NATO INTERNATIONAL ADVANCED STUDY INSTITUTE
"Theoretical Foundations of Computer Graphics and CAD"
Il Ciocco, Castelvecchio Pascoli, Lucca, Near Pisa, Tuscany, Italy
Saturday, 4th July - Friday 17th July, 1987

COMPUTATIONAL GEOMETRY -- THEN AND NOW

David P. Dobkin

CS-TR-084-87

March 1987

COMPUTATIONAL GEOMETRY -- THEN AND NOW

David P. Dobkin+

Department of Computer Science
Princeton University
Princeton, NJ 08544

ABSTRACT

In this paper we explore the development of solutions to five key problems in the computational geometry of the plane. The problems we consider were chosen on the basis of their longevity, their significance in computational geometry and the existence of reasonable solutions to each. In each case, the problem has been actively considered for a decade by both practitioners and theoreticians. The set of problems gives an accurate overview of the problems and methods of computational geometry (in the plane). In most cases, the solutions are practical and we will describe implementation issues. Where appropriate, we discuss extensions to higher dimensions.

1. INTRODUCTION

In 1975, Mike Shamos coined the term computational geometry to describe work that he was doing on the analysis of algorithms and data structures for geometric algorithms. He didn't realize that Robin Forrest had used the same word to describe work that he had done in his PhD thesis 7 years earlier [Fo68]. In time, both computational geometries have grown [PS85,FP81]. To alleviate the concern about two areas of intellectual endeavor bearing the same name, Pierre Rosenstiehl [Ro85] suggested distinguishing the two computational geometries by using ''discrete computational geometry'' to refer to the field named by Shamos and ''continuous computational geometry'' to refer to the field named by Forrest. When necessary we will use this distinction to differentiate the two.

This paper concerns the development of key ideas within discrete computational geometry. We will consider five problems that have been considered for the past 10 years and study their development during that time period. The problems we consider are the computation of convex hulls, the searching of planar subdivisions, the finding of nearest neighbors, range searching and intersection detection and computation. We choose these problems for three reasons. First, they give an accurate representation of the state-of-the-art in computational geometry circa 1977 and thus are of historical interest. Second, the development of solutions to each over the past 10 years provides evidence of the success of computational geometry and also provides a good survey of the techniques used throughout the field. Finally, the problems are sufficiently simple to be easily studied.

As we describe in greater detail below, much of what is done in computational geometry arises from the study of methods of sorting and searching in dimensions higher than one. Sorting and searching are well understood for totally ordered sets in 1-dimension [Kn73]. However, beyond this simple case it is often difficult to define the exact sorting/searching problems under consideration. When it is useful in what follows, we will characterize problems as sorting and searching problems and identify the ordering information that is available to us. This is useful in establishing connections among geometric problems that superficially appear dissimilar. These connections are described as a means to provide the reader with techniques that might be useful in solving his/her next geometric problem.

+This research supported in part by the National Science Foundation under Grant CCR85-05517.

The planar convex hull problem described below is a good example of a higher dimensional sorting problem. This problem requires finding that subset of a set of points which define the limit of the set. This is done by ordering the points to determine a single convex polygon. Next, the problem of planar subdivision searching is considered. Appropriate planar subdivisions can be considered as convex hulls of 3-dimensional polyhedra. Here, we are organizing the points on the convex hull so that the faces of the hull, corresponding to regions in the planar subdivision, can be efficiently searched. We will see this representation arise again when we discuss hierarchical representations of convex polygons and polyhedra that can be used for efficient intersection calculation and searching.

The problems of finding nearest neighbors and planar subdivision searching are considered. Nothing in the problem statements suggests that the problems are related. However, the algorithms proposed to solve the problems are quite similar. This is a situation where the algorithms proposed unite the problems rather than vice versa as is more common. Ultimately, finding nearest neighbors is best done by building a Voronoi diagram. This Voronoi diagram is a planar subdivision. Processing of this planar subdivision is done by the general methods introduced for processing general planar subdivisions. This involves the same techniques used for processing convex polyhedra in 3 dimensions. Some of these connections are recent and should lead to further organization of the field of computational geometry.

The connections mentioned above are examples of the ideas we intend to develop below. The organization of the paper is as follows: in the next section, we describe the problems we intend to pursue in the text. Next, we survey the state of the art for each problem as it stood in 1977. In the statement and description of early work, we describe where the problem arose and what initial solutions were found (i.e. state of the art circa 1977). Finally, we describe developments leading to the current best-known solutions. Where appropriate, implementation details are discussed. Connections between the problems are described to document how computational geometry is becoming unified.

2. Five easy pieces of computational geometry

We now describe the five problems that form the basis of this paper. Each has been considered by computational geometers for over 10 years. And, each demonstrates central ideas of the field in both its statement and solution. The uninformed reader is encouraged to pause after reading each problem to ponder reasonable methods that might lead to a solution.

Convex hull of a point set

The input is a set of points $P_1, P_2, \cdots, P_n$ in d-dimensional Euclidean space. The desired output is the smallest convex set containing all of the points. In general this set will be a polyhedron in the space.

Specializing to the plane, in the non-degenerate case the solution will be a polygon P with vertices from the original set. Each vertex of P will be extremal in some direction. That is, if $P_i = (x_i , y_i)$ is a vertex of the convex hull, then there exist real numbers a and b such that for all $j \neq i$,

$$ax_i + by_i > ax_j + by_j.$$

This suggests a relationship with linear programming problems where the feasible region is specified as the intersection of a set of halfplanes (in the plane or halfspaces in higher dimension). Indeed, as we will see below, this relationship has been exploited in establishing relationships between the convex hull problem and the halfspace intersection problem.

Another characterization of the convex hull problem is given for an analog computing model. We might consider a board corresponding to the plane with nails hammered in at points of the point set. If a large rubber band has a circumference that includes of all the points, the convex hull is the set of points (or the polygon) that impede the motion of the rubber band when it is released [Sh74].

Finding nearest neighbors

The input is a set of points $P_1, P_2, \cdots, P_n$ in d-dimensional Euclidean space. The desired output is the nearest neighbor of each point or possibly the pair of points that are closest. The answer will consist of a directed (possibly disconnected) graph with an edge from i to j iff P_i is the nearest neighbor of P_j. Note that the graph is directed since nearest neighbor is not a symmetric relation. Or, the answer will consist of

the pair of points that realize

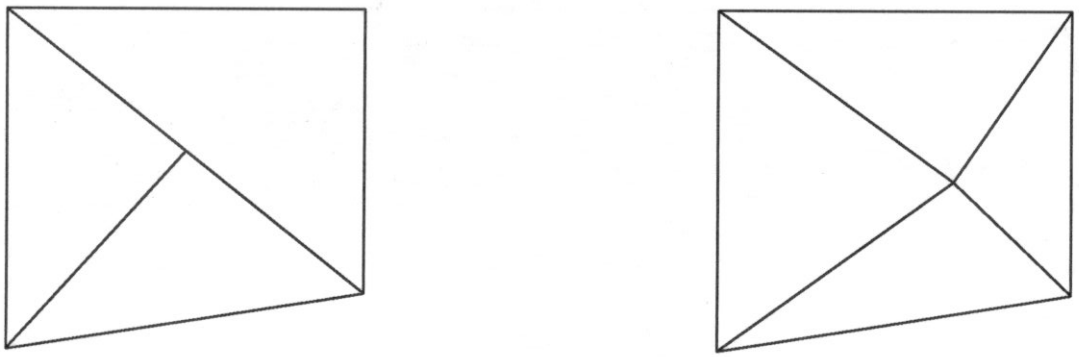
$$\min_{1 \le i,j \le n} d(P_i, P_j).$$

A closely related problem asks for furthest neighbors. Here again the output is a graph. The problem where we request the pair of points that realize the maximum interpoint distance in the set is known as the *diameter* problem since this distance is called the diameter of the set. We show below that the diameter of a set is determined by two points that lie on its convex hull. This establishes a potential relationship between this problem and the convex hull problem.

In our study of the nearest and furthest neighbors, we study a structure commonly known as the closest point Voronoi diagram. This is a subdivision of space assigning to each point P_i a region R_i such that points in R_i are closer to P_i than to any of the other points. Each of the regions R_i is a convex polytope. Furthermore, the only unbounded regions correspond to points on the convex hull of the original set. We will also consider a furthest point Voronoi diagram. We've mentioned these structures here because they provide interesting examples of planar subdivisions for consideration in our next problem.

Planar subdivision searching

The input is a subdivision of the plane into regions that are convex polygons that only overlap in edges and vertices. For example, these polygons may have arisen from a closest point Voronoi diagram as described above. For convenience, let us assume that the polygons have all been triangulated so that the input is a set of triangles $T_1, T_2, \cdots, T_n$. We will assume further that two triangles overlap in an edge iff the complete edge belongs to both of them. This disallows configurations such as shown in Figure 1.



not a planar subdivison

a planar subdivison

Figure 1 -- planar subdivisions -- valid and invalid

The problem here is to efficiently search the planar subdivision. Searching a planar subdivision assumes that the subdivision has been sorted. This sorting is done during a preprocessing phase. The result of the sorting is to create a data structure that can be searched. It is assumed that the sorting is done once and supports numerous searches. Hence, the cost of the sorting can be amortized over the searches. Algorithms for planar subdivision searching have their complexities specified by a triple of numbers. This triple involves the preprocessing time for the sort, the space needed to store the searching data structure and the time for each search.

Planar subdivision searching can be viewed as a multidimensional extension of binary search. In the latter problem, we are given a set of numbers that can be viewed as a set of points on a line (say the x-axis with the numbers being x coordinates). Then, the problem becomes one of organizing the points, this we do by sorting x coordinates, and searching them, this we do by standard binary search (see e.g. [Kn73]). This operation requires n log n time for the sort, linear space to store the search data structure and log n time for each search.

Just as planar subdivision searching is an extension of binary searching from 1 dimension into 2, there are extensions into higher dimensions. We will mention below extensions of the problem into d dimensions and consider the general problem of spatial subdivision searching. Our next problem is a different version of the multidimensional searching problem.

Range searching

Next we consider a problem that is similar in spirit to the planar/spatial subdivision search. The input is a set of points $P_1, P_2, \cdots, P_n$ in d-dimensional Euclidean space. We again want to preprocess the points to allow for efficient searches. Here the searches involve ranges in the d-dimensional Euclidean space and ask which of the points lie in the range. Here a range can be viewed as a box, as a simplex or as a halfspace. For example, a range might be described as a set of ranges $m_1, m_2, \cdots, m_d$, $M_1, M_2, \cdots, M_d$ where all points $P_i = (x_{i1}, x_{i2}, \cdots, x_{id})$ with $m_k \leq x_{ik} \leq M_k$ for all k are desired. A point is said to satisfy a hyperplane if it lies in the positive halfspace determined by the hyperplane. A hyperplane search asks for the subset of the points that satisfy the hyperplane. A simplex search asks for the subset of the points that simultaneously satisfy all of a set of hyperplanes.

Two versions of this problem must be distinguished. These are the **counting** problem and the **reporting** problem. In the former, the output is a count of all points that satisfy the conditions (i.e. lie in the range). The latter asks not only for the number of points but also for the names of the points in the range. An algorithm for the reporting problem will obviously solve the counting problem. However, there may be a more efficient solution if only a count is desired. This distinction is important in what follows.

In considering this problem in conjunction with the previous problem, it appears as though a dual transformation should be possible (see e.g. [Br79b]). That is, it should be possible to transform this problem into one of point location in a spatial subdivision. In practice, this appears to not be the case as the algorithms developed for the problems have significantly different forms.

Intersection problems

Our final problem is the problem of intersection computation. We consider here a version of the problem for which the input is a set of line segments $s_1, s_2, \cdots, s_n$. There are two versions of the problem. In one, the output is a listing of all pairs of segments that intersect. For the second, a YES/NO answer is desired telling whether any pair of segments intersect. The analog to the reporting/counting problems above is obvious Clearly, determining whether any pair of segments intersect is no harder than enumerating all pairs that do.

Intersection is a basic problem of geometry. This problem forms a basis for many others where the line segments are replaced by axis-oriented rectangles or convex polygons. The distinction between detection and computation is significant since for many applications, a fast detector is sufficient.

Initially, intersection problems seem different from the other problems we consider here. In particular, there is no sorting or searching obvious in the problem. But, we shall see otherwise below. Convex hull computations are the analogs of sorting. In the same way, intersection of convex bodies is the analog of merging two sorted lists. This will become clear in the next sections.

Lower bounds

In our discussion below, we will have occasion to ask whether we've achieved the best possible algorithm. Establishing a lower bound is a difficult process. The lower bound will only hold for a specific model of computation. This model must be very clearly specified. And, it must be shown that no possibilities for a faster algorithm exists within the model.

Our lower bound results will arise from counting considerations and from reductions to problems of known lower bound. The counting considerations arise from sorting problems. It is known (see e.g. [Kn73]) that a binary tree that decides among k possible outcomes that are pairwise disjoint must have at least k leaves and therefore must have depth at least $\log_2 k$. A binary algorithm (i.e. comparison based) for sorting must decide among n! possible permutations to establish the total order among n numbers. Thus, an $n\log_2 n$ lower bound on sorting is established. This will be the principal lower bounds used in this paper.

The other lower bound we consider will be on the *element uniqueness problem.* In this problem the input is the set $x_1, x_2, \cdots, x_n$. of real numbers. The output is a YES/NO answer corresponding to whether there exist $1 \le i,j \le n$ such that $x_i = x_j$. We assume a model of computation where queries are linear forms in the inputs (i.e. tests comparing $\sum_{i=1}^{n} a_i x_i$ to c for real numbers a_i, i=1,...,n and c). In this case, it is known [DL79,Be83] that O(n log n) queries are necessary and sufficient.

3. THEN

This section is devoted to surveying the state-of-the-art in computational geometry as it stood in 1977, ten years ago. The problems of the previous section were chosen as pioneer problems because each had been the subject of some consideration before 1977 and their union provides a reasonable overview of the breadth of the field.

Curiously, for each problem, the initial statement of the problem was accompanied by significant progress towards a solution. This progress was sufficient in most cases to convince researchers of that time that the problem was solved. As we will see in the next section, in the past decade further progress has been made on each of the problems so that the old solutions have been improved and expanded upon. From the vantage point of 1987, we are willing to suggest that progress has been made and many issues have been resolved. But, we have learned the lesson from history that a problem is never completely solved.

Convex hull of a point set

Computing convex hulls is the oldest and most popular problem of computational geometry. Many results on the problem date back to the late 1960's and seem to have their roots in statistical applications. Connections between convex hulls and mathematical programming seem to have led to the strategies behind the initial algorithms. The two earliest published algorithms are due to Graham [Gr71] in dimension 2 and to Chand and Kapur [CK70] in higher dimensions.

At the time, Chand and Kapur were working at Lockheed and the convex hull problem arose in their work. They describe an algorithm based upon the principle of ''gift-wrapping'' for computing convex hulls of spatial point sets. The key idea of their algorithm is the observation that in any dimension, an edge (of dimension 1) of the convex hull belongs to exactly two facets of the convex hull. The dimension of the facets will be one less than the dimension of the resulting hull. Thus, their algorithm works by finding an initial facet of the hull and then joining facets to the growing hull along edges that belong to only a single facet. They describe in their initial paper a FORTRAN implementation and report having successfully computed the convex hull of a set of 1000 points in 6 dimensions.

A 2 dimensional rendering of their algorithm might go as follows:

Find a point, p, of the convex hull and make it the current point.
while you haven't returned to p, **do**
    find the point that is *most counterclockwise* from p
    add that point to the convex hull and make it the current point.

The initial point could be the point of minimal y coordinate. The *most counterclockwise* point is found by scanning the points and doing tests to see if triples are counterclockwise. To see whether B is more counterclockwise than C with respect to A, we ask whether the triangle ABC has a counterclockwise orientation. The progress of the algorithm is illustrated in Figure 2, where we see the addition of the first 5 points followed by the final hull.

The extensions of this algorithm to dimension d is reasonably straight-forward. The initial facet can be found as a extremal flat of dimension d-1. This can be found by computing a convex hull in R^{d-1}. Thus, we can recurse on dimension to find the first facet. The counterclockwise test is merely a test to see if a matroid is oriented. In implementation, the hull is built by adding one point at a time. Again, this is done by folding up along the crease that is an edge in a gift-wrapping fashion.

The Graham algorithm was discovered independently and arose in response to a request from colleagues at Bell Labs who were doing the computation for statistical applications. His algorithm consists of two steps. First, the points are sorted with respect to an interior point. Next, a ''Graham scan'' is

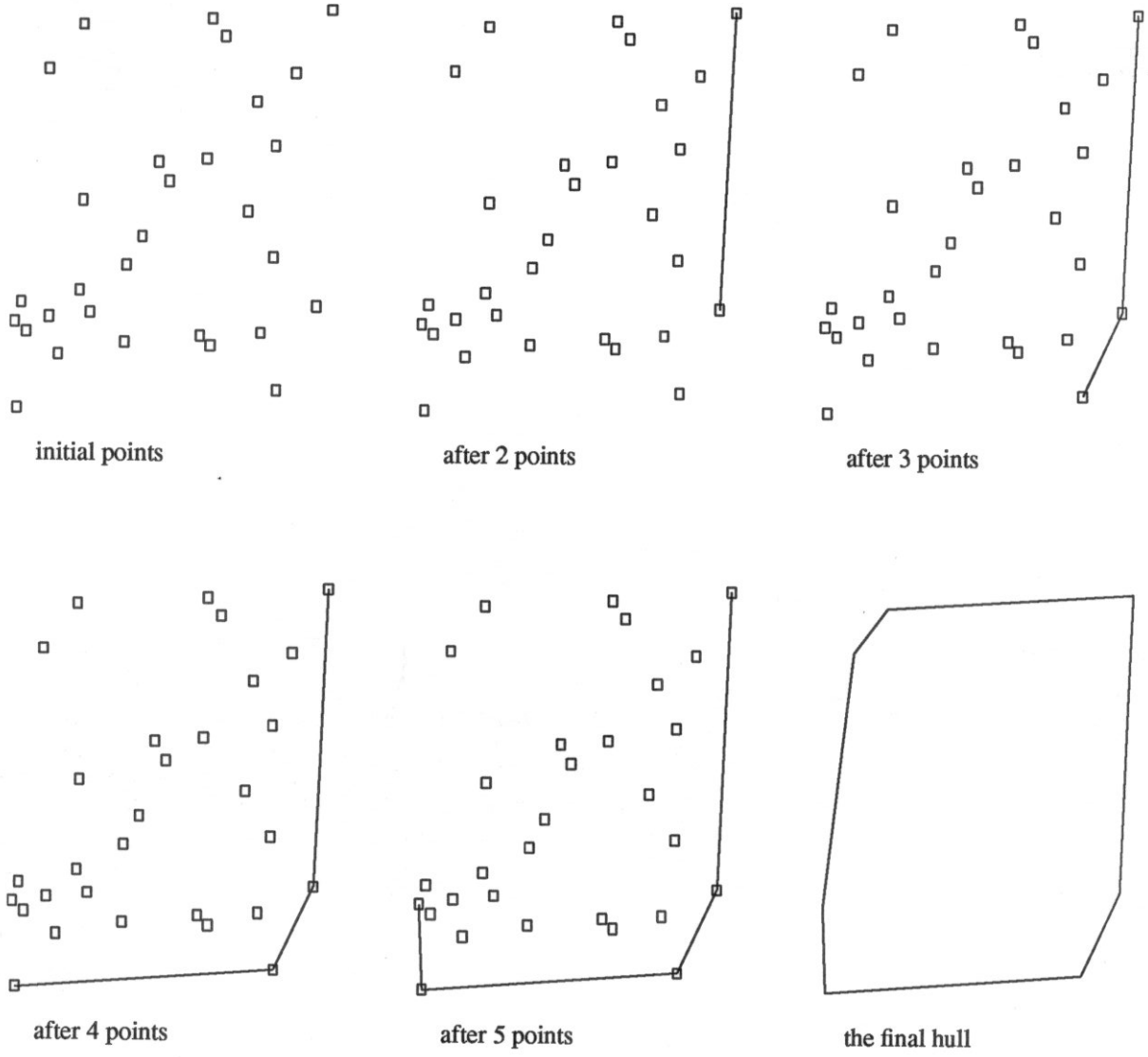
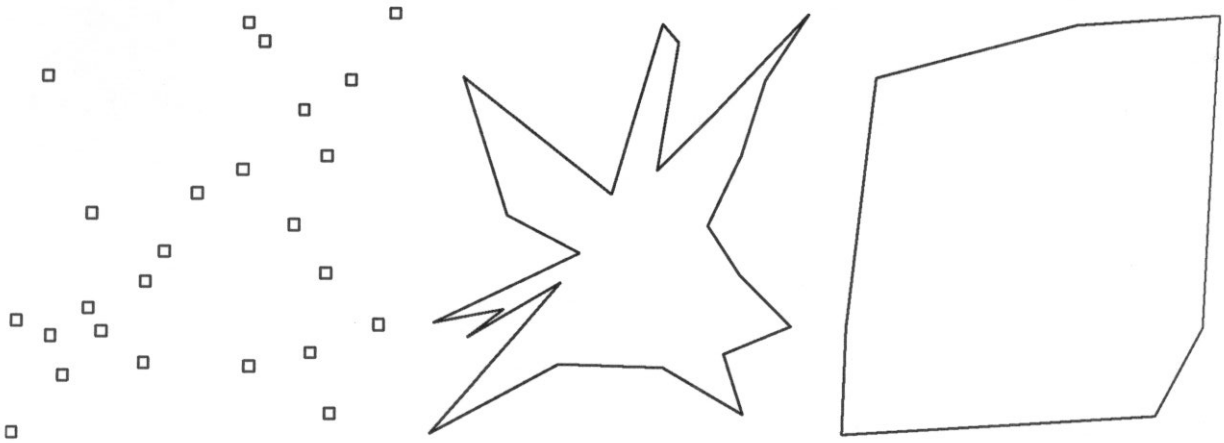


Figure 2 -- Convex hull via gift wrapping

performed. During this step, points are traced about the perimeter of the polygon determining whether a *left turn* or *right turn* was made at each vertex. These correspond to tests to see whether we are moving clockwise or counterclockwise. If the motion is counterclockwise, the point is tentatively accepted and the scan moves forward. At a clockwise point, the point is rejected and the scan moves back. Since the scan can have no more than n-3 rejections, the process halts in a linear number of steps.

The behavior of this algorithm on a set of points is shown in Figure 3. As a 2 dimensional algorithm, it possesses a simplicity that is not present in the gift wrapping algorithm, However, it does not naturally extend to higher dimensions.

In the years following these algorithms, numerous improvements were suggested often proposing a special model where one's algorithm was the best. Of the probabilistic algorithms and analyses, those of Eddy and Floyd are probably the most significant ([Ed77,Fl76]). In most cases, linear behavior will result and performance will be superior to that of a sorting based method. Preparata and Hong proposed an algorithm for 3 dimension convex hull computation in 1977 [PH77]. This algorithm is based upon divide-and-conquer. They compute left and right hulls and then determine common tangents (in 2 dimensions) and a wrapping (in 3 dimensions) that can be used to merge the hulls.



input points the star the convex hull

Figure 3 -- Convex hull via a Graham scan

Already by 1977, it had been determined that sorting must play a central role in the derivation of a convex hull algorithm. It was seen that orientation would be the significant test in the inner loop. And, optimal algorithms were known in 2 and 3 dimensions. Open questions remained however. Of these, central questions asked for extensions to higher dimensions and for methods that took advantage of the information they had on input and/or had running times proportional to their input size. For example, suppose we were given a star-shaped polygon, could we compute its convex hull in linear time? If we determined in our computation that few points were on the convex hull, could we compute the hull in faster asymptotic time? How hard would it be to just identify points of the convex hull (in higher dimensions) without building all of its structure?

Finding nearest neighbors

The next problems to be considered on point sets were computations involving distance properties. Convex hull algorithms found points that were extrema of a set. It was natural to next focus attention on finding pairs of points that realized extremal distances. The first problems that arose were to find the closest and furthest pair of a set of n points.

First, the 1 dimensional versions of the problem were considered. Here, connections could be made with results from Knuth [Kn73] on sorting and selection. It was observed that the closest pair of points must be neighbors and that finding all pairs of neighbors would require information similar to a sort [Sh75]. The furthest pair of points would be the maximum and minimum of the set. Identifying them would require an algorithm for finding the max and min of a set of n numbers. Thus were derived informal arguments that n log n was asymptotically correct for the closest pair of points and linear time was necessary and sufficient for the furthest pair.

Next, attention was focussed on higher dimensions. In 2 dimensions, more geometry was necessary in order to establish which points were neighbors and had to be considered as potential closest pairs in the set. Here, two methods ultimately emerged for solving the problem.

The first method was based on building a structure that would contain all potential nearest neighbors. This structure assigned to each point a region containing points closer to it than to any of the others. As such, it could be constructed from the perpendicular bisectors generated from each pair of points. Under further study, it became clear that in the plane the structure was actually a planar subdivision. This followed since the region corresponding to each point was convex and hence the structure corresponded to a planar graph of n faces and hence a linear number of vertices and edges. Furthermore, it was discovered that this structure could be built by a divide-and-conquer method in O(n log n) operations. Finally, it was learned that this structure had been previously defined in other contexts. Among its other names were Voronoi diagram and Dirichlet tesselation. From the structure arose the first sub-quadratic algorithms for finding nearest neighbors in 2 dimensions. The algorithms consisted of finding the Voronoi diagram as a

planar graph. Then, all potential nearest neighbors were given by considering edges of the dual of this graph. This dual was known as the Delaunay triangulation and had been previously discovered in the 1930's. Since there were only linearly many edges in the Delaunay triangulation, the minimum distance could be found in linear time. This method was discovered independently by Shamos-Hoey and Reiss at Yale in 1975. The details appear in [SH75]. Reiss implemented his algorithm as part of a BLISS program to perform other functions. However, there was no sense at the time of reasonable data structures for doing such a task and his implementation was quite complex.

This algorithm seemed to be overkill to solve ''so simple a problem'' and alternatives were sought. Strong at IBM and Bentley [Be76] discovered a simpler divide and conquer method to solve the problem. Their algorithms divided the points into a left set L and a right set R. For each set, the nearest neighbors were found along with their distance d(L) and d(R). Next, they observed that a point in R could only be within distance d(L) of a constant number of points of L. Hence, a merge of two sets with shortest distances known could be done by considering only a linear number of distances and hence linear time would suffice.

Each of the two methods has its advantages that were well known by 1977. The Voronoi diagram method also solves numerous other problems on planar point sets (see [PS85]). The Strong-Bentley method extends more naturally to higher dimensions while the Voronoi diagram on n points can be of size $O(n^{d/2})$ in dimension d. However, by 1977 neither method was sufficiently well understood to lead to a clean implementation. Even for planar point sets, it was not understood what data structure ought to be used to implement the Voronoi diagram.

Determining the furthest distance was equivalent to determining the diameter of the set. The diameter of the set was shown to be the diameter of the convex hull. This followed from the observation that if one or both of the furthest points did not lie on the hull, a larger distance was possible. Next, it was shown that potential endpoints of a diameter must be antipodal. That is, each edge could be considered to have a range of angles that tangents to it (or one of its endpoints) might make. When the ranges for 2 vertices involve values differing by π, they are considered to be antipodal. Connections with work of Erdos[Er46] established that there were only linearly many pairs of antipodal points. This yielded the linear time bound on the scan step of the diameter algorithm.

In 3 dimensions, it was again shown that the diameter would be determined by a pair of points on the convex hull. It is known that the diameter can be realized at most a linear number of times[Gr56]. This corresponds to having only a linear number of antipodal pairs in the plane. However, after a few false starts, the problem of finding the diameter in fewer than a quadratic number of operations remained open.

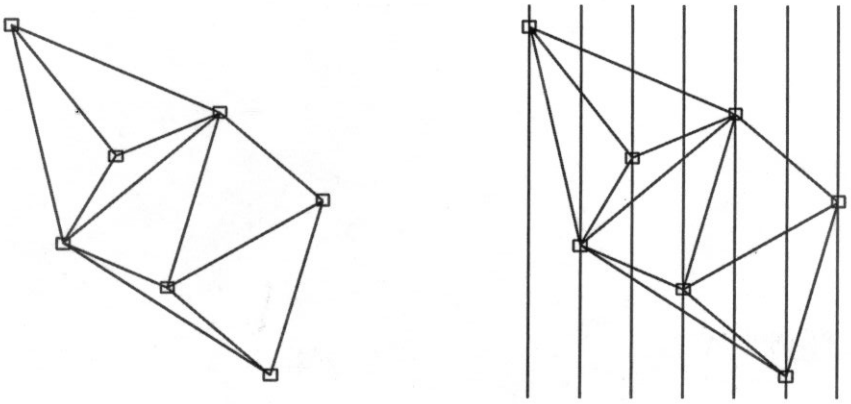
Thus, the state of the art for finding nearest neighbors as of 1977 was well understood. Nearest neighbors required n log n operations by the element uniqueness lower bounds. n log n algorithms were known in all dimensions by a divide and conquer method. There was also a method in 2 dimensions based upon the Voronoi diagram that had numerous other applications. However, no clean method of implementing the Voronoi diagram or handling degeneracies of point sets (e.g. 4 cocircular points) was known and there was no accepted data structure for the problem. Curiously, at about this time, Baumgart was writing a thesis in computer vision [Ba74] that would provide such a structure. With regards to furthest distances, linear time was known to be necessary and sufficient in 1 dimension, n log n was known to be necessary and sufficient in 2 and the problem was open in higher dimensions.

Planar subdivision searching

Planar subdivision by polygons in dimension 2 is the generalization of the subdivision of a line by a set of points in dimension 1. There are numerous applications of an algorithm for searching planar subdivisions. For example, the Voronoi diagram presents applications where we might want to locate the region of a point in a planar subdivision. Similar problems arise in using finite element analyses to solve partial differential equations. Also, the problem arises in the ''post-office problem'' as stated by Knuth. Finally, in higher dimensions, the knapsack problem can be posed as a problem of searching spatial subdivisions. Planar subdivisions for applications mentioned here might arise in either of two similar contexts. Either they arise from the regions formed from the arrangement of a set of n infinite lines. Or, they might arise from the Euclidean embedding of a planar graph. In either case, the structure to be searched is the same. And, we can assume that the structure can be built once and will remain static for a large number of

searches. Hence, it is reasonable to think of preprocessing to build a usable search structure that can be rapidly searched.

The central problem here is that of finding an ordering on the regions of the planar subdivision. The earliest methods [DL74,DL76] did this by reducing the problem dimension by one using "slabbing methods". Here the observation was that the planar subdivision could be divided into a set of slabs that could be easily searched. This was done by creating slabs in which the regions were monotonically ordered. Furthermore, the slabs were created in monotone order also. Thus, the unknown problem of how to do binary search in two dimensions was reduced to the problem of doing 2 binary searches. The first was a binary search through an ordered set of slabs to find the correct one and the second was a search through an ordered set of regions in the slab to find the correct one. In Figure 4, we show the effect of overlaying slabs on a planar subdivision. Notice that two segments never cross within a slab. However, the storage required can be such that a segment has to be stored within every slab.



a planar subdivision and its slabs

Figure 4 -- a planar subdivision and its slabs

The slab algorithm resulted in an algorithm requiring $O(n^2 \log n)$ preprocessing time and $O(n^2)$ preprocessing space in the worst case. However, the retrieval time was 2 log n accounting for two binary searches of n items each.

Thus, a solution existed. The challenge now became to improve upon the running times. For such algorithms, complexity is measured via triples of numbers (PT(n),PS(n),ST(n)) where PT(n) (resp. PS(n)) represents the preprocessing time (resp. space) and ST(n) represents the search time. It became clear that PT(n) and PS(n) could be made linear (e.g. by doing nothing), but that if ST(n) was to be sublinear then PT(n) would have to be at least n log n. Then the question arose as to whether, it was to possible to achieve a solution that was (n log n, n, log n). Lee and Preparata [LP79] reduced the best known algorithm to $(n \log n, n, \log^2 n)$

Finally, in an elegant fashion, Lipton and Tarjan [LT77a,LT77b] achieved a solution of (n log n, n, log n). Their methodology was to show that divide and conquer could be applied to the problem of splitting a planar graph. In linear time they were able to find a separator that divided the graph into two parts each containing a positive fraction of the original vertices. Furthermore, the separator was small. While their method was not easily implementable, it did provide the first proof that divide and conquer could be made to work in 2 dimensions as it had in 1. This supported the belief that sorting could be applied to higher dimensional geometries.

Range searching

The range searching problem has its roots in the implementation of database-like queries for determining ranges of values. We might be given a set of attributes (e.g. salary, age, height and weight) and represent employees of a company as points in R^4 corresponding to their 4-tuple of these attributes. Then, we might probe in order to determine all employees satisfying a set of ranges simultaneously. This might

consist of finding all employees within a certain age range, having salary between specified limits, ... Usually, the database is reasonably static so that preprocessing of the points can be done off-line in support of faster queries. Knuth posed the problem of designing a data structure [Kn73] for multidimensional searching problems. The search queries to be supported are similar to those considered in the previous section, however subtle differences lead to algorithms of a very different form.

Bentley was the first to propose algorithms for doing rectangular range queries [Be75,BS75,FB74]. He proposed a structure called the "quad tree" for doing the search in 2 dimension. Knuth later named the generalization of this search structure the "k-D tree". The quad tree is relatively simple. The problem of searching in 2 dimensions is reduced to two methods of searching in 1 dimension just as we saw in the slab methods of the previous section. However, here the reduction occurs in an iterative fashion that is symmetric. The resulting algorithm is one that has good average case behavior, though its worst case can be bad [LW77].

The quad tree is a 2 dimensional binary search tree that alternates search directions. So, given a set of points in the plane, a divider is found parallel to the y-axis. Next, each of the two regions is divided by a divider parallel to the x-axis. The alternating process continues until regions contain only 1 point. An example of the process is shown in Figure 5.

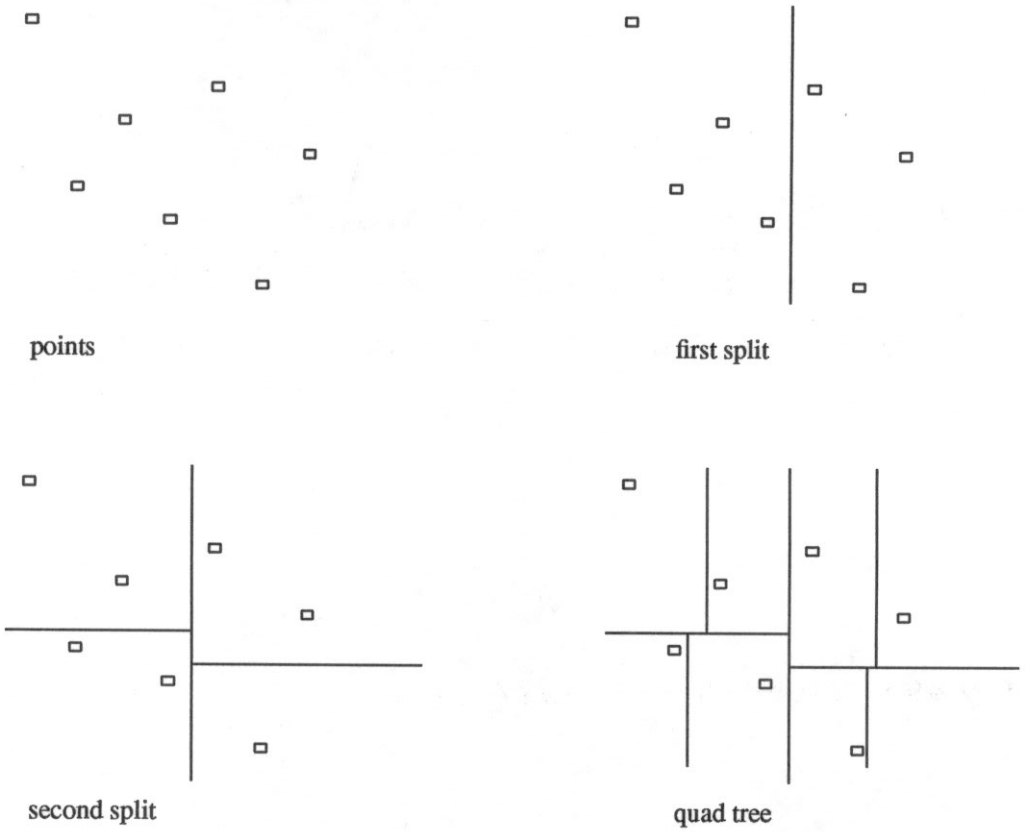


Figure 5 -- Splitting to form a k-d tree

Searching of this structure now can be done by following a path down the binary search tree corresponding to the subdivision given. If we wish to determine how many points lie within a rectangle, we determine which of the two regions at a node the rectangle intersects and use this information to proceed. As opposed to other binary trees, it could be the case that both subtrees need to be followed.

Quad tree based algorithms were already popular within computer graphics and pattern recognition by 1977[Hu78,HS79]. Indeed, the method of traversing a quad tree uses techniques similar to those proposed by Warnock [Wa69] for use in hidden surface removal. However, the discoveries appear to have been independent.

The subsequent development of range searching techniques after quad and k-d trees is interesting within the context of this paper. As opposed to the other problems we consider, this is an example where the first algorithm found was the practical one and led work of theoretical interest. In all other cases, the theoretical work typically happened before 1977 and the work of the last decade has involved refining theoretical insights and looking for simplified algorithms that are implementable. For range searching, the initial algorithms were driven by practical considerations and are quite implementable. The algorithms that followed were born out of theoretical considerations. Most of these theoretical considerations sought to fix the bad worst case possibilities of range searching and extending range searching to work for non-rectangular search domains.

Intersection problems

No study of geometry would be complete without a discussion of intersection problems. Much of the motivation for intersection comes from a desire to display geometries on a graphics screen. In 2 dimensions, we are often concerned with a screen full of overlapping windows and our goal is to determine which windows need to repainted or a paint order for windows. In 3 dimensions, intersection detection and computation forms the basis of most hidden surface algorithms. Both of these problems involve determining intersections among line segments and identifying which (if any) pairs from a set of line segments intersect.

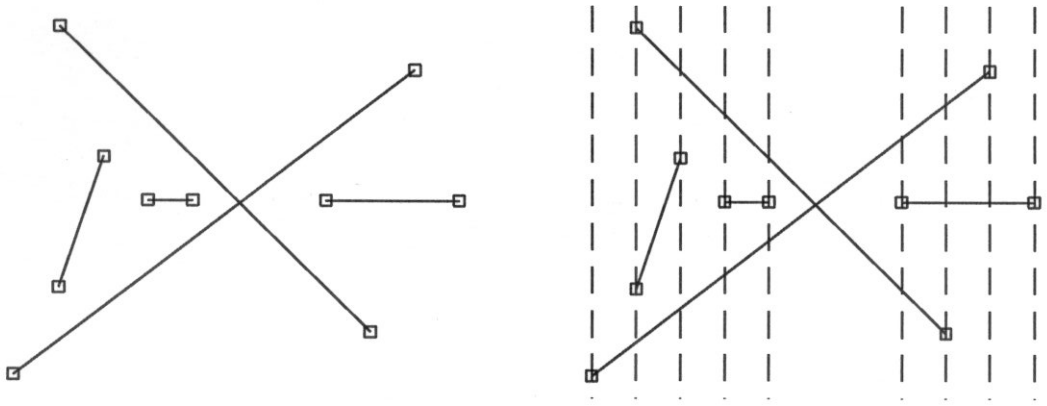
This problem was first considered by Shamos and Hoey [SH76] who developed a method based upon the notion of plane sweep to trace across a set of segments and determine whether any pair intersected. Their technique builds on ideas similar to the slabs methods for planar subdivision searching. Its implementation generalizes the implementation of scan-line based hidden surface routines as developed by Watkins [Wa70].

The plane sweep uses slabs defined by endpoints of the segments. We consider these endpoints to be projected on the x-axis and slabs to extend vertically. Segments at any value of x can be ordered by their y coordinates. And, within a slab, this order can change only if an intersection has occurred. On the boundary between slabs, a new line segment is inserted or deleted. For n line segments, there are 2n slabs. A very naive algorithm is to determine the slabs and sweep across slabs determining for each slab if it contains an intersection. This can be done in time linear in the number of segments within the slab since the order of the segments at the start of the slab is known. This leads to a quadratic algorithm.

Shamos and Hoey improve upon this algorithm by observing that if two segments intersect, they must have been adjacent in the ordering of segments at some point (not necessarily at a slab boundary). Furthermore, one of 4 situations must have occurred as seen in Figure 6.

1. When the second segment was inserted, they were adjacent in the ordering of segments.
2. When the second segment was inserted, they were not adjacent in the ordering of segments, but all segments occurring between them either intersect one (or both) of the segments or were deleted before they intersected.
3. When the first segment was deleted, they were adjacent in the ordering of segments.
4. When the first segment was deleted, they were not adjacent in the ordering of segments, but all segments occurring between them either intersect one (or both) of the segments or were inserted after they intersected.

As they do their sweep, Shamos and Hoey are careful to always maintain the order of the line segments. This is done by using a balanced tree scheme so that insertions and deletions can be done at a cost of O(log n). As a result, they can scan across the segments in O(n log n) operations. Whenever a segment is about to be inserted, it is checked against its neighbors for intersection. And, whenever a segment is about to be deleted, its neighbors are checked to see if they intersect each other. Thus, any two segments that will ever be adjacent in some ordering are checked for intersection. By appealing to the list of situations given above, it can then be shown that if there is an intersection it will be detected. Furthermore, the running time of the algorithm is O(n log n) because the x coordinates of the endpoints must be sorted and because O(n) insertions and deletions must be done. Finally, this is optimal by appealing to the element uniqueness problem and showing that a sort must be done.



planar segments their slabs

Figure 6 -- Setting up slabs for a plane sweep algorithm

The only failing of the Shamos-Hoey algorithm is that it will not necessarily find all intersections. Bentley and Ottman [BO78] proposed an extension that does compute all intersections. For this algorithm to work, more slabs are necessary. Every time an intersection is found, an old slab must be subdivided to create a two new ones. With appropriate modification, it can be shown that the Shamos-Hoey algorithm always detects intersections **before** they occur and therefore, we will always know in advance that slabs will have to be created. When we reach an intersection point corresponding to the subdivision of a slab into two, we must change the order of the two segments that intersected. This requires O(log n) operations. We also do a test of these segments against their new neighbors to see if there is any intersection. Again, this can be done in O(log n) operations. Putting this together yields an algorithm with running time $O(n \log n + I \log n)$ for reporting all intersections where I is the number of intersections reported. In cases where I is small, this represents a significant improvement over previous methods. However, in cases where I is large, this algorithm will actually be slower than the naive algorithm.

The primitive operations needed to make either of these algorithms work are minimal and easily specified. Thus, they can be easily extended to compute intersections among sets of rectangles and sets of convex polygons.

Conclusion

This completes our survey of the state of the art in computational geometry as of 1977. Our survey is far from complete but is designed to present early ideas so that we can see how the field has progressed during the past decade. The interested reader is referred to [PS85] and references therein for further details.

There are a few similarities among the problems stated above that we should observe before seeing how the field has progressed. First, although the problems appear to have very different statements, their solutions seem to always return to similar considerations. Next, the main failing of the solutions given here (with the possible exception of k-d trees) is that they do not easily lend themselves to implementation. And as we casually mention, many of the solution techniques are similar to those that were being developed or had been developed to solve problems in ''continuous computational geometry''.

In each problem, there is an aspect of the solution that reduces to either sorting or searching. For the convex hull, the presence of sorting is obvious. Gift wrapping corresponds to finding the next largest and Graham scanning corresponds to sorting and then eliminating interior points. The Voronoi diagram constructed for finding nearest neighbors merely consists of finding each point's (multiple) neighbors in the same way that a 1-dimensional sort finds each point's 2 neighbors. The convex hull represents extremal points and finding antipodal pairs for determining furthest neighbors merely involves identifying extremal pairs. Planar subdivision and range searching involving coping with searching in 2 (or more) dimensions. They achieve solutions by finding means of dividing a 2-dimensional search into 2 appropriate 1-

dimensional searches. The divide and conquer techniques for searching planar graphs extend these techniques by finding good dividers without respect to direction rather than alternating direction (as in quad tree) or favoring one direction over another (as in slab method). Finally, the plane sweep method builds upon the same ideas as the slab techniques and leads us to wonder whether a divide and conquer extension will easily emerge.

We have attempted to present enough details about the techniques used to solve these problems to allow the reader to understand the methods. Time and space made it impossible to provide complete details. Implementing the techniques described here would in most cases be a formidable task. There are few clues as to the data structures that ought to be built to manage the geometry making implementation a non trivial task. And, we have assumed that degeneracies either do not occur or cause little problem if they do. While this is a reasonable assumption for the algorithm designer, it is unfortunately the bane of the implementor's existence.

In 1977, there were very few links between the two computational geometry communities. Workers in one community were seldom aware that fellow workers in another community were considering similar techniques to solve similar problems. Yet, we see many instances above of duplication of effort and a visionary in 1977 could have seen the seeds planted that would ultimately bring the communities closer together.

4. NOW

The next decade of computational geometry (as applied to the problems considered here) is marked by two trends. First, we see the development of more sophisticated techniques to resolve many of the problems that remained open or hadn't even been considered by 1977. Next, there is the development of a sophistication in determining methods of greater practicality and of techniques for implementing the algorithms of the past. In some ways these trends go at cross purposes. The development of more sophisticated techniques leads to algorithms of ever increasing complexity of description. The quest for implementable algorithms requires algorithms of ever decreasing complexity of description. As we watch the next decade emerge within this section, we will often observe these forces working. However, they appear to work together productively since the development of a more sophisticated technique almost always builds upon a simpler description of the previously known methods.

Convex hull of a point set

By 1977, most of the important questions about convex hulls had been asked and many had already been answered. The connections between computing convex hulls and sorting were already well established. What remained were questions about optimal convex hull algorithms in cases where the output size was small. For example, the gift-wrapping algorithm of the previous section runs faster than the Graham scan algorithm on random examples. Its running time is O(nh) where n is the number of points and h is the number of points on the convex hull. Typically h is O(log n) and so both the gift wrapping and Graham algorithms have asymptotic running time which are the same. Since the gift wrapping algorithm has a cleaner implementation, it is preferred. However, for a set of n points all on the hull, it gives quadratic performance. The situation is worse in higher dimensions. Here, there is a lower bound of $O(n^{d/2})$ (resp. $O(n^{(d-1)/2})$) was applied in even (resp. odd) dimensions. This bound comes from worst cases where it could take this amount of effort to report the output. However, in practice the output size is significantly smaller.

It is known that computing components of the convex hull is polynomially reducible to linear programming [DR80]. Three key questions arose. First, if we know something about the input can we use this information to compute the convex hull. Secondly, in two dimensions, is it possible to have an algorithm with running time optimal in the number of points on the hull even if this number is not known in advance. Finally, can linear-time linear programming algorithms [Me83,Me84] and other techniques be used to determine all components of the convex hull in time linear in their number. Progress has been made on all of these questions in the past decade.

Perhaps the most interesting instance of the first question is the problem of determining the convex hull of a set of points given as the vertices of a simple polygon. Here, the issue is that the points have

already been sorted in some order. That is, by virtue of forming a polygon, they have an order. We would be able to do a Graham scan in linear time if we knew an ordering about an interior point. However, the order we are given on the boundary may not correspond to such an ordering. Hence, the Graham scan must be modified to determine when there has been an inappropriate backup. Here, the algorithms proposed typically compute an upper hull and a lower hull, keeping track of turns to assure that vertices are not incorrectly deleted [GY83,Le83]. It remains open to characterize other situations where the input contains sorting information that can be used to simplify the hull computation.

Next we turn to the situation where not all points are on the convex hull. Here, the convex hull could potentially be computed in o(n log n) operations if o(n) points were on the hull. To do so requires not sorting the points since this requires O(n log n) operations. Kirkpatrick and Seidel [KS86] were able to do so. Their algorithm finds a method of avoiding the recursive technique of finding 2 halves and merging. That technique has a running time which satisfies the recurrence relation

$$T(N) = 2\,T(N/2) + O(N).$$

This method can be slow because it will compute many hull edges that are removed in later mergings. Instead, they find a merging edge in linear time (via a variant of the linear time linear programming method [Dy84,Me83]) and reduce to the problem of solving two subproblems. This leads to a time bound determined by the recurrence relation

$$T(N,h) = T(N/2,h_1) + T(N/2,h_2) + O(N)$$

where h_1 and h_2 represent the number of points on the two smaller hulls and $h_1 + h_2 = h$. This yields an algorithm of running time O(n log h) that is aptly titled *the ultimate convex hull algorithm.*

In higher dimensions, two methods have evolved to compute convex hulls. One is an extension of the gift-wrapping technique of Chand and Kapur [Sw85]. The other is a method named beneath-beyond [Se81]. Using either method, near optimal algorithms can be found in the case where the convex hull is as bad as possible. However, the running time is exponential in the dimension even if the output is small. In his thesis, Seidel has improved the situation [Se86]. By using the linear-time linear programming algorithms, he computes convex hulls in dimension d. If the input consisted of m points and the output had F facets, he can enumerate all facets in time $O(m^2 f(d) + d^4 F \log m)$. Unfortunately, f(d) is the ''constant'' term that arises in the Megiddo linear programming algorithm and grows exponentially with d. This algorithm is not practical, though it yields insights that should lead to practical implementations during the next decade of computational geometry.

Thus we see that significant progress has been made on the convex hull problem since the time of the Chand-Kapur and Graham results. It is now the case, that theoretically optimal (or nearly optimal) algorithms are known in all cases. Also, practical algorithms are understood for many situations. There remain a few implementation issues that become relevant for applications where convex hull of large point sets were to be computed or where only part of the structure of the convex hull is needed. Many of the techniques given in this and the previous convex hull section find use in real problems of statistics, pattern recognition and classification algorithms (see e.g. [Ba84]) in describing and classifying point sets. The convex hull is an especially desirable indicator to be used in recognizing patterns because of its rotational invariance.

Finding nearest neighbors

As we saw in the previous section, the rediscovery of the Voronoi diagram provided a framework for discussing nearest neighbor problems. Furthermore, by 1977, there were known algorithms requiring O(n log n) operations for building Voronoi diagrams. While one such algorithm had been implemented [Re77], implementation was a non-trivial task. In addition to the problem of finding a good data structure upon which to build the implementation, there were questions of numerical accuracy and degeneracy. The divide-and-conquer nature of all known algorithms was cause for concern in terms of resulting efficiency and numerical stability of algorithms. What was needed was a simple yet complete explanation of the Voronoi diagram. There also remained the question of higher dimensional Voronoi diagrams. However, Brown established a relationship between convex hulls and Voronoi diagrams [Br79a] that related convex hull computation in dimension d+1 to Voronoi diagram construction in dimension d. As a result, the results of Seidel [Se86] given in the previous section can be applied to higher dimensional cases.

Significant work was done on the development of good Voronoi diagram algorithms during the decade 1977-1987. Authors sought clean formulations of algorithms that would be implementable, extensions to the Voronoi diagram that would also handle line segments and curves and data structures to use in doing the computation. This work is presented in two papers [Fo86,GS85] that resolve many of the outstanding issues and represent the current state of the art.

Guibas and Stolfi[GS85] present a generalization of the winged-edge data structure of Baumgart [Ba74] to a quad-edge structure that can represent planar subdivisions and their duals simultaneously. Their work can even represent subdivisions on non-orientable manifolds. Their development is complete justifying the choice of the quad-edge as the data structure of choice for the representation of subdivisions of 2-manifolds (ie planar subdivisions and 3-dimensional polyhedra). They are able to give a one page implementation of the Voronoi diagram (and its dual, the Delaunay triangulation) based on their data structure. Since the manipulation of their data structure is straightforward, the result is a simple implementation of the Voronoi diagram that is reasonably robust. They can identify and handle many degeneracies, though they remain unable to identify a set of (more than 3) cocircular points efficiently. This identification would be desirable in constructing Delaunay triangulations since the resulting regular polygon should not be triangulated.

Fortune [Fo86] presents the first optimal algorithm for computing the Voronoi diagram that is not based upon divide and conquer. He uses a sweepline technique. In sweeping across the plane, the Voronoi region of a point will be encountered before the point is reached. Any sweepline algorithm must have a method of identifying the region's beginning as it occurs. An algorithm that does otherwise is destined to have quadratic running time. Fortune introduces a transformation of the plane that causes a point's Voronoi region to be first encountered at the point when doing a planar sweep. The result is that he is able to insert the region at that point, in a manner similar to the plane sweeps of [SH76,BO78] described above. On insertion, he can identify when adjacent regions will change discontinuously in shape or disappear adding an event to the event queue. There can only be a linear number of events* and hence the algorithm has O(n log n) running time. His transformation is a conceptual device for understanding the algorithm. By undoing the transformation, Voronoi regions are identified as they appear. He gives further transformations for cases where the Voronoi diagrams of line segments or weighted point sets are to be computed.

Once again, the progress of the next decade is exciting to behold. The Voronoi diagram has been transformed from a theoretical device used to solve problems into a tool that is used in practical situations. This progress obviously arises from the development of algorithms that can be used in the practical situations. Surprisingly, it also arises from the dissemination of some of the early papers on the subject to engineers in related fields who have problems that need such a structure. Thus, it seems likely that these applications will drive the next decade of work on the Voronoi diagram and Delaunay triangulation. A step in this direction is the thesis work of Mike Laszlo [DL87,La87]. This work provides an extension of the work of Guibas and Stolfi to spatial subdivisions. There appears to be the potential for the application of this work to numerous problems involving computational geometry as well as applications to finite element calculations (see e.g. [JB87]).

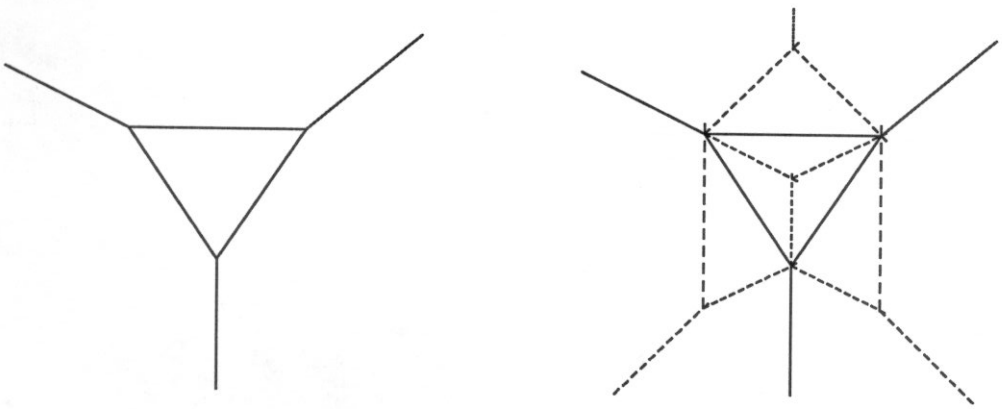
Planar subdivision searching

The situation regarding planar subdivision searching in 1977 was similar to that of Voronoi diagrams. The Lipton-Tarjan result had solved the theoretical problem but was not of practical significance. It remained for the practitioners to propose techniques that would be simpler and hence likely to apply in practice. Two themes emerged in the development of practical techniques.

The first is the notion of hierarchical search. Kirkpatrick [Ki83] proposed this as a technique for representing a planar subdivision. The basic idea is that when searching a planar subdivision, more and more detail are needed about less and less of the planar subdivision as the search progresses. This suggests a technique whereby the planar subdivision is represented by a tree. At the root of the tree is a small planar subdivision consisting of a constant number of vertices and regions. This can be searched in constant time. When the appropriate region has been identified, only this region of the subdivision is refined by adding

* Note that this differs from the sweepline for finding all segment intersections where there can be a quadratic number of events.

vertices and so further subdividing the region. This is shown in Figure 7. Note that we only visit one of the subdivided regions.



root subdivision

after 1 subdivision

Figure 7 -- The first step of a hierarchical planar subdivision

This process continues for log n steps. Then, in O(log n) time, the search has been completed. The hierarchical technique appears to be a basic technique of computational geometry [DS87] having also found application to polyhedral intersection detection [DK85].

The other approach follows [EGS87] the topology of the arrangement and applies a divide and conquer technique. It is based upon ideas that led to some of the initial methods for solving this problem. The idea is simple. A chain of edges is found that divides the subdivision in half. The chain also has the property that it is monotone and hence can be searched quickly. Furthermore, future chains build upon the initial chain so that all searches can be completed in time O(log n). A similar technique is given in [ST86].

Once again, significant progress was made on a basic problem of computational geometry. The progress on planar subdivision searching provides the insights necessary to lift searching from a one dimensional primitive to a well understood operation in 2 dimensions. The techniques described above have made the problem sufficiently practical to allow its use in application systems. Indeed, as graphical editors and window systems become more complex, naive algorithms are demonstrating behaviors that are insufficient and these techniques will be needed. There remain open the problems of extending searching to spatial subdivisions. The results in [DL74,DL76] stood unchallenged for almost a decade. Recent work [Ch85a, Cl85] has improved upon the space bounds and gave probabilistic algorithms that vastly improve the older results. Extensions beyond dimension 3 remain open.

Range searching

As opposed to the other problems we discuss in this section, range searching is a situation where the practical has driven the theoretical. The quad tree was proposed as an interesting search structure and remains the data structure of choice in numerous applications [Sa84]. However, the problem of optimal range searching has attracted significant theoretical interest. Here, the search problem is the halfspace searching problem. So, points are to be preprocessed so that those lying in a halfspace can be determined efficiently. The search structure is to require only linear space and the search time is to be sublinear. It has been shown that this searching model embodies all polyhedral range searches [DE84]. It has also been shown that with more storage, all searches can be achieved in O(log n) time [EMK82]. Finally, a lower bound of $O(n^{1/3})$ is known on the problem [Fr81].

The first observation that sublinear halfplane searches were possible is due to Willard [Wi82] who produced an algorithm of search time $O(n^{.774})$. The method of Willard consisted of creating 6 regions in the plane each containing an equal share of the points and such that a line would intersect no more than 4 of them. Then, a search could be done by reducing a problem to 4 problems of 1/6 of its original size in constant time. This yields the recurrence relation

$$T(n) = 4T(n/6) + c$$

from which his result follows. Edelsbrunner and Welzl [EW82] improve this by reusing the results of previous queries to build a structure they call the conjugation-tree. Their work is built on the 2-dimensional *ham sandwich* theorem. Given a set of n points in the plane and a line dividing the points into 2 sets each having no more than n/2 points, they show how to construct a line that further subdivides into 4 sets each having no more than n/4 points. This yields a search algorithm of recurrence relation

$$T(n) = T(n/2) + T(n/4) + c$$

with solution $O(n^{.695})$.

In higher dimensions, there was a flurry of activity resulting in an upper bound of $O(n^{.899})$ [DEY84] in 3 dimensions and an observation that the commonly used technique would not extend past 4 dimensions [Av85]. Next, Cole [Co85] proposed an alternative technique using portions of multiple hyperplanes for the subdivision rather than simply building quadrants or octants as had been done in dimensions 2 and 3. He was able to achieve an upper bound of $O(n^{.977})$ in dimension 4. Next, there followed a construction built upon Cole's that achieved a query time of $O(n^{\alpha})$ for $\alpha < \frac{\log_2(2^d-1)}{d} + \varepsilon$ for any $\varepsilon > 0$ [YY85]. Finally, Haussler and Welzl [HW86] showed that a probabilistic technique could be used to achieve a query time of $O(n^{\alpha})$ for $\alpha < \frac{d(d-1)}{d(d-1)+1} + \varepsilon$ for any $\varepsilon > 0$. Their construction is related to the Vapnik-Chervonenkis dimension of probability theory. Curiously, their construction is non-constructive. They show that an algorithm exists but cannot demonstrate one. Their ε-net construction appears likely to have significant application to related problems of region counting in arrangements [GS87]. A similar construction was discovered independently by Clarkson [Cl86].

As mentioned above, range searching has proceeded in a different direction from that taken in the other four problems we discuss. There has been some development of techniques for using range searching and new applications. However, the significant advances during the past decade have been theoretical. Elegant mathematics has resulted from the analysis of extensions to what is basically an intuitive method. It is likely that these extensions will generate new results of practical interest during the next decade.

Intersection problems

Since the work of Bentley and Ottman, there has been significant progress on intersection problem. There have been both theoretical and practical improvements to their algorithm. Also, numerous other directions in intersection have been pursued. Furthermore, work on significant practical problems, most notably hidden surface removal has begun to occur at both the theoretical and practical levels.

On the basic problem of finding all intersections, improving the upper bound of $O(n \log n + I \log n)$ has been a nagging open problem. Mairson and Stolfi [MS82] showed that if there were two sets of line segments, n painted red and n painted green such that all intersecting pairs involved a red segment and a green segment, then $O(n \log n + I)$ time sufficed to report all intersections. Myers [My85] produced an algorithm that had an expected running time of $O(n \log n + I)$ and was efficient in practice. Finally, Chazelle [Ch84] improved the upper bound to $O(n \log^2 n/\log\log n + I)$. Kalin and Ramsey [KR84] did significant empirical studies of algorithms for the problem and concluded that Myers's was the fastest in practice and that further improvements of his algorithm were possible. Their conclusion was that for randomly generated line segments, a technique that wisely combines enumeration and bucketing will give the best behavior. While, most pictures sent to a hidden surface remover are not random, it is reasonable to believe that their observations will extend.

Most of the intersection work is geared towards extending methods towards objects of increasing complexity. The holy grail of intersection problems is an algorithm that will operate on a database of arbitrary objects and allow for insertions and deletions of objects as well as determining all objects that a new object intersects. The applications of such a technique are manifold. Towards this end, there has been work on understanding intersections of convex polygons [Sh75], convex polyhedra [MP78,DK83,DK85,CD87] and curved polygons [DSV86]. Also, searching techniques have been developed to strike at the problem of a dynamic database [Ch86]. In all cases, the central problem is the

same. One is always seeking a method of sorting objects that do not have a total order. The goal of such sorting is to be able to later search a database of such objects to determine overlap. An interesting technique that gets at the essence of this problem is the work of Chazelle [Ch85b] on embedding intersection problems in higher dimensional spaces and solving them by subdivision searching techniques.

The next step is to relate this work to hidden surface removal problems in a meaningful fashion. This has begun with theoretical work [De86] giving lower bounds for worst case hidden surface removal. Of more immediate potential practical importance are results that use theoretical techniques to study key problems. Notable in this category is the work of Hubschman and Zucker on frame-to-frame coherence [HZ82] and the forthcoming thesis of Panduranga [Pa87]. The former paper is a first attempt at using object space techniques to represent an object for multiple hidden surface removals. The latter considers the problem of ray tracing in object space by tracing back reflections onto primary objects. As we've seen above, intersection problems represent an area where theory has applied in practice and the current progress suggests that this trend will indeed continue.

Conclusion

This section shows the excitement that the past ten years have produced within computational geometry. The literature is growing at such a rate that it was only possible to scratch the surface of developments on the five problems that were active a decade earlier. Beyond these problems, there has been significant work in a number of other directions that go unmentioned here. This decade has seen computational geometry pulled in two opposing directions. On the one hand, the quest for improved theoretical results has caused the field to make contact with a mathematical community that may provide the tools for these improvements. In the other direction, the quest for relevance has continued contact with researchers in continuous computational geometry. computer graphics, solid modeling and robotics. The combination of these pulls has led to an interesting and applicable intellectual discipline.

What is most significant about the improvements during the period 1977-1987 is the level of ''technology transfer''. As the results mentioned here were developed, they became known beyond the theoretical computer science community from which discrete computational geometry first emerged. Practitioners were quick to recognize the potential relevance of this field and cross-fertilization was a key theme in the second decade. The momentum of the field is such that this trend will clearly continue. This momentum is fueled in large part by workshops such as this and related workshops that bring computational geometers together with their mathematical colleagues. As the field grows, the community of scholars involved reaches a size and excitement that neither Robin Forrest nor Mike Shamos could have realized when they named it.

5. NEXT?

It is necessary to end a paper such as this with some prediction of what the next decade will bring. Obviously, more researchers will be attracted to the field, old problems will be solved, new problems will emerge, the process of turning theoretical results into practical algorithms will continue, ... as is the nature of intellectual disciplines.

Beyond the obvious, there appears to be one need that is just being recognized and deserves mention. This is the concern about numerical issues when translating efficient algorithms such as those mentioned above into practice. Very few members of our community are well trained in numerical analysis and most techniques of numerical analysis do not easily extend to problems that are geometric. Thus, there is a need to not only apply existing techniques but also to identify the limitations of such techniques and find methods of extending them. These problems will keep us all busy and excited until 1997!

Acknowledgements

It is a pleasure to acknowledge the careful readings of this manuscript by Chris van Wyk and Diane Souvaine which greatly reduced the number of errors and improved clarity. I'd also like to thank all of the people who made compuitational geometry fun both then and now.

6. References

[Av85] Avis, D., On the partitionability of point sets in space, ACM symposium on computational geometry, 1985, 116-120.

[Ba84] Baird, H. , Model-based image matching using location, PhD Thesis, Princeton University, May, 1984.

[Ba74] Baumgart, B. G., Geometric modeling for computer vision, Stanford U., Computer Science Dept. Technical Report, STAN-CS-74-463.

[Be83] Ben-Or, M., Lower bounds for algebraic computation trees, Proceedings of the 15th ACM Symposium on the Theory of Computing, 80-86, 1983.

[Be75] Bentley, J. L., Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 18, 509-517, 1975.

[BO78] Bentley, J.L. and Ottman, Th., Algorithms for reporting and counting geometric intersections, Carnegie-Mellon University, August 1978.

[BS75] Bentley, J.L. and Stanat, D.F., Analysis of range searches in quad trees, *Information Processing Letters*, 3, 170-173, 1975.

[Be76] Bentley, J.L., Divide and conquer algorithms for closest point problems in multidimensional spaces, PhD thesis, University of North Carolina, 1976.

[Br79a] Brown, K. Q., Voronoi diagrams from convex hulls, *Information Processing Letters*, 9, 223-228, 1979.

[Br79b] Brown, K. Q., Geometric transformations for fast geometric algorithms, PhD thesis, Carnegie Mellon University, December, 1979.

[CK70] Chand, D.R. and Kapur, S.S., An algorithm for convex polytopes, *JACM*, 17, 78-86, 1970.

[Ch84] Chazelle, B., Intersecting is easier than sorting, Proceedings of the 16th ACM Symposium on the Theory of Computing, 125-134, 1984, also *JCSS*, to appear.

[Ch85a] Chazelle, B., How to search in history, *Information Control*, 64, 77-99, 1985.

[Ch85b] Chazelle, B., Fast searching in a real algebraic manifold with applications to geometric complexity, Proc. CAAP'85, LNCS, Springer-Verlag, 1985.

[Ch86] Chazelle, B., Filtering search: a new approach to query-answering, *SIAM J. on Computing*, 15, 703-722, 1986.

[CD87] Chazelle, B. and Dobkin, D., Intersection of convex objects, *JACM*, 34, 1-27, 1987.

[Cl85] Clarkson, K., A probabilistic algorithm for the post office problem, Proceedings of the 17th ACM Symposium on the Theory of Computing, 175--185, 1985.

[Cl86] Clarkson, K., Further applications of random sampling to computational geometry, Proceedings of the 18th ACM Symposium on the Theory of Computing, 414-423, 1986.

[Co85] Cole, R., Partitioning point sets in 4 dimensions, ICALP '85, Springer-Verlag Lecture Notes in Computer Science, 194, 111-119, 1985.

[De86] Devai, F., Quadratic bounds for hidden line elimination, ACM symposium on computational geometry, 269-275, 1986.

[DE84] Dobkin, D.P. and Edelsbrunner. H.E., Space searching for intersecting objects, IEEE Symposium on foundations of computer science, 387-392, 1984, *J. of Algorithms*, to appear.

[DEY84] Dobkin, D.P., Edelsbrunner. H.E., and Yao, F.F., A 3-space partition and its applications, unpublished manuscript.

[DK83] Dobkin, D.P. and Kirkpatrick, D.G., Fast detection of polyhedral intersections, *Theoretical Computer Science*, 27, 241-253, 1983.

[DK85] Dobkin, D.P. and Kirkpatrick, D.G., A linear algorithm for determining the separation of convex polyhedra, *J. of Algorithms*, 6, 381-392, 1985.

[DL87] Dobkin, D.P. and Laszlo, M.J., Primitives for the manipulation of three-dimensional subdivisions, ACM symposium on computational geometry, 1987, to appear.

[DL74] Dobkin, D.P. and Lipton, R. J. , On some generalizations of binary search, Proceedings of the 6th ACM Symposium on the Theory of Computing, 310-316, 1974.

[DL76] Dobkin, D.P. and Lipton, R. J. , Multidimensional searching, *SIAM Journal on Computing*, 5, 181-186, 1976.

[DL79] Dobkin, D.P. and Lipton, R.J., On the complexity of computations under varying sets of primitives, *JCSS*, 18,1(1979), 86-91.

[DR80] Dobkin, D. P. and Reiss, S. P., On the complexity of linear programming, *Theoretical Computer Science*, 11,3, 1-18, 1980.

[DS87] Dobkin, D.P. and Souvaine, D., Computational geometry -- a user's guide, in *Advances in Robotics, Vol. 1.* edited by J. T. Schwartz and C. K. Yap, Lawrence Erlbaum Associates, 1987.

[DSV86] Dobkin, D.P., Souvaine, D., and Van Wyk, C., Decomposition and intersection of simple splinegons, submitted for publication

[Dy84] Dyer, M., Linear time algorithms for two- and three- variable linear programs, *SIAM Journal on Computing*, 13, 31-45, 1984.

[Ed77] Eddy, W. A new convex hull algorithm for planar sets, *ACM Transactions on Mathematical Software*, 3,4, 398-403, 1977.

[EGS87] Edelsbrunner, H.E., Guibas, L.J. and Stolfi, J. , Optimal point location in a monotone subdivision, *SIAM J on Computing*, to appear.

[EMK82] Edelsbrunner, H., Maurer, H. and Kirkpatrick, D., Polygonal intersection searching, *Information Processing Letters*, 14, 74-79, 1982.

[EW82] Edelsbrunner, H. and Welzl, E., Halfplane range search in linear space and $O(n^{0.695})$ query time, unpublished manuscript.

[Er46] Erdos, P., On sets of distances of n points, *American Mathematical Monthly*, 53, 248-250, 1946.

[FP81] Faux, I.E. and Pratt, M.J., *Computational geometry for design and manufacture*, Ellis Horwood Ltd., Chichester, 1981

[FB74] Finkel, R.A. and Bentley, J.L., Quad trees: a data structure for retrieval on composite keys, *Acta Informatica*, 4, 1-9, 1974.

[Fl76] Floyd, R., private communication.

[Fo68] Forrest, A.R., Curves and surfaces for computer-aided design, PhD thesis, Cambridge University, July, 1968.

[Fo86] Fortune, S., A sweepline algorithm for Voronoi diagrams, ACM symposium on computational geometry, 313-322, 1986.

[Fr81] Fredman, M.L., Lower bounds on the complexity of some optimal data structures, *SIAM J on Computing*, 10, 1-10, 1981.

[Gr71] Graham, R.L., An efficient algorithm for determining the convex hull of a finite planar set, *Information Processing Letters*, 1, 132-133, 1972.

[Gr83] Graham, R.L. and Yao, F.F., Finding the convex hull of a simple polygon, *J. of Algorithms*, 4, 324-331, 1983.

[Gr56] Grunbaum, B., A proof of Vaszonyi's conjecture, *Bulletin Res. Council of Israel*, 6, 77-78, 1956.

[GS85] Guibas, L.J. and Stolfi, J. , Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Trans. on Graphics*, 2, 75-123, 1985.

[GS87] Guibas, L.J. and Sharir, M., private communication.

[HW86] Haussler, D. and Welzl, E., Epsilon-nets and simplex range queries, ACM symposium on computational geometry, 1986, 61-71, *Discrete and Computational Geometry*, to appear.

[HZ82] Hubschman, H. and Zucker, S., Frame-to-frame coherence and the hidden surface computation: constraints for a convex world, *ACM Transactions on Graphics*, 1,2, April, 1982, 129-162.

[Hu78] Hunter, G.M., Efficient computation and data structures for graphics, PhD thesis, Princeton University, 1978.

[HS79] Hunter, G.M. and Steiglitz, K., Operations on images using quad trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1, 145-153, 1979.

[JB87] Jameson, A. and Baker, T., Improvements to the aircraft Euler method, AIAA 25th Aerospace sciences meeting, paper AIAA-87-0452, 1987.

[KR84] Kalin, R. and Ramsey, N., A new analysis and comparison of algorithms for the planar segment intersection problem, unpublished manuscript.

[Ki83] Kirkpatrick, D.G., Optimal search in planar subdivisions, *SIAM J on Computing*, 12, 28-35, 1983.

[KS86] Kirkpatrick, D.G. and Seidel, R., The ultimate convex hull algorithm?, *SIAM J on Computing*, 15, 287-299, 1986.

[Kn73] Knuth, D.E., *The art of computer programming, Vol. 3 -- Sorting and Searching*, Addison-Wesley, Reading, MA, 1973.

[La87] Laszlo, M.J., Manipulating three-dimensional subdivisions, PhD thesis, Princeton University, 1987, to appear.

[Le83] Lee, D.T., On finding the convex hull of a simple polygon, *International Journal of Computing and Information Sciences*, 12, 87-98, 1983.

[LP79] Lee, D. T. and Preparata, F.P., Location of a point in a planar subdivision and its applications, *SIAM Journal on Computing*, 6, 3, 594-606, 1979.

[LW77] Lee, D.T. and Wong, C.K., Worst-case analysis for region and partial region searches in multidimensional binary trees and balanced quad trees, *Acta Informatica*, 9, 23-29, 1977.

[LT77a] Lipton, R.J. and Tarjan, R.E., A separator theorem for planar graphs, Conference on theoretical computer science, 1-10, 1977.

[LT77b] Lipton, R.J. and Tarjan, R.E., Applications of a planar separator theorem, IEEE Symposium on foundations of computer science, 162-170, 1977.

[MS82] Mairson, H. G. and Stolfi, J., unpublished manuscript.

[Me83] Megiddo, N., Linear time algorithm for linear programming in R^3 and related problems, *SIAM J on Computing*, 12, 759-776, 1983.

[Me84] Megiddo, N., Linear programming in linear time when the dimension is fixed, *JACM*, 31, 114-127, 1984.

[MP78] Muller, D.E. and Preparata, F.P., Finding the intersection of two convex polyhedra, *Theoretical Computer Science*, 7,217-236, 1978.

[My85] Myers, E. W., An O(E log E + I) expected time algorithm for the planar segment intersection problem, *SIAM J on Computing*, 14, 625-637, 1985.

[Pa87] Panduranga, E.S., Reflections in curved surfaces, PhD Thesis, Princeton U., to appear.

[PH77] Preparata, F.P. and Hong, S. J., Convex hulls of finite sets of points in two and three dimensions, *Communications of the ACM*, 20, 2, 87-93, 1977.

[PS85] Preparata, F.P. and Shamos, M.I., *Computational Geometry* Springer-Verlag, 1985.

[Re77] Reiss, S.P., private communication.

[Ro85] Rosenstiehl, P., private communication.

[Sa84] Samet, H., The quadtree and related hierarchical data structures, *ACM Computing Surveys*, 16, 187-260, 1984.

[ST86] Sarnak, N. and Tarjan, R.E., Planar point location using persistent search trees, *Communications of the ACM*, 29, 669-679, 1986.

[Se81] Seidel, R., A convex hull algorithm optimal for point sets in even dimensions, Technical Report, University of British Columbia, 1981.

[Se86] Seidel, R., Output-size sensitive algorithms for constructive problems in computational geometry, PhD thesis, Cornell University, 1986.

[Sh74] Shamos, M.I., private communication.

[Sh75] Shamos, M.I., Geometric complexity, Proceedings of the 7th ACM Symposium on the Theory of Computing, 224-233, 1975.

[SH75] Shamos, M.I. and Hoey, D., Closest point problems, IEEE Symposium on foundations of computer science, 151-162, 1975.

[SH76] Shamos, M.I. and Hoey, D., Geometric intersection problems, IEEE Symposium on foundations of computer science, 208-215, 1976.

[Sw85] Swart, G., Finding the convex hull facet by facet, *J. of Algorithms*, 6, 17-48, 1985.

[Wa69] Warnock, J. E., A hidden-surface algorithm for computer generated half-tone pictures, University of Utah Computer Science Department, TR 4-15, 1969.

[Wa70] Watkins, G. S., A real-time visible surface algorithm, University of Utah Computer Science Department, UTEC-CSc-70-101, June, 1970.

[Wi82] Willard, D., Polygon retrieval, *SIAM Journal on Computing*, 11, 149-165, 1982.

[YY85] Yao, A. and Yao, F., A general approach to d-dimensional geometric queries, Proceedings of the 17th ACM Symposium on the Theory of Computing, 163-169, 1985.