ELSEVIER

Computational Geometry 8 (1997) 57–65

Computational Geometry
Theory and Applications

Intersection of unit-balls and diameter of a point set in $\mathbb{R}^3$ ☆

Edgar A. Ramos [1]

Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Communicated by David Dobkin; submitted 20 December 1993; accepted 13 March 1996

Abstract

We describe an algorithm for computing the intersection of n balls of equal radius in $\mathbb{R}^3$ which runs in time $O(n \lg^2 n)$. The algorithm can be parallelized so that the comparisons that involve the radius of the balls are performed in $O(\lg^3 n)$ batches. Using parametric search, these algorithms are used to obtain an algorithm for computing the diameter of a set of n points in $\mathbb{R}^3$ (the maximum distance between any pair) which runs in time $O(n \lg^5 n)$. The algorithms are deterministic and elementary; this is in contrast with the running time $O(n \lg n)$ in both cases that can be achieved using randomization (Clarkson and Shor, 1989), and the running times $O(n \lg n)$ and $O(n \lg^3 n)$ using deterministic geometric sampling (Brönnimann et al., 1993; Amato et al., 1994). © 1997 Elsevier Science B.V.

Keywords: Geometric algorithms; Intersection of balls; Diameter of a point set; Parametric search

1. Introduction

Let S be a set of n points in $\mathbb{R}^3$. We consider the problems of (i) computing the intersection of n balls of equal radius centered at the points in S; and (ii) computing the *diameter* of S, the maximum distance between any two points in S.

Clarkson and Shor [3] gave optimal randomized algorithms for both problems which run in time $O(n \lg n)$. They solve the diameter problem through a reduction to computing the intersection of equal radius balls. For the lower bounds, see the text by Preparata and Shamos [13]. Since then, it has been a challenge to match that time complexity with deterministic algorithms. Chazelle et al. [4] gave an algorithm for the diameter problem that runs in time $O(n^{1+\varepsilon})$, where $\varepsilon > 0$ is arbitrary (the constant in the O notation depends on ε); the algorithm uses parametric search and deterministic sampling. Matoušek and Schwarzkopf [10] improved their algorithm to a running time $O(n \lg^c n)$ where c is a constant for which they do not give an explicit bound and which is possibly very large. They did

☆ This research was supported by the National Science Foundation under Grant CCR-9118874.

[1] Current address: DIMACS Center, Rutgers University, P.O. Box 1179, Piscataway, NJ 08855-1179, USA. E-mail: ramose@dimacs.rutgers.edu.

PII S0925-7721(96)00010-7

not consider the ball intersection problem directly, and do not provide an elementary solution. Here, we give a deterministic algorithm for the ball intersection problem that runs in time $O(n \lg^2 n)$, and using parametric search we obtain a deterministic algorithm for the diameter problem that runs in time $O(n \lg^5 n)$. These algorithms are elementary in the techniques they use, and simple as well.

Recently, Brönnimann et al. [2] have given algorithms for these problems that run in time $O(n \lg n)$ and $O(n \lg^3 n)$ respectively. They are based on a complex derandomization technique for geometric sampling. Amato et al. [1] have given alternative algorithms with the same running times, also based on derandomization techniques but somewhat simpler. We believe that despite these developments our algorithms are of interest because they show what can be achieved using elementary methods. It remains an open problem to design an $O(n \lg n)$ time deterministic algorithm for computing the diameter.

In Section 2, we state geometric and combinatorial properties of the intersection of equal radius balls. Section 3 describes the algorithm for intersection of balls, while Section 4 describes how it can be made into a parallel algorithm in a comparison model of computation. Section 5 indicates how parametric search, using the sequential and parallel algorithms for ball intersection, gives an algorithm for the diameter problem.

2. Geometry of ball intersection

Let S be a set of n points in $\mathbb{R}^3$. For $p \in S$, let $b(p,r)$ be the ball of radius r centered at p and let $s(p,r)$ be the bounding sphere of $b(p,r)$ (p and r may be omitted when they are understood or not relevant). Let $B(S,r) = \{b(p,r)\colon\ p \in S\}$, and let $B(S,r)^{\cap} = \bigcap_{p \in S} b(p,r)$ (again, S and r may be omitted). $B^{\cap}$ is a convex body which we call a *spherical polytope*. The boundary of $B^{\cap}$, denoted $\mathrm{bd}(B^{\cap})$, consists of faces, edges and vertices corresponding to the intersection of one, two and three bounding spheres, respectively. We assume that the point set S is in general position so that more than three bounding spheres have empty intersection; this can be achieved using symbolic perturbation [8]. The faces are intersections of circular caps on a sphere (of different radii in general), which we call *spherical polygons*. These faces, edges and vertices form the *boundary graph* of $B^{\cap}$, denoted $\mathcal{G}(B)$. The desired output consists of an appropriate data structure representing $\mathcal{G}(B)$, allowing certain standard adjacency operations in time $O(1)$ (for example, obtaining the next edge bounding the same face in a particular direction, which allows a walk along the boundary of a face in time linear with its length). Here, the *size* of a graph $\mathcal{G}$, denoted $|\mathcal{G}|$, is the total number of faces, edges and vertices. In general, if the radii are not equal, $|\mathcal{G}(B)|$ can be $\Omega(n^2)$. However, $|\mathcal{G}(B)|$ is $O(n)$ for equal radius balls as a result of the following "convexity" property of the faces.

"Convexity" of faces [9]. Let p, q be two points in $\mathrm{bd}(B^{\cap})$ both on the same bounding sphere s (assuming $n \geqslant 2$, they cannot be antipodal). Then the *geodesic* $\widehat{pq}$ connecting p and q on s (a segment of great circle) is also in $\mathrm{bd}(B^{\cap})$. This is the case because if another ball b' contains p, q but not some other point in $\widehat{pq}$, then b' would have a radius greater than that of b.

This implies that for the bounding sphere s of each ball in B, $\mathrm{bd}(B^{\cap}) \cap s$ has at most one connected component. Since the degree of a vertex in $\mathcal{G}(B)$ is exactly three, it follows that $|\mathcal{G}(B)|$ is $O(n)$. The same bound applies for the *dual* graph $\mathcal{G}_D(B)$ (the graph with a vertex corresponding to each face in $\mathcal{G}(B)$, and an edge between two vertices if the corresponding faces in $\mathcal{G}(B)$ are adjacent).

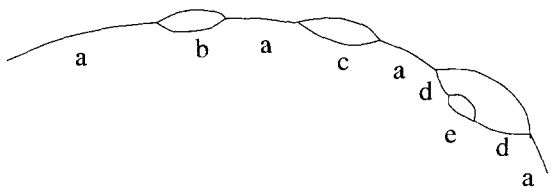

Fig. 1. Multiple edges in a boundary chain.

Hierarchical decomposition. It is well-known that a planar graph has a large independent set of vertices of degree bounded by a constant. Specifically, in our case, $\mathcal{G}_D(B)$ has an independent set of vertices of size at least $n/20$ each of degree at most 9 (we are not concerned with the best possible constants). Such a set can be found by a greedy algorithm in $O(|\mathcal{G}_D(B)|) = O(|B|)$ time. Note that two independent vertices in $\mathcal{G}_D(B)$ correspond to two non-adjacent faces in $\mathcal{G}(B)$. Our algorithms use a hierarchical decomposition of $B^\cap$ dual to that of Dobkin and Kirkpatrick [6]; in particular, it is built in the exterior of $B^\cap$. Let $B_0 = B$ and let $I_i \subseteq B_i$ be the set of balls corresponding to an independent set of vertices in $\mathcal{G}_D(B_i)$. We have $|I_i| \geqslant |B_i|/20$. Let $B_{i+1} = B_i - I_i$. Then

$$B_0^\cap \subset B_1^\cap \subset B_2^\cap \subset \cdots \subset B_k^\cap,$$

where $|B_k| = O(1)$ and $k = O(\lg n)$ since $|B_{i+1}| \leqslant \frac{19}{20}|B_i|$. A ball $b \in B_i$ is *new* in B_i if $b \notin B_{i+1}$, otherwise it is *old*. A sphere or face is new (respectively old) in $\mathrm{bd}(B_i^\cap)$ if the corresponding ball is new (respectively old).

Multiple edges in the boundary chain of a face. The *boundary chain* $\mathrm{bd}(f)$ of a face f of $\mathrm{bd}(B^\cap)$ is the circular chain of edges (or corresponding neighbor faces) that bound f. A neighbor face can appear repeatedly in the chain since the intersection of two spheres can contribute more than one edge to $\mathrm{bd}(B^\cap)$; see Fig. 1. However, $\mathrm{bd}(f)$ has the DS_2 property: there is no subsequence of the form $\ldots a \ldots b \ldots a \ldots b \ldots$, where $a \neq b$ are faces, for such a subsequence would contradict the connectedness of the faces.

3. Sequential algorithm for ball intersection

We use a divide and conquer approach: divide B into two sets B_1 and B_2 of nearly equal size, compute $B_1^\cap$ and $B_2^\cap$ recursively, and merge them. The bottom of the recursion, when $|B|$ is bounded by a constant, can be solved by some brute force approach. Thus, we can concentrate on the problem of merging two spherical polytopes. For this, it would be sufficient to identify the *laces*, the connected components of $\mathrm{bd}(B_1^\cap) \cap \mathrm{bd}(B_2^\cap)$. The trouble here is that we do not know of a scheme to split B so that these laces are easily identified. The approach that works for halfspaces, to separate the dual points by a plane, produces multiple laces in the case of balls. Our approach is to split arbitrarily, and to design a general merge algorithm.

Let $|\mathcal{P}|$ denote the number of faces of the spherical polytope $\mathcal{P}$. We describe an algorithm that merges two spherical polytopes $\mathcal{P}$ and $\mathcal{Q}$ in time $O(m \lg m)$ where $m = |\mathcal{P}| + |\mathcal{Q}|$. Let

$$\mathcal{P} = \mathcal{P}_0 \subset \mathcal{P}_1 \subset \cdots \subset \mathcal{P}_k \quad \text{and} \quad \mathcal{Q} = \mathcal{Q}_0 \subset \mathcal{Q}_1 \subset \cdots \subset \mathcal{Q}_l$$

be hierarchical decompositions of $\mathcal{P}$ and $\mathcal{Q}$.

Dynamic programming. The approach is to compute all intersections $\mathcal{R}_{i,j} = \mathcal{P}_i \cap \mathcal{Q}_j$, $0 \leqslant i \leqslant k$, $0 \leqslant j \leqslant l$, using a dynamic programming method.

For each $i = 0, \ldots, k$, $\mathcal{R}_{i,l}$ can be computed using a brute force method in time $O(|\mathcal{P}_i| + |\mathcal{Q}_l|)$. Similarly, for each $j = 0, \ldots, l$, $\mathcal{R}_{k,j}$ can be computed in time $O(|\mathcal{P}_k| + |\mathcal{Q}_j|)$.

The main procedure is to compute $\mathcal{A} = \mathcal{R}_{i,j}$ from $\mathcal{B} = \mathcal{R}_{i,j+1}$ and $\mathcal{C} = \mathcal{R}_{i+1,j}$ in time linear in $|\mathcal{B}| + |\mathcal{C}|$.[2] Let the balls of $\mathcal{P}_i$ and $\mathcal{Q}_j$ be colored *blue* and *red*, respectively, and correspondingly for bounding spheres and faces. Recall the definition of new and old balls (spheres, faces) in a hierarchical decomposition. Thus, we have new blue and old blue balls (spheres, faces) in $\mathcal{P}_i$, and new red and old red balls (spheres, faces) in $\mathcal{Q}_j$. The balls (spheres, faces) in $\mathcal{A}$ are given the corresponding labels in $\mathcal{P}_i$ and $\mathcal{Q}_j$. Note that in $\mathcal{A}$ no two new faces of the same color may form an edge, by construction of the hierarchical decomposition (and therefore no three new faces of either color can form a vertex). To compute $\mathcal{A}$ it is sufficient to determine the boundary chain $\mathrm{bd}(f)$ of each face f. Each edge in $\mathrm{bd}(f)$ is old or new and blue or red depending on the corresponding neighbor face. First, we deal with old faces; only for them one needs to perform comparisons that depend on the radius of the balls. To determine the boundary of new faces, it is sufficient to follow pointers because all the vertices have already been computed (not all edges have been identified though).

Obtaining the boundary chain of an old face. Consider an old blue sphere s. We want to determine the boundary chain of f, the face of s in $\mathcal{A}$. Let f_1 and f_2 be the corresponding faces of s in $\mathcal{B}$ and $\mathcal{C}$, respectively, thus $f = f_1 \cap f_2$. Of course, if either f_1 or f_2 is empty, then f is empty. To compute the boundary chain $\mathrm{bd}(f)$ of f, it is sufficient to merge the boundary chains $\mathrm{bd}(f_1)$ in $\mathcal{B}$ and $\mathrm{bd}(f_2)$ in $\mathcal{C}$ in an appropriate manner. This problem is similar to that of intersecting convex polygons (in the plane) and, as for that problem, a sweep algorithm will do the job.[3] We only need to understand spherical polygons and their intersections to be ready to use a sweep algorithm. More specifically, we consider the restricted type of spherical polygons that appear in the algorithm.

For each sphere s, let the *north pole* of s be the point of largest third coordinate, denoted $N(s)$; its *south pole* $S(s)$ is the antipodal of $N(s)$. The *meridians* (geodesics between N and S) establish a circular order that can be used to perform a sweep. Given a spherical polygon g on s, we assume that $\mathrm{bd}(g)$ contains neither N nor S (a general position assumption that can be enforced with a symbolic perturbation). An edge e of $\mathrm{bd}(g)$ is a *northern* (respectively *southern*) edge if g is on its south (respectively north) side. A boundary chain is northern (respectively southern) if each of its edges is northern (respectively southern). For g, there are two cases: (i) either N or S is in g, and we say that g is a northern or southern face respectively; or (ii) neither N nor S is in g. In the first case, because of the "convexity" property of faces, $\mathrm{bd}(g)$ forms a *monotone* chain (every meridian intersects $\mathrm{bd}(g)$ in a unique point); the boundary is a southern or a northern chain respectively. In the second case, the chain $\mathrm{bd}(g)$ splits into two monotone chains, one northern and one southern (two edges may be split in the process). A spherical polytope has at most one northern face and at most one southern face. We assume that for each face of a spherical polytope one maintains the information of whether it is northern, southern or neither, and in this last case its northern and southern boundary chains. Note in

[2] Computing $\mathcal{R}_{i,j}$ from $\mathcal{R}_{i+1,j+1}$, $\mathcal{P}_i$ and $\mathcal{Q}_j$ in time linear in $|\mathcal{R}_{i+1,j+1}| + |\mathcal{P}_i| + |\mathcal{Q}_j|$ would result in a linear time merge, and an $O(n \lg n)$ time algorithm for the ball intersection problem.

[3] We have not been able to avoid the sweep and rather perform walks along the boundaries as in the intersection algorithm for convex polygons in [12] (see [13, p. 265]).

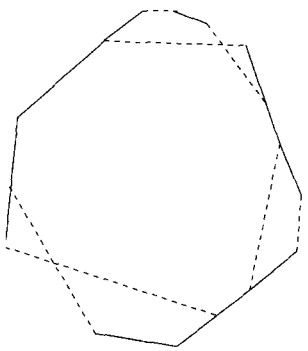

Fig. 2. Merging faces (new edges appear dashed).

particular that f is a northern (respectively southern) face iff both f_1 and f_2 are northern (respectively southern).

In each of $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$, new edges are independent, that is, they are not incident to common vertices. The chain $\mathrm{bd}(f)$ consists of pieces of boundary each of which can be either a piece common to both $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$ (old edges), a piece in $\mathrm{bd}(f_1)$ but not in $\mathrm{bd}(f_2)$ (new blue edges), or a piece in $\mathrm{bd}(f_2)$ but not in $\mathrm{bd}(f_1)$ (new red edges). For each of the last two types of pieces, there is a corresponding segment of boundary on the outside forming a region called a *sickle*. The interior boundary of a sickle, a single edge in $\mathrm{bd}(f)$, is a piece of a new edge, while the exterior boundary can consist of both old and new edges. See Fig. 2. Thus, $\mathrm{bd}(f)$ can be obtained from $\mathrm{bd}(f_1)$ by replacing pieces of $\mathrm{bd}(f_1)$ by pieces of new edges that appear in $\mathrm{bd}(f_2)$, which we call *bridges* (and similarly from $\mathrm{bd}(f_2)$). Given the circular chain for $\mathrm{bd}(f_1)$, it can be *augmented* to include the bridges from $\mathrm{bd}(f_2)$ (and similarly for $\mathrm{bd}(f_2)$).

With the information above in mind, a straightforward variation of the sweeping line algorithm to compute the intersection of two convex polygons (see, e.g., [13, pp. 263–265]) results in a sweeping meridian algorithm that determines all the intersections of $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$, and the chain $\mathrm{bd}(f)$. As a byproduct, the boundary chains $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$ are augmented to include the corresponding bridges. This takes time $O(|\mathrm{bd}(f_1)| + |\mathrm{bd}(f_2)|)$.

Obtaining the boundary chain of a new face. All vertices are already known either from $\mathcal{B}$ or $\mathcal{C}$ or from computing old faces in $\mathcal{A}$, since no vertex in $\mathcal{A}$ is incident to only new faces; thus, we only need to trace pointers. Let s be a new blue sphere, and let f and f' be the corresponding faces in $\mathcal{A}$ and $\mathcal{B}$. Here we do not have two boundaries to merge, but only $\mathrm{bd}(f')$ (because s is a new blue sphere, it does not appear in $\mathcal{C}$). $\mathrm{bd}(f)$ is obtained from $\mathrm{bd}(f')$ by determining the new red edges in $\mathcal{C}$ that create bridges. If no vertex incident to s has been found during the construction of old faces, then f is empty. Thus, let v be one such vertex; v is in $\mathrm{bd}(f')$ though it may not be a vertex of $\mathrm{bd}(f')$. Starting at v, construct $\mathrm{bd}(f)$ by tracing $\mathrm{bd}(f')$. At each step the tracing is on an old edge e, and at the same time we trace the boundary of the corresponding neighbor face g. A sickle starts when a vertex in $\mathrm{bd}(f')$ incident to a new red edge is reached (this information appears in the augmented chain of $\mathrm{bd}(g)$ where g is the neighbor face); the sickle ends when the next vertex in $\mathrm{bd}(f')$ incident to the same new red edge is reached (no vertices incident to other new red edges can be reached in between). In this procedure there is no need to compute new intersections (in particular, no comparisons involving

the radius), one just uses the circular chains of the old faces augmented with their bridges. Thus, it takes time $O(|\mathrm{bd}(f')|)$ to obtain the boundary chain of f.

Running time. The basic merge step, obtaining $\mathcal{R}_{i,j}$, takes time $O(|\mathcal{P}_i| + |\mathcal{Q}_j|)$ (since $\sum_{f\in\mathcal{P}} |\mathrm{bd}(f)| = O(|\mathcal{P}|)$, where the sum is over the faces f of $\mathcal{P}$). Thus, the time to obtain all $\mathcal{R}_{i,j}$, $0 \leqslant i \leqslant k$, $0 \leqslant j \leqslant l$ is

$$O\left(\sum_{i=0}^{k}\sum_{j=0}^{l}\left(|\mathcal{P}_i| + |\mathcal{Q}_j|\right)\right).$$

We have

$$\sum_{i=0}^{k}\sum_{j=0}^{l}|\mathcal{Q}_j| = O\left(\sum_{i=0}^{k}|\mathcal{Q}_0|\right) = O\big(|\mathcal{Q}_0|k\big) = O\big(|\mathcal{Q}|k\big),$$

and similarly the sum of the $|\mathcal{P}_i|$ is $O(|\mathcal{P}|l)$. The hierarchical decompositions of $\mathcal{P}$ and $\mathcal{Q}$ can be computed in time $O(|\mathcal{P}| + |\mathcal{Q}|)$. Since $k = O(\lg(|\mathcal{P}|))$ and $l = O(\lg(|\mathcal{Q}|))$, the time to merge $\mathcal{P}$ and $\mathcal{Q}$ is $O(m \lg m)$ where $m = |\mathcal{P}| + |\mathcal{Q}|$. Finally, the running time of the ball intersection algorithm satisfies the recurrence

$$T(n) = T\big(\lfloor n/2\rfloor\big) + T\big(\lceil n/2\rceil\big) + O(n \lg n).$$

Hence, $T(n) = O(n \lg^2 n)$.

4. Parallel algorithm for ball intersection

With the use of parametric search in view, we are interested in a parallel algorithm for a comparison model of computation (often called Valiant's model [14]). That is, we are interested in parallelizing those comparisons performed by the algorithm that involve a certain parameter; other computations do not need to be parallelized. Thus, when we speak of parallel running time in this model we actually mean the number of parallel batches of comparisons.

In our specific application, ball intersection, the parameter we are concerned with is the radius r of the balls. Each of these comparisons requires the determination of the sign of a polynomial of bounded degree in the variable r. For example, consider the comparison that determines whether a given ball contains a vertex which is one of the intersection points of three other balls; here the outcome depends on the sign of $r - r_0$, where r_0 is the radius of the sphere containing the four centers.

The divide and conquer approach parallelizes in a natural manner, introducing a factor $\lg n$ in the parallel running time. The construction of the hierarchical decomposition also parallelizes: no comparisons are involved in determining a large independent set of faces, and the reconstruction of the spherical polytope after the corresponding balls are removed is independent for each of the "cone" like portions of spherical polytope that appear. So there is a factor $\lg n$ in the parallel running time from the hierarchy construction. The dynamic programming construction can be parallelized by computing all $\mathcal{R}_{i,j}$ for fixed $i + j$ in parallel, thus introducing a $\lg n$ factor in the parallel running time. Below we show that $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$ can be merged in parallel to obtain $\mathrm{bd}(f)$ using $O(\log m)$ time and $O(m \log m)$ work where $m = |\mathrm{bd}(f_1)| + |\mathrm{bd}(f_2)|$. Thus, the basic merging step, $\mathcal{R}_{i,j}$ from $\mathcal{R}_{i,j+1}$ and

Fig. 3. A new blue edge intersected by several new red edges (dashed).

$\mathcal{R}_{i+1,j}$, also parallelizes, since the remaining computations do not require comparisons involving the radius.

Merging two boundaries in parallel. The vertices in $\mathrm{bd}(f)$ are old–old, old–new or new–new depending on the edges to which they are incident (disregarding orientation). Consider first the new–new vertices. A particular new blue edge e in $\mathrm{bd}(f_1)$ (which may be one of several contributed by the same neighbor face) may have multiple intersections with the red edges of $\mathrm{bd}(f_2)$; see Fig. 3. Note that all of the interior intersections of e are with new red edges, and except for the *extreme* ones (which can be the vertices of e) they are double intersections of e with new red edges. In this case, by the DS_2 property of boundaries, the two intersections of e with a new red edge e' are the only intersections of e' with $\mathrm{bd}(f_1)$. Thus, in order to find the new–new vertices, it is sufficient to find for each new edge, blue in $\mathrm{bd}(f_1)$ or red in $\mathrm{bd}(f_2)$, its extreme intersections with the other face (the intersections of e and e' are accounted for as the extreme intersections of e'). For this, for each new edge we perform a binary search in the boundary of the other face. This is explained below.

Now consider old–old and old–new vertices. Although they already appear in $\mathrm{bd}(f_1)$ or in $\mathrm{bd}(f_2)$, one needs information to decide their relative order along the boundary. Again the solution is to perform for each old edge e a binary search in the boundary of the other face. To be precise, taking e in $\mathrm{bd}(f_1)$, $\mathrm{bd}(f_2)$ does not cross e, but there may be vertices of $\mathrm{bd}(f_2)$ on e; we are interested in identifying the extreme ones, and hence all of them.

Binary search to determine the vertices. Let e be a new blue edge in $\mathrm{bd}(f_1)$. We use binary search to determine its extreme intersections with $\mathrm{bd}(f_2)$. If e is not monotone, then it can be split into two monotone pieces; let σ be on of the at most two monotone subchains of $\mathrm{bd}(f_2)$. Now that both e and σ are monotone, it is an easy task to perform the binary search required to determine the extreme intersections.

Let us assume that e is a southern edge. If σ is a southern chain, then e and σ can have multiple intersections (the case illustrated in Fig. 3). If σ is a northern chain, then there are at most two intersections between e and σ (by "convexity"). A first task of the binary search is to restrict σ to a subchain σ' within the interval of meridians determined by e. The second task is to determine the extreme intersections; it begins with a single initial interval that contains both potential intersections, and eventually splits into two binary searches, one for each extreme intersection. An essential observation for the use of binary search is that even though σ' may intersect e several times, the vertices of σ' behave well: in each of the intervals determined by the extreme intersections, all vertices are to north or to the south of e.

For the case of a southern old blue edge e, the binary search is as that for a southern new blue edge, if we interpret the vertices of σ on e as being to the south of e.

Tracing the boundaries. Once the vertices have been found using the binary search, the boundary of an old face $f = f_1 \cap f_2$ is determined by walking along $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$, determining the sickles and bridges (the positions of the new vertices along $\mathrm{bd}(f_1)$ and $\mathrm{bd}(f_2)$ are already known as a result of the

binary searches). Finally, the boundaries of new faces are determined as for the sequential algorithm (no comparisons are involved so it does not need to be parallelized).

Parallel running time and total work. There is a factor $\lg n$ due to the divide and conquer approach, a factor $\lg n$ due to dynamic programming method for merging, and a factor $\lg n$ due to the binary search for merging faces (the factor $\lg n$ from the hierarchy construction does not affect this because it is performed once in parallel for each of the levels of the divide and conquer tree). Thus, the parallel running time is $O(\lg^3 n)$. The total work performed by the parallel algorithm is $O(n \lg^3 n)$; the $\lg n$ factor increment over the sequential algorithm is due to the binary searches.

5. Algorithm for diameter

We follow Chazelle et al. [4] and Matoušek and Schwarzkopf [10] in using the parametric search technique of Meggido [11]. Our approach differs in that we have at hand sequential and parallel algorithms for ball intersection (which is also the case in [1]).

Let S be a set of n points, and let D be its diameter. As in [3], we need an oracle $\mathcal{O}$ that for any $r > 0$ decides whether $r < D$, $r = D$ or $r > D$. Their solution is as follows: if there are points of S outside $B(S,r)^{\cap}$ then $r < D$, if there are no points of S outside $B(S,r)^{\cap}$ but there are points in $\mathrm{bd}(B(S,r)^{\cap})$ then $r = D$, otherwise $r > D$. Thus, to implement $\mathcal{O}$, compute $B(S,r)^{\cap}$ and then for each point determine its position relative to $\mathrm{bd}(B(S,r)^{\cap})$ (this can be done in time $O(n \lg n)$ using standard planar point location techniques [7,13] after an appropriate projection).

We have at hand sequential $\mathcal{O}_s$ and parallel $\mathcal{O}_p$ versions of the oracle $\mathcal{O}$, which run in time $O(n \lg^2 n)$ and $O(\lg^3 n)$, respectively, the parallel one using $O(n \lg^3 n)$ work (the data structure for point location can be constructed in parallel within these time and work bounds; the n point locations are performed in parallel).

The algorithm for the diameter is as follows. We do not explain the general method of parametric search, but directly its application. Run the parallel oracle $\mathcal{O}_p$ with input S and $r = D$. Since the diameter D is not known, a parallel batch of comparisons that the algorithm presents is resolved as follows. Compute the roots of all the polynomials involved in the comparisons and sort them; then perform a binary search on the sorted list, using $\mathcal{O}_s$ at each step, to determine the position of D among the roots. Eventually, the value of D is determined as one of the roots.

This gives an overall running time in $O(n \lg^6 n)$: a factor $\lg^3 n$ due to $\mathcal{O}_p$, a factor $\lg n$ due to the number of iterations needed to resolve a batch of comparisons through binary search, and $O(n \lg^2 n)$ due to $\mathcal{O}_s$ (the $O(n \lg^3 n)$ work of $\mathcal{O}_p$ is an additive term, so it does not affect the asymptotic running time).

However, we can use Cole's trick [5] to reduce the running time to $O(n \lg^5 n)$. This is applied to the $O(n \lg n)$ simultaneous binary searches that determine vertices for merging faces. Each binary search is independent, so a search whose comparison has been resolved can proceed. Cole's trick, in our case, consists in assigning a weight 2^{-d} to the comparison of depth d in each binary search. Using a weighted median on the ordered list of active comparisons (those waiting to be resolved), each iteration resolves comparisons with at least half of the active weight. The comparisons resolved introduce comparisons with half the weight, so the total weight of active comparisons decreases by at least $1/4$. Thus, the total active weight decreases exponentially as $(3/4)^i$; since the initial weight

is $O(n \lg n)$, and the smallest weight of a comparison is $2^{-O(\lg n)}$, then all comparisons are resolved in $O(\lg n)$ iterations. This collects the factor $O(\lg n)$ of the length of a binary search and the factor $O(\lg n)$ of the number of iterations to resolve a parallel batch of comparisons into a single one.

Acknowledgements

I am indebted to Herbert Edelsbrunner for observing that the dynamic programming method can be parallelized with a number of parallel stages $O(\lg n)$ rather than $O(\lg^2 n)$, thus cutting a factor $\lg n$ in the running time of the diameter algorithm.

References

[1] N. Amato, M. Goodrich and E. Ramos, Parallel algorithms for higher dimensional convex hulls, in: Proc. 35th IEEE Sympos. Found. Comput. Sci. (1994) 683–694.
[2] H. Brönnimann, B. Chazelle and J. Matoušek, Product range spaces, sensitive sampling, and derandomization, in: Proc. 34th IEEE Sympos. Found. Comput. Sci. (1993) 400–409.
[3] K.L. Clarkson and P.W. Shor, Applications of random sampling in computational geometry II, Discrete Comput. Geom. 4 (1989) 387–421.
[4] B. Chazelle, H. Edelsbrunner, L. Guibas and M. Sharir, Diameter, width, closest line pair, and parametric searching, Discrete Comput. Geom. 10 (1993) 183–196.
[5] R. Cole, Slowing down sorting networks to obtain faster sorting algorithms, J. Assoc. Comput. Mach. 34 (1987) 200–208.
[6] D.P. Dobkin and D.G. Kirkpatrick, Fast detection of polyhedral intersection, Theoret. Comput. Sci. 27 (1983) 241–253.
[7] H. Edelsbrunner, Algorithms in Combinatorial Geometry (Springer, Heidelberg, 1987).
[8] H. Edelsbrunner and E.P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, ACM Trans. Graph. 9 (1990) 66–104.
[9] A. Heppes, Beweis einer Vermutung von A. Vázsonyi, Acta Math. Acad. Sci. Hungar. 7 (1956) 463–466.
[10] J. Matoušek and O. Schwarzkopf, A deterministic algorithm for the three-dimensional diameter problem, in: Proc. 25th ACM Sympos. Theory Comput. (1993) 478–484.
[11] N. Meggido, Applying parallel computation algorithms in the design of serial algorithms, J. Assoc. Comput. Mach. 30 (1983) 852–865.
[12] J. O'Rourke, C.-B. Chien, T. Olson and D. Naddor, A new linear algorithm for intersecting convex polygons, Computer Graphics Image Processing 19 (1982) 384–391.
[13] F. Preparata and M.I. Shamos, Computational Geometry – An Introduction (Springer, New York, 1985).
[14] L. Valiant, Parallelism in comparison problems, SIAM J. Comput. 4 (1975) 348–355.