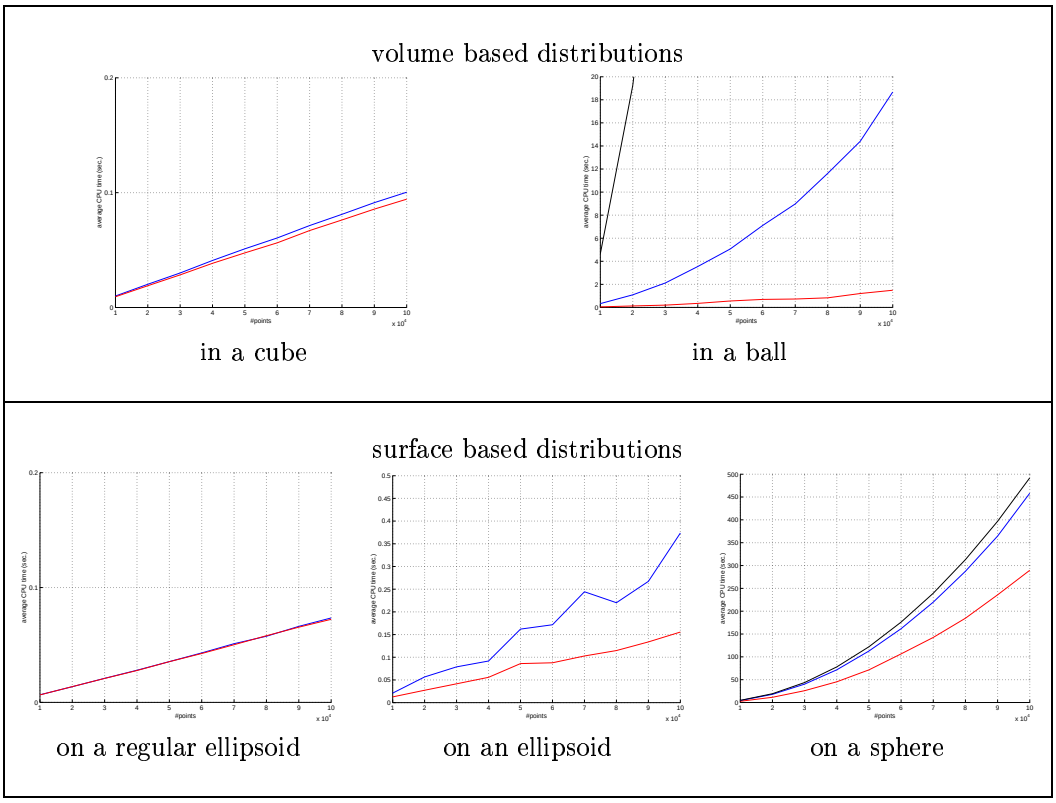
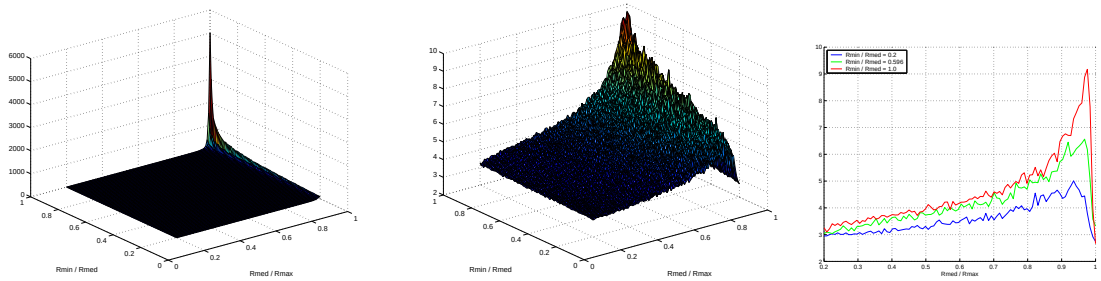


Figure 4: Exact computation of the diameter : CPU time (in seconds) for various distributions in $\mathbb{R}^3$. Blue line: with no reduction of $\mathcal{Q}$. Red line: with reduction of $\mathcal{Q}$. Black line (when displayed): brute-force method. For the ellipsoids and the cube (resp. the ball and the sphere) each plotted point was obtained by averaging 1000 trials (resp. 100 trials).

As a result, the number of points outside $B[pq]$ for these distributions is larger than for the ball and decreases when the number of vertices of the polygons increases. Similarly, the complexity of our algorithm for such point distributions decreases when the number of vertices increases and tends to be the same as for points in a ball.

7.3.2 Point sets in $\mathbb{R}^d$

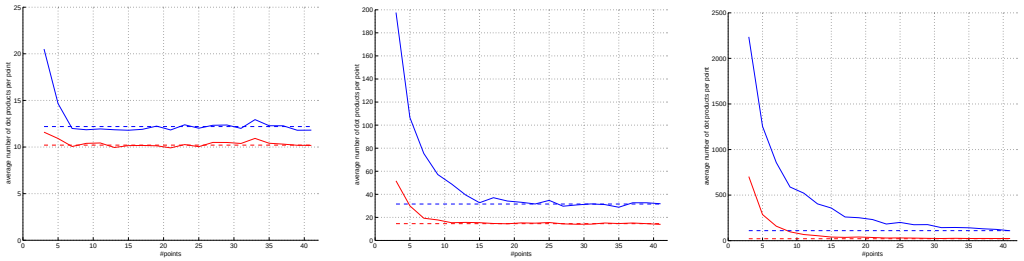
In figure 7, we provide experimental results in $\mathbb{R}^d$ and evaluate the complexity as a function of the dimension d.



Average number of remaining points outside the ball $B[pq]$

Average number of FP scans needed to compute a double normal

Figure 5: Some statistics for sets of 10 000 points on the surface of a 3D ellipsoid. 100 trials were performed for each set of axes.



100 points 1000 points 10000 points

Figure 6: 2D distributions inside domains of constant width. The figure shows the complexity as a function of the number of vertices of the polygons. Blue line: with no reduction of $\mathcal{Q}$. Red line: with reduction of $\mathcal{Q}$. Dash lines: the complexity for a 2D distribution inside a ball.

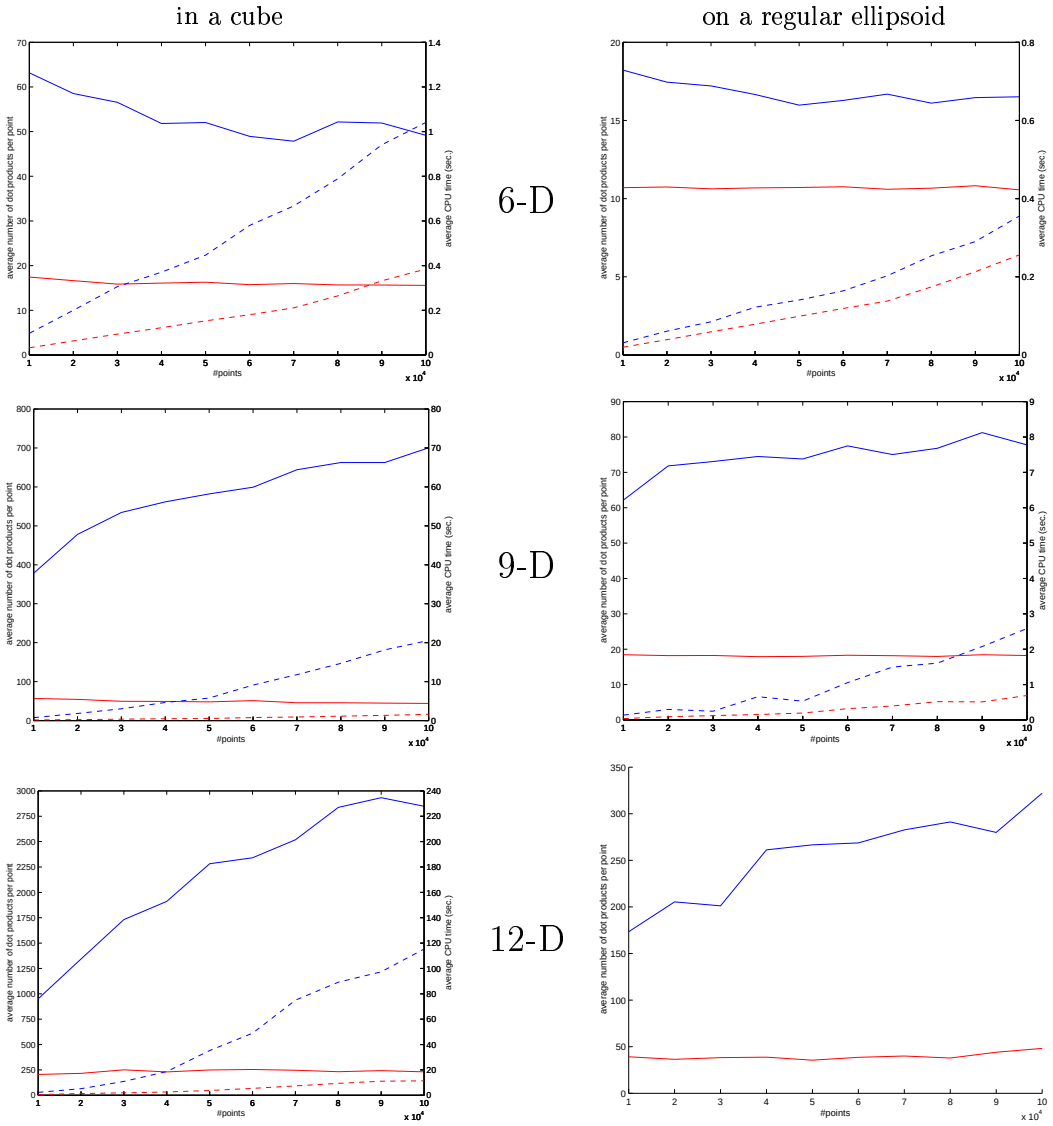


Figure 7: Exact computation of the diameter : complexity measures and CPU time (in seconds) for distributions in $\mathbb{R}^d$, $d \in \{6, 9, 12\}$. Blue lines: with no reduction of $\mathcal{Q}$. Red lines: with reduction of $\mathcal{Q}$. Continuous lines: complexity measure (left Y axis). Dashed lines: CPU time (right Y axis).

It should be observed that the complexity of computing a double normal is always a small constant. Differently, the complexity of computing the diameter increases rapidly with d. This is due to the increasing number of points in $\mathcal{Q}$.

7.4 Diameter approximation

Figures 8 and 9 shows the complexity of the method with and without reduction of $\mathcal{Q}$ for various 3D point distributions. These figures have to be compared to figures 3 and 4.

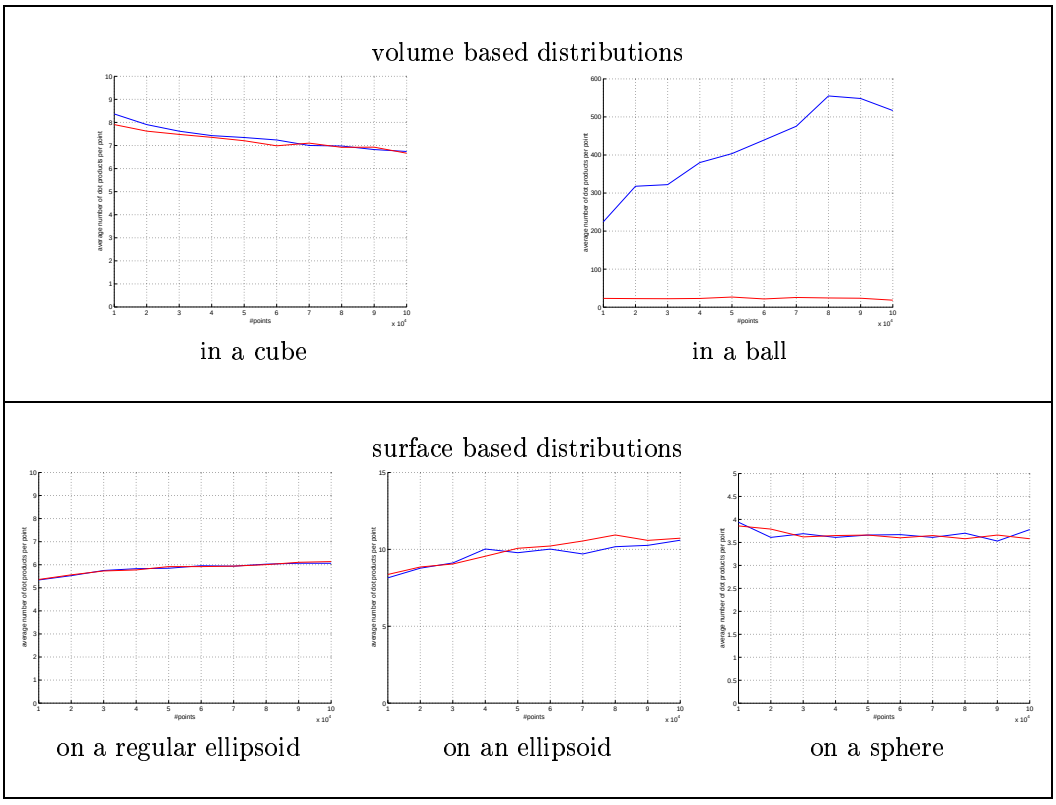


Figure 8: Approximate computation of the diameter ($\varepsilon = 0.01$) : complexity measures for various distributions in $\mathbb{R}^3$. Blue line: with no reduction of $\mathcal{Q}$. Red line: with reduction of $\mathcal{Q}$. Black line (when displayed): complexity of the brute-force method (i.e. $(n-1)/2$). For the ellipsoids and the cube (resp. the ball and the sphere), each plotted point was obtained by averaging 1000 trials (resp. 100 trials).

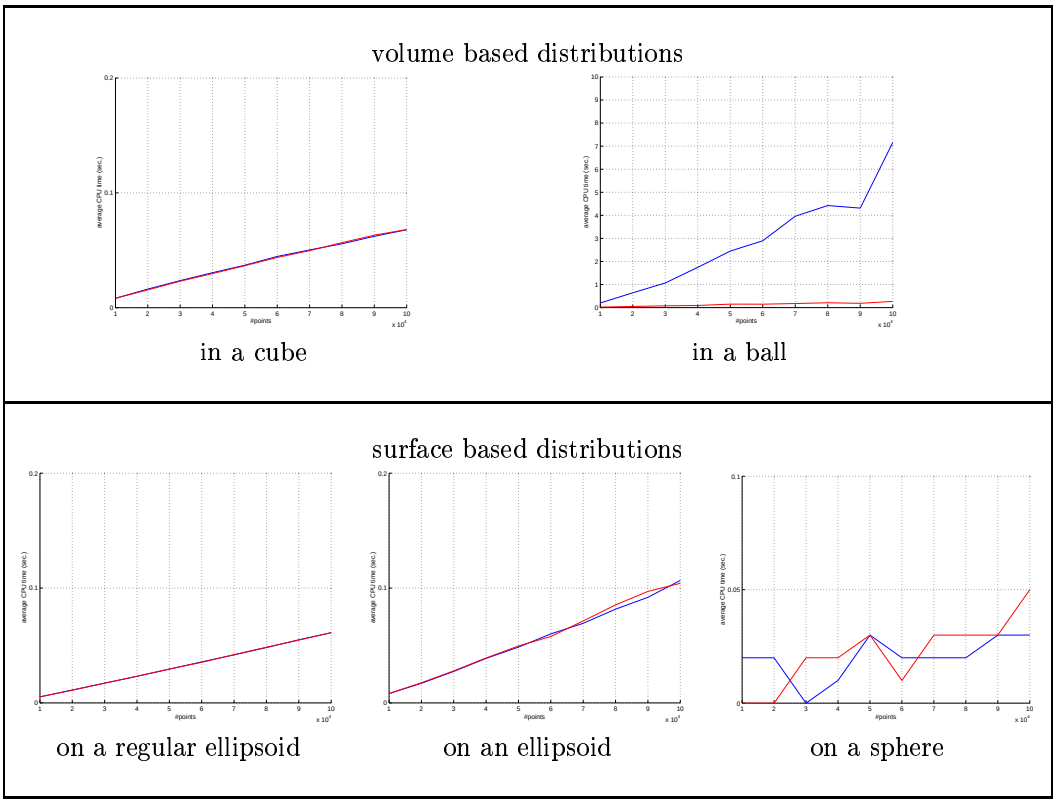


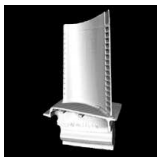
Figure 9: Approximate computation of the diameter ($\varepsilon = 0.01$) : CPU time (in seconds) for various distributions in $\mathbb{R}^3$. Blue line: with no reduction of $\mathcal{Q}$. Red line: with reduction of $\mathcal{Q}$. Black line (when displayed): brute-force method. For the ellipsoids and the cube (resp. the ball and the sphere) each plotted point was obtained by averaging 1000 trials (resp. 100 trials).

7.5 Real 3D objects

We present now results on real data from the Large Geometric Models Archive [lar].







Inputs	Bunny	Hand	Dragon	Buddha	Blade
Points	35,947	327,323	437,645	543,652	882,954
Diameter	0.198339	7.01781	0.210876	0.208685	669.508
Exact computation of the diameter					
Method with no reduction of $\mathcal{Q}$					
complexity	1079.37	4.79996	2121.69	2664.49	6.41997
CPU time	3.98	0.3	171.08	268.94	1.06
Method with reduction of $\mathcal{Q}$					
complexity	1073.52	4.84996	115.892	2165.14	6.06998
CPU time	3.98	0.29	7.69	217.42	1.05
Approximation of the diameter (with $\varepsilon = 0.01 \times$ diameter)					
Method with no reduction of $\mathcal{Q}$					
complexity	816.643	4.94996	1542.34	1378.25	6.27998
CPU time	3.09	0.3	125.66	139.17	1.03
Method with reduction of $\mathcal{Q}$					
complexity	816.577	4.85996	81.1558	1139.12	6.42997
CPU time	3.08	0.3	5.36	118.05	1.02

Table 2: Complexities and times for various models. Each value is an average of 100 trials. As already pointed out, the total time is mainly due to the remaining points in $\mathcal{Q}$.

8 Comparison with Har-Peled's method

The most comparable approach to ours is the one developed very recently by S. Har-Peled [Har01]. Although it is similar in spirit, Har-Peled's algorithm is quite different from ours. We first summarize his method and then compare experimentally the two methods. Since the two methods have different advantages and drawbacks, it is worth combining them, leading to good hybrid algorithms with more stable performances.

In his approach, Har-Peled recursively computes pairs of boxes (each enclosing a subset of the points). He throws away pairs that cannot contain a maximal segment.

	Running time in seconds				
Inputs	Bunny	Hand	Dragon	Buddha	Blade
Points	35,947	327,323	437,645	543,652	882,954
our method	5.73	0.29	8.51	172.91	0.49
Har-Peled's method (HPM) - original	0.08	0.45	0.90	0.72	1.00
HPM - our implementation	0.07	0.43	0.89	0.69	0.94
hybrid method #1	0.07	0.41	0.86	0.67	0.90
hybrid method #2	0.10	0.32	1.37	1.09	0.50

Table 3: CPU times on real inputs.

To avoid maintaining too many pairs of boxes, Har-Peled does not decompose a pair of boxes if both contain less than $n_{\min}$ points (initially set to 40 in Har-Peled's implementation). Instead, he computes the diameter between the two corresponding subsets using the brute-force method. Moreover, if the number of pairs of boxes becomes too large during the computation (which may be due to a large number of points or to the high dimension of the embedding space), $n_{\min}$ can be doubled: however, doubling $n_{\min}$ increases the computing time.

Differently from our method, Har-Peled's algorithm depends on the coordinate axes (see table 5).

We provide an experimental comparison of both approaches, using the original Har-Peled's implementation[1] which only works for 3D inputs. In order to be able to deal with inputs in higher dimensions, we have re-implemented his algorithm, following the same choices that were made in the original implementation.

8.1 Hybrid methods

It should be first notice that both methods can easily be modified to compute the diameter between two sets, *i.e.* the segment of maximal length with one endpoint in the first set and the other in the second set.

Both methods have quadratic parts. Ours with the final computation over $\mathcal{Q} \times \mathcal{P}$, and Har-Peled's one when computing the diameter for a pair of *small* boxes.

We have implemented two hybrid methods that combines Har-Peled's method and ours. We first modified Har-Peled's algorithm by replacing each call to the brute-force algorithm by a call to our algorithm. We also tested another hybrid method where we modified our algorithm by replacing the final call to the brute-force algorithm by a call to the first hybrid method. The two hybrid methods can be tuned by setting several parameters. The experimental results presented here have been obtained with the same values of the parameters.

The results show that the hybrid methods are never much worse than the best method. Moreover, their performances are more stable and less sensitive to the point distribution.

[1] Available at `http://www.uiuc.edu/ sariel/papers/00/diameters/diam_prog.html`.

	Running time in seconds 3D Volume Based distributions					
Inputs	Cube	Cube	Cube	Ball	Ball	Ball
Points	10,000	100,000	1,000,000	10,000	100,000	200,000
our method	0.01	0.19	0.53	0.04	0.79	1.20
HPM - original	0.01	0.18	1.96	0.31	18.16	53.88
HPM - our implementation	0.02	0.18	1.92	0.20	5.12	20.57
hybrid method #1	0.01	0.18	2.00	0.13	2.25	5.26
hybrid method #2	0.02	0.35	1.50	0.07	1.05	3.29

Table 4: CPU times for 3D volume based synthetic distributions.

	Running time in seconds 3D Surface Based distributions					
Inputs	Ellipsoid (regular)	Ellipsoid	Ellipsoid (rotated)	Sphere	Sphere	Sphere
Points	1,000,000	1,000,000	1,000,000	10,000	100,000	200,000
our method	1.34	2.02	1.61	1.08	358.21	not computed
HPM - original	1.78	3.84	37.70	2.13	95.49	328.90
HPM - our implementation	1.81	3.51	23.88	0.63	39.97	166.26
hybrid method #1	1.82	3.38	6.38	0.33	6.99	16.75
hybrid method #2	2.30	3.10	1.79	0.44	8.58	19.75

Table 5: CPU times for 3D surface based synthetic distributions. The points sets in the second and the third columns are identical up to a 3D rotation.

	Running time in seconds				
	volume distributions		surface distributions		
Inputs	Cube	Ball	Regular Ellipsoid	Ellipsoid	Sphere
Points	100,000	100,000	100,000	100,000	100,000
	Dimension = 6				
our method	0.31	36.95	0.11	0.33	not computed
HPM - our implementation	0.85	466.44	0.97	0.87	465.08
hybrid method #1	0.67	77.20	0.79	0.73	118.06
hybrid method #2	0.66	63.31	0.19	0.65	142.38
	Dimension = 9				
our method	0.89	128.02	0.51	0.52	not computed
HPM - our implementation	139.23	568.99	264.96	590.14	569.08
hybrid method #1	17.42	135.90	44.54	67.27	232.39
hybrid method #2	1.21	121.91	1.25	16.03	302.86
	Dimension = 12				
our method	3.87	445.03	1.08	7.88	not computed
HPM - our implementation	629.37	651.56	648.88	650.98	647.74
hybrid method #1	44.45	354.14	58.53	56.11	511.41
hybrid method #2	19.72	380.41	13.00	24.62	745.60
	Dimension = 15				
our method	10.99	798.66	7.26	20.31	not computed
HPM - our implementation	734.69	735.26	731.76	733.70	737.51
hybrid method #1	64.49	610.70	69.11	90.35	701.18
hybrid method #2	44.37	782.20	21.30	70.41	1120.57

Table 6: CPU times for synthetic distributions in higher dimensions.

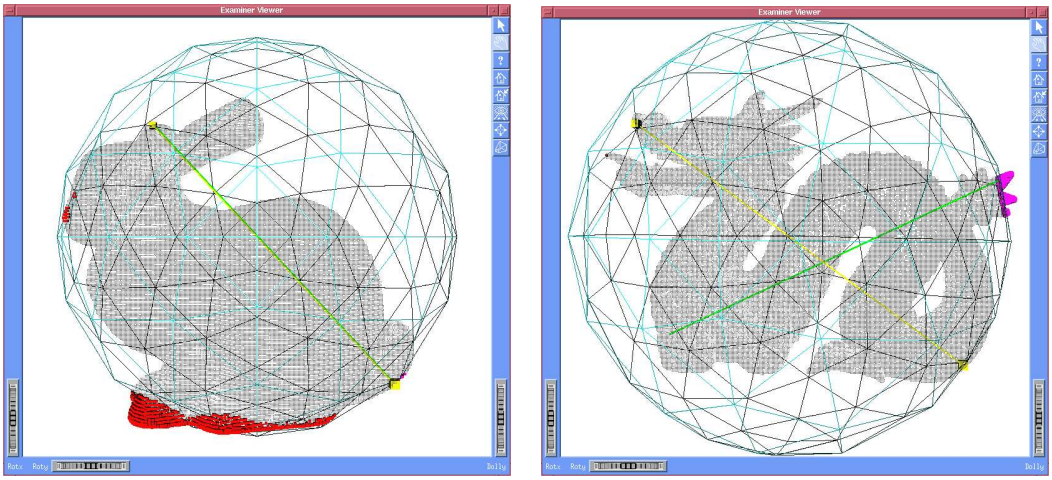


Figure 10: In red: the points remaining for the final quadratic part (the search over $\mathcal{Q} \times \mathcal{P}$); in purple: the points eliminated during the reduction of $\mathcal{Q}$; in yellow: the maximal segment (the corresponding sphere is displayed too); in green: one other double normal.

9 Discussion

Our method is based on the computation of double normals. Computing a double normal appears to be extremely fast under any practical circumstances and in any dimension (despite the quadratic lower bound of lemma 5). Moreover, the reported double normal is very often the true maximal segment. This is not too much surprising since, on a generic surface, the number of double normals is finite and small. In any case, having a double normal provides a $\sqrt{3}$-approximation of the diameter in any dimensions.

However, even if the reported double normal is a maximal segment s, it may be costly to verify that this is indeed the case. A favourable situation is when the point set is contained in the ball B of diameter s. The bad situation occurs when there are many points in set $\mathcal{P} \setminus B$ since we verify that none of these points is the endpoint of a maximal segment. This case occurs with sets of constant width (see section 7.1) but also with some real models: e.g. bunny, dragon and buddha (see tables 2 and 3).

For these three cases, the first double normal found by the algorithm was the maximal segment (in yellow in figure 10). The second found double normal was shorter (in green). After Algorithm 3, $\mathcal{Q}$ contains respectively 1086, 2117, and 2659 points for the bunny, dragon, and buddha models. For both the bunny and the buddha, the second double normal was very close to the first one, then very few points were removed from $\mathcal{Q}$ (respectively 7 and 36), and most of the points of $\mathcal{Q}$ will undergo the final quadratic search. This explains

why there is a so little difference between our method with and without the reduction of $\mathcal{Q}$ for these two models (see table 2).

For the dragon model, the second double normal is quite different from the first one, hence the noticeable improvement of our method with the reduction of $\mathcal{Q}$.

Har-Peled's method does not suffer from this drawback. However, it depends on the coordinate axes (since the boxes are aligned with the axes) and on the dimension d of the embedding space.

The first hybrid method compensates for the quadratic search between *small boxes* (boxes containing less than $n_{\min}$ points), i.e. one major drawback of original Har-Peled's method.

The second hybrid method compensates for the major drawback of our method, by building pairs of boxes from $\mathcal{Q} \times \mathcal{P}$.

References

[AGR94] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 683–694, 1994.

[Bes98] S. Bespamyatnikh. An efficient algorithm for the three-dimensional diameter problem. In *Proc. 9th Annu. ACM-SIAM Symp. Discrete Algorithms*, pages 137–146, 1998.

[CS89] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry. *Discrete Comput. Geom.*, 4:387–421, 1989.

[Har01] S. Har-Peled. A practical approach for computing the diameter of a point-set. pages 177–186, 2001.

[HP] Sariel Har-Peled. A practical approach for computing the diameter of a point-set. `http://valis.cs.uiuc.edu/~sariel/papers/00/diam.html`.

[lar] Large geometric models archive. `http://www.cs.gatech.edu/projects/large_models/`. Georgia Institute of Technology.

[PS90] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction.* Springer Verlag, October 1990. 3rd edition.

[Ram97a] E. Ramos. Construction of 1-d lower envelopes and applications. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 57–66, 1997.

[Ram97b] E. Ramos. Intersection of unit-balls and diameter of a point set in R^3. *Comput. Geom. Theory Application*, 8:57–65, 1997.

[Ram00] Edgar A. Ramos. Deterministic algorithms for 3-D diameter and some 2-D lower envelopes. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 290–299, 2000.

A Geometrical properties and computations

A.1 Deciding if a point belongs to a ball

Locating a point m with respect to a ball $B[pq]$ of diameter pq requires a single dot product. Indeed, we have

$$\begin{cases} \overrightarrow{mp}.\overrightarrow{mq} < 0 & \Longleftrightarrow & m \text{ is inside } B \\ \overrightarrow{mp}.\overrightarrow{mq} = 0 & \Longleftrightarrow & m \text{ is on the boundary of } B \\ \overrightarrow{mp}.\overrightarrow{mq} > 0 & \Longleftrightarrow & m \text{ is outside } B \end{cases}$$

Developping $\overrightarrow{mp}.\overrightarrow{mq}$ with respect to the middle of the segment $[pq]$ yields

$$\delta^2\left(m, \frac{p+q}{2}\right) = \overrightarrow{mp}.\overrightarrow{mq} + \frac{\delta^2(p,q)}{4}$$

Thus, the computation of $\overrightarrow{mp}.\overrightarrow{mq}$ enables us to classify all the points with respect to any ball centered at the middle of segment pq and gives us additionnaly the squared distance to this particular point.

This dot product is then used

- to compute the furthest point from a sphere $\Sigma[pq]$ (line 9 of algorithm 2)
- to sort points with respect to a sphere (line 7 of algorithm 2, line 1 of algorithm 3, and lines 7 and 3 of algorithm 5).

A.2 Removing points from $\mathcal{P}$

Assume that we have at our disposal an approximation Δ of the diameter of set $\mathcal{P}$ and a subset $\mathcal{Q} \subset \mathcal{P}$ that contains an endpoint of each maximal segment longer than Δ (plus possibly other points of $\mathcal{P}$).

The principle of our method is to keep the number of points in $\mathcal{Q}$ as small as possible, since these points will finally have to be compared against all points of $\mathcal{P}$ (algorithm 4).

We may also reduce the complexity due to this exhaustive search by removing from $\mathcal{P}$ those points that can not be endpoints of any segment larger than Δ.

This is explained, from a geometrical point of view, in the next section. Algorithms 2, 3, and 5 can easily be modified to incorporate these modifications.

However, we did not observe a significant improvement in our experiments, then we did not present the results obtained with the reduction of $\mathcal{P}$.

A.2.1 A bit of geometry

If pq is a double normal of $\mathcal{P}$ and $\mathcal{I}$ is the set in light grey in figure 11, we have :

$$\mathcal{P} \subset \mathcal{I} = B(p, \delta(p,q)) \cap B(q, \delta(p,q)).$$

The points of $\mathcal{I}$ that cannot be endpoints of a segment of length larger or equal to Δ are included in all the balls of radius Δ whose center lies in the set. In fact, we can reduce the locus of the centres to be considered to the intersection $\Sigma(p, \delta(p,q)) \cap \Sigma(q, \delta(p,q))$.

Thus the points that can be eliminated belong to

$$\mathcal{E} = \bigcap B(c, \Delta) \quad \text{for } c \in \Sigma(p, \delta(p,q)) \cap \Sigma(q, \delta(p,q)$$

In the plane, there are only two such balls $B(c, \Delta)$ (dash line in figure 11) but there are an infinity of such balls in higher dimensions. The intersection $\mathcal{E}$ is the dark grey region in figure 11.

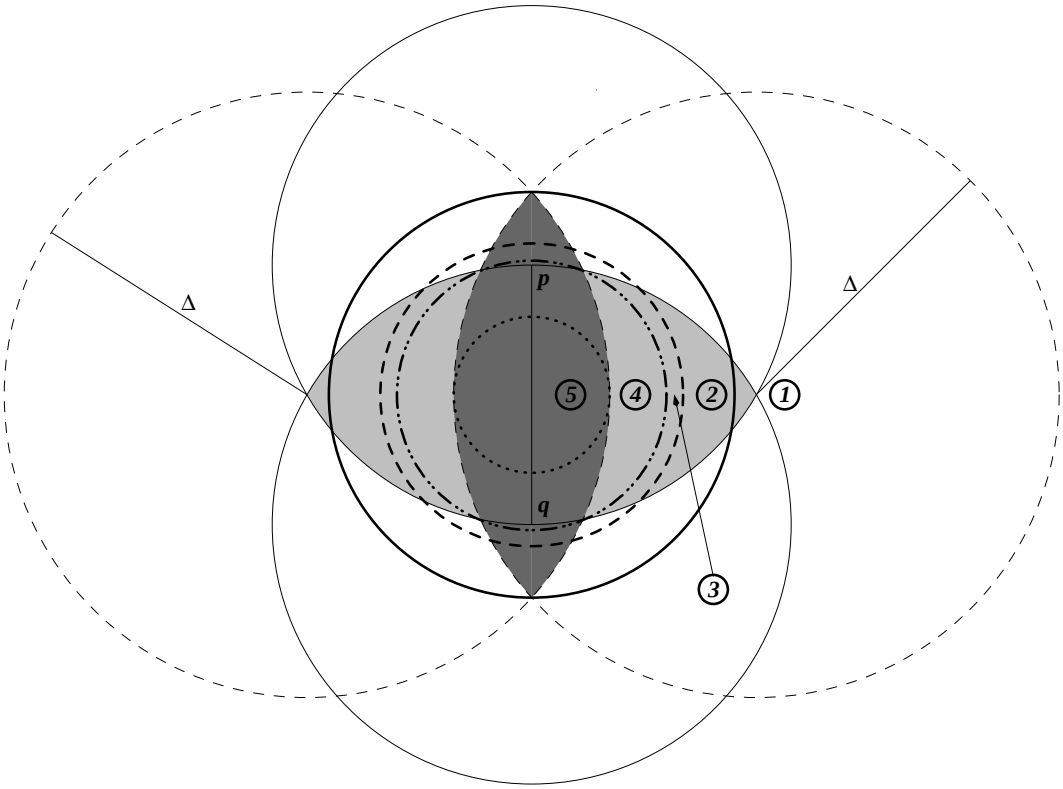


Figure 11: Geometrical construction (see text for details).

Some calculations sjow that the points m of this intersection can be characterized by two conditions

$$\begin{cases} \overrightarrow{mp}.\overrightarrow{mq} \leq \Delta^2 - \delta^2(p,q) \\ 3\left(\delta^2(p,m)\delta^2(p,q) - \left(\delta^2(p,m) - \overrightarrow{mp}.\overrightarrow{mq}\right)^2\right) \leq \left(\Delta^2 - \delta^2(p,q) - \overrightarrow{mp}.\overrightarrow{mq}\right)^2 \end{cases}$$

which can be evaluated by means of only two dot products: $\overrightarrow{mp}.\overrightarrow{mq}$ and $\overrightarrow{mp}.\overrightarrow{mp}$.

It should be pointed out that the first condition expresses nothing but the fact that m is inside a ball of centre $\frac{p+q}{2}$ and radius

$$R_{1,2} = \sqrt{\Delta^2 - 3\delta^2(p,q)/4}$$

(bold continuous line in figure 11).

Having a closer look at figure 11 yields two additionnal remarks:

- The portion of $\mathcal{E}$ (points that can be eliminated) inside $\mathcal{I}$ (overset containing $\mathcal{P}$) is inside the ball (bold dash-dot-dot line in figure 11) of center $\frac{p+q}{2}$ and radius $R_{3,4}$ which can be written (after some tedious calculations)

$$R_{3,4} = \sqrt{\frac{3}{4}\delta^2(p,q) + \frac{1}{4}\Delta^2 - \frac{\sqrt{3}}{4}\delta(p,q)\sqrt{4\,\delta^2(p,q) - \Delta^2}}$$

  this radius is defined by the distance of $\frac{p+q}{2}$ to the intersection of the boundaries of $\mathcal{E}$ and of $\mathcal{I}$.

- As $\mathcal{E}$ contains the point $\frac{p+q}{2}$, we can compute the maximal radius $R_{4,5}$ such that $B\left(\frac{p+q}{2}, R_{4,5}\right)$ is the largest ball centered at $\frac{p+q}{2}$ that is included in $\mathcal{E}$ (bold dot line in figure 11). We get

$$R_{4,5} = \Delta - \frac{\sqrt{3}}{2}\delta(p,q)$$

Finally, we had in figure 11 the ball of centre $\frac{p+q}{2}$ and radius

$$R_{2,3} = \Delta/2$$

(bold dash line in figure 11), outside which we will search for diameter endpoints (see section 5.2.1).

This way, we have defined several concentric balls (bold lines) of radii $R_{i,j}$ which are the boundaries of several regions ⓚ (see figure 11).

A.2.2 Point removal principle

The idea is to try to remove points without increasing the complexity too much, i.e. adding only a few additional dot products.

Assuming we have a segment pq and an estimation Δ of the diameter, the squared radius $R^2_{i,j}$ can easily be precomputed.

The dot product $\overrightarrow{mp}.\overrightarrow{mq}$ is used several times in the method to compute the furthest point from a sphere, or to sort point with respect to a sphere (see section A.1). It is strictly equivalent to compute the squared distance r^2 of m to the middle of the segment pq, $\frac{p+q}{2}$ because

$$r^2 = \overrightarrow{mp}.\overrightarrow{mq} + \frac{\delta^2(p,q)}{4}$$

Then, any point m can be attributed to the right region ⓚ thanks to the single dot product $\overrightarrow{mp}.\overrightarrow{mq}$, yielding the following scheme:

IF $\overrightarrow{mp}.\overrightarrow{mq} \leq R_{4,5}^2 - \frac{\delta^2(p,q)}{4}$

m belongs to region ⑤ and can be removed from $\mathcal{P}$.

ELSE IF $\overrightarrow{mp}.\overrightarrow{mq} \leq R_{3,4}^2 - \frac{\delta^2(p,q)}{4}$

m belongs to region ④ and can be removed if the additionnal condition

$$3\left(\delta^2(p,m)\delta^2(p,q) - \left(\delta^2(p,m) - \overrightarrow{mp}.\overrightarrow{mq}\right)^2\right) \leq \left(\Delta^2 - \delta^2(p,q) - \overrightarrow{mp}.\overrightarrow{mq}\right)^2$$

is satisfied.

ELSE IF $\overrightarrow{mp}.\overrightarrow{mq} > R_{2,3}^2 - \frac{\delta^2(p,q)}{4}$

m belongs to region ② or ① and is outside the ball of centre $\frac{p+q}{2}$ and radius $\Delta/2$ (is a potential diameter endpoint).

ELSE

m belongs to region ③ and can neither be removed nor is a potential diameter endpoint.

Unité de recherche INRIA Sophia Antipolis
2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399