Computing a Point in the Center of a Point Set in Three Dimensions
(Extended Abstract)

Nir Naor\textsuperscript{(1)} and Micha Sharir\textsuperscript{(1,2)}
(1) School of Mathematical Sciences, Tel-Aviv University
(2) Courant Institute of Mathematical Sciences, New-York University
June 28, 1990

1 Introduction

Let \( A \) be a set of \( n \) points in three dimensional space. The center of \( A \) is a set of points \( q \), such that any plane \( m \) containing \( q \) divides the points in \( A \) in a fairly balanced way, namely each of the two closed half spaces bounded by \( m \) contains at least \( \frac{n}{4} \) points of \( A \).

This paper describes two versions of an algorithm that finds a point in the center of \( A \). The first version solves the problem in \( O(n^2 \log^9 n) \) time. The second version solves the problem in time \( O(n^2 \log^6 n) \) but with a larger constant factor in the bound. Both algorithms improve considerably naive solutions which require \( O(n^9) \) time.

Our algorithm is a generalization of the one given in [CSY] for the 2-D problem. It makes extensive use of the “parametric searching” technique of Megiddo [M] which uses parallel algorithms to generate efficient sequential algorithms. We also apply the technique of Cole [C87] in the second version, the parallel sorting algorithms [C86],[AKS] and a generalization of the [EOS] algorithm for constructing a planar arrangement of \( n \) lines.

2 The Center - Its Existence and Properties

Let us denote the center of \( A \) by \( Q(A) \) or \( Q \) for short. Using Helly’s Theorem on convex sets we can show that for any set \( A \), \( Q(A) \) always exists and it is a closed convex domain.

Given a point \( q \), any point \( a_0 \in A \) divides the unit sphere \( S^2 \) by a great circle \( c_0 \) into two halves, where \( c_0 \) is the collection of all the normals to planes which contain both \( q \) and \( a_0 \). Given a plane \( m \) containing \( q \), to determine whether \( a_0 \) lies above \( m \) one needs to determine the hemisphere of \( S^2 \) bounded by \( c_0 \) which contains the normal of \( m \). The set \( A \) thus induces an arrangement \( R(q) \) of \( n \) great circles on \( S^2 \). When the normal of any plane \( m \) containing \( q \) remains within one of the faces of \( R(q) \), the number of points of \( A \) above \( m \) is unchanged. The arrangement \( R(q) \) can be computed in time \( O(n^2) \) by a generalization of the [EOS] algorithm for computing the arrangement of \( n \) lines in the plane.

The number of points of \( A \) above any plane \( m \) containing \( q \) changes only by \( \pm 1 \) as the normal of \( m \) moves from one face of \( R(q) \) to a neighboring face. Thus in \( O(n^2) \) time we
can compute for each region $r \in R(q)$ the number of points of $A$ above $m$ when the normal of $m$ is in $r$. (For the first region we can simply count it.) Note that $q \in Q(A)$ iff for each $r \in R(q)$ the number of points above any plane $m$ whose normal is in $r$ is between $\frac{n}{4}$ and $\frac{3}{4}n$. Hence we conclude:

**Lemma 1** For each point $q$ we can decide in $O(n^2)$ time if $q \in Q(A)$.

### 3 Spherical Sort - SS

In order to facilitate the search for a point in the center of $A$, we slightly modify the above construction of $R(q)$ to fit it better into Megiddo's technique. We call the new algorithm Spherical Sort (SS for short) around $q$. Let $C(q)$ denote the collection of $n$ great circles on $S^2$ induced by $q$ and $A$. The algorithm sorts separately for each circle in $C(q)$, its intersection points with the other circles in $C(q)$. There is a 1-1 correspondence between the output of the SS algorithm and $R(q)$. Hence we could compute SS more efficiently as a by-product of the $R(q)$ computation. However, it is much easier to parallelize the SS algorithm, and this is an essential precondition for Megiddo's technique. It also implies that the center membership can be determined by the SS algorithm. Note that the SS algorithm requires $O(n^2 \log n)$ time.

Let us consider a single comparison in SS, e.g., the comparison between $c_j$ and $c_k$ along $c_i$. We can show that the result of this comparison depends upon the location of $q$ relative to three critical planes.

- The plane containing $a_i, a_j, a_k$.
- The plane containing $a_i, a_j$ and perpendicular to the plane $z=0$.
- The plane containing $a_i, a_k$ and perpendicular to the plane $z=0$.

Hence, to find a point $q$ in the center we can apply Megiddo's technique [M] as follows. We run the SS algorithm "generically" without knowing the value of $q$. Each comparison that the algorithm executes is resolved by testing the position of $q$ with respect to three corresponding critical planes. We assume the existence of an algorithm called IPSS (Implicit Planar Spherical Sort), that for any given plane $m$ determines on which side of $m$ the center $Q$ lies, or in case $m$ intersects $Q$, reports a point in the intersection. Thus each comparison in the generic SS algorithm can be resolved by applying the IPSS algorithm to the relevant critical planes. The SS algorithm always terminates when one of the critical planes is found by the IPSS procedure to contain a point of $Q$. Indeed, if the generic execution runs to completion, its output contains as a by-product a convex polyhedron $K$, which is the intersection of all half-spaces returned by the calls to IPSS, so that $Q$ is contained in $K$, and so that the output of SS is constant over $K$, necessarily implying $K = Q$. Hence one of the critical planes bounding $K$ will be found to contain a point in the center. The complexity of this solution is $O(n^2 \log n \cdot T(IPSS))$ where $T(IPSS)$ is the time required by the IPSS algorithm. The complexity of the IPSS that we develop is more than quadratic, so we seek a better solution.
4 The Megiddo Technique - MT

The second idea in Megiddo's Technique (MT for short) is to run a parallel version of the generic algorithm, so that the comparisons are executed in a small number of "batches" each consisting of independent comparisons. Let \( E = \{e_1, \ldots, e_n\} \) be such a batch of independent comparisons. Suppose also that \( \{e_1, \ldots, e_n\} \) can be ordered in such a way that given the result of one comparison \( e_i \), we can either determine the results of all the comparisons preceding \( e_i \) or determine the results of all the comparisons succeeding \( e_i \) in time \( O(1) \) per comparison. Then we can perform binary search over this sequence and by resolving \( O(\log n) \) comparisons we can find two adjacent comparisons \( e_l, e_r \) such that given the result of \( e_l \) we can find the results of \( e_1, \ldots, e_{l-1} \), and given the result of \( e_r \) we can find the results of \( e_{r+1}, \ldots, e_n \) in time \( O(1) \) for each. If the time required to resolve one comparison is \( T(e) \) then we can resolve all the comparisons in \( E \) in time \( O(\log n \cdot T(e) + n) \) instead of \( O(nT(e)) \). It means that the essence of MT is reducing the number of expensive comparisons and replacing most of them by cheaper comparisons.

5 The Algorithm

Each individual sort in SS can be easily parallelized into an \( O(\log n) \) parallel time algorithm such that each parallel step requires the computation of \( O(n) \) independent comparisons [C86], [AKS]. The individual sorts in SS are independent, hence we can combine them into an overall parallel SS algorithm of \( O(\log n) \) steps, each step requires the solution of \( O(n^2) \) independent comparisons. If we could impose some order on the comparisons according to the setup of MT, we would be able to apply MT and perform each parallel step in:

\[
O(\log n \cdot T(\text{comparison}) + T(\text{sorting the critical planes}) + n^2)
\]

instead of \( O(n^2 \cdot T(\text{comparison})) \) without MT.

We can resolve the \( O(n^2) \) comparisons of a single parallel step by determining the location of \( Q \) versus the \( O(n^2) \) corresponding critical planes. Let IPSS be the algorithm that we referred to earlier. Let CPS (Critical Plane Sort) be an algorithm which sorts the \( O(n^2) \) critical planes of a single parallel step along the vertical unknown line \( l_q \) containing \( q \) and parallel to the \( z \)-axis. Note that if \( l_q \) were known, the CPS algorithm would simply have to sort the planes in increasing order of their intersections with \( l_q \) and then run a binary search through these intersections, using, say IPSS to guide the search. However, since \( l_q \) is not known, we have to run CPS generically as well. Every time CPS compares the intersections with \( l_q \) of two critical planes, we look at the line \( \lambda \) of their intersection, project it on the \( xy \) plane, and determine (using IPSS) on which side of this projection of \( \lambda \) lies \( l_q \). Note that this approach reduces the degree of freedom — instead of testing the position of \( Q \) with respect to \( O(n^2) \) arbitrary planes, we have to test it only with respect to \( O(n^2) \) vertical planes. Unfortunately, this is still not sufficient to sort the comparisons of CPS, so we add one more intermediate algorithm VPS (Vertical Planes Sort) in which we assume that we know the plane \( \pi_q \) parallel to the \( yz \) plane that contains \( l_q \) (and \( q \)), and try to locate \( l_q \) within \( \pi_q \). If we really knew \( \pi_q \), the process just explained would have found \( l_q \). But since \( \pi_q \) is unknown, we need to run the VPS procedure generically too, and resolve comparisons between pairs of vertical planes by finding their intersection line and testing
its position with respect to $\pi_q$. Since we assume $\pi_q$ intersects $Q$, we can resolve this test by applying $IPSS$ to the plane through their intersection which parallel to the $yz$ plane.

The resulting battery of algorithms and sub-algorithms is fairly complex, and the above description only hints of the level of sophistication that is needed. A complete description of the algorithm is given in [NS]. Substituting $IPSS$ and $CPS$ in the complexity of a single step and multiplying by the number of parallel steps — $\log n$, implies that the entire algorithm can be executed in time:

$$O(\log^2 n \cdot T(IPSS) + \log n \cdot T(CPS) + n^2 \log n)$$

Careful analysis of the complexity of these sub-procedures yields:

**Theorem 2** The above algorithm finds a point in the center of a set of $n$ given points in 3-D in time $O(n^2 \log^9 n)$.

The second version of our solution is achieved by modifying the previous solution according to Cole [C87]. These modifications allow us to solve only a single expensive comparison at each parallel step instead of $O(\log n)$ comparisons. Careful analysis of the complexity of our modified system of algorithms yields:

**Theorem 3** The modified algorithm finds a point in the center of a set of $n$ given points in 3-D in time $O(n^2 \log^6 n)$.

**References**


[NS] N. Naor, M. Sharir, On Finding a Point in the Center of n Given Points in Three Dimensional Space, *In preparation*