GEOMETRIC COMPLEXITY

Michael Ian Shamos *
Department of Computer Science
Yale University
New Haven, Connecticut 06520

Abstract

The complexity of a number of fundamental problems in computational geometry is examined and a number of new fast algorithms are presented and analyzed. General methods for obtaining results in geometric complexity are given and upper and lower bounds are obtained for problems involving sets of points, lines, and polygons in the plane. An effort is made to recast classical theorems into a useful computational form and analogies are developed between constructibility questions in Euclidean geometry and computability questions in modern computational complexity.

I. Introduction

Geometric problems arise in a wide variety of application areas, from pattern recognition to operations research and numerical analysis. Many of these are simply-stated questions involving points, lines, and polygons, but their computational complexity has never been systematically investigated. It is our purpose here to develop a number of geometric algorithms, give bounds on their space and time requirements, and exhibit a close connection between geometric questions and such well-understood algorithms as sorting and searching.

We shall use three major approaches to obtain geometric results : first, analytic geometry provides the crucial link that enables algebraic complexity theory to be brought directly to bear on geometric problems. Second, it is often possible to map a geometric problem into a combinatorial problem of known complexity, such as sorting. Third, geometric properties of the problem, such as convexity or a convenient metric, can often be exploited to yield fast algorithms. We give examples involving the use of each of these methods.

* This research was supported in part by the Office of Naval Research under Grant Number NR044-483 and my wife.

Unfortunately, the classical mathematics of geometry is not well-suited to the development of good computational techniques, largely because there was no need for fast algorithms during the period when geometry flourished. It is our plan to recast geometry into a computational setting, prove theorems that are of direct algorithmic value, and investigate the complexity of basic geometric problems.

An example of the unsuitability of traditional methods is provided by the problem of separability. Two finite plane point sets, P and Q are said to be (linearly) separable iff there exists a straight line ℓ with the property that every point of P lies on one side of ℓ and every point of Q lies on the other.

Problem 1. *Given two finite plane sets, each containing* n *points, determine if they are separable.*

This problem is of importance in pattern recognition and clustering. The theorem of combinatorial geometry that pertains to separability is that of Kirchberger [1].

Theorem 1. Two finite plane sets P and Q are separable iff every subset of four or fewer points of $P \cup Q$ is separable.

Since there are $O(n^4)$ such subsets, the theorem suggests an $O(n^4)$ algorithm for separability, namely, one that examines all subsets. A much more efficient (in fact, optimal) algorithm can be obtained from the following, more computationally oriented, theorem.

Theorem 2. Two finite plane sets are separable iff their convex hulls are disjoint.
(see figure 1.) As we shall see later, an O(n log n) algorithm is possible.

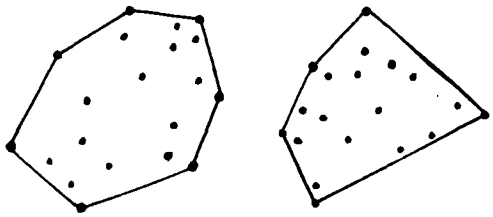


Figure 1. Separable sets and their convex hulls.

II. Algebraic methods

Since many geometry problems can be conveniently expressed in algebraic terms, the theory of algebraic complexity can often be borrowed directly. An example of this approach is the problem of finding the area of plane polygon with n vertices whose coordinates are (x_i,y_i) , $i = 0, \ldots, n-1$. The area of the polygon is then given by

$$\frac{1}{2} \left| \sum_{i=0}^{n-1} x_i(y_{i+1} - y_{i-1}) \right| \qquad [2]$$

where indices are taken modulo n. If this expression is evaluated explicitly as written, n multiplications and 2n-1 addition/subtractions are required, not counting the multiplication by one-half. We certainly would expect the number of multiplications required to compute the area of an n-gon to increase strictly monotonically with n. It is surprising that this is not the case and the author [3] has proven the following

Theorem 3. The area of a plane polygon with n vertices can be found in n-1 multiplications if n is odd and n-2 multiplications if n is even, and these are lower bounds.

Proof. The above summation can be written as

$$\frac{1}{2} \left| \sum_{i=0}^{n-2} (x_i - x_{n-1})(y_{i+1} - y_{i-1}) \right| \quad , \text{ n odd}$$

$$\frac{1}{2} \left| \sum_{i=1}^{\frac{n}{2}-1} (x_{2i} - x_0)(y_{2i+1} - y_{2i-1}) + (x_{2i-1} - x_{n-1})(y_{2i} - y_{2i-2}) \right|$$

n even, from which the theorem follows explicitly. The lower bound is proved in [3]. Using the above formulas it is possible to compute the area of a triangle in two multiplications and five addition/subtractions. It is also possible to find the area of a quadrilateral in two multiplications and five addition/subtractions !

III. Representation issues

Before going on to more complicated problems, a word is in order concerning our model of computation. Formally, we will prove lower bounds by showing that a reducibility exists between the given problem and sorting. Thus the lower bound of O(n log n) will apply to any model in which sorting requires this much time. Models which admit square roots and transcendental functions are not known to obey this criterion. Informally, however, we assume that the underlying machine is a random-access computer, and all algorithms presented will be readily seen to be programmable on such a machine. It follows that all upper bounds in this paper are achievable to within a constant factor.

Geometric objects will be represented as lists of points, where a point is a vector of coordinates in any coordinate system whatever. Since a point can be transformed into an arbitrary coordinate system in constant time, our asymptotic bounds are unaffected by choice of reference frame. A polygon is a list of points in the order in which they occur on the boundary of the polygon. All polygons are assumed to be simple, that is, they do not intersect themselves. Convex and star-shaped polygons are simple a fortiori. In many algorithms it is convenient to choose the ordering of vertices around a polygon so that, as vertices are scanned sequentially, the interior of the polygon is to the right of the induced direction.

IV. Geometric reducibility

Not all problems profit from being translated into algebraic terms as did the area problem. An example is Problem 2. *Given a set of* n *points in the plane, determine its convex hull.* It is extremely cumbersome to write down an analytic expression for the vertices of the convex hull in terms of the given original coordinates. Even if one were to do so, he would probably obtain no insight into the complexity of the problem. Graham [4] has given an O(n log n) algorithm for finding the planar convex hull. We will show that this is also a lower bound.

Theorem 4. Let H(n) be the time required to find the convex hull of n points and let S(n) be the time required to sort n real numbers. Then we have

$$H(n) \geq S(n) - O(n) .$$

Proof. We show that any convex hull algorithm can sort. Given n real numbers $\{x_1, \ldots, x_n\}$, let $m = \min \{x_i\}$ and $r = \max \{|x_i - x_j|\}$. Thus m is the smallest number and r is the range of the numbers. Choose some number α in the interval $(0,2\pi)$ and define $\theta_i = \alpha(x_i - m)/r$. Note that all θ_i lie in the semi-open interval $[0,2\pi)$. We now associate with each x_i a point on the unit circle having polar coordinates $(1,\theta_i)$. The convex hull of these can be found in H(n) time. The vertices of the resulting convex polygon, taken in order, constitute a sort of the θ_i . In O(n) time these can be mapped back to the corresponding sorted x_i. So we have that $S(n) \leq H(n) + O(n)$, from which the theorem follows. [5] □

In this case, where formulating an algebraic solution was not inviting, we were able to bypass algebraic methods by exhibiting an equivalence between the problem at hand and another problem of known complexity.

V. Geometric algorithms

It is our goal in this section to present a number of fast algorithms which can be used as "core" algorithms for obtaining efficient solutions to more complex problems.

Problem 3. *Given a convex polygon, determine the maximum distance between two of its points.* This distance is called the diameter of the polygon. It is elementary that the diameter is realized by two vertices. A naive algorithm is to examine all n(n-1)/2 interpoint distances, finding the greatest in $O(n^2)$ time. By creating a new geometric structure and applying classical theorems we can obtain a linear algorithm.

A line of support of a convex polygon P is a line that has at least one point in common with P but with the property that all of P lies on one side of the line. The diameter of P is the greatest distance between parallel lines of support. [6] A pair of vertices v,w of P will be called antipodal if there exist parallel lines of support of P passing through v and w. It is clear that only distances between antipodal pairs of points need be examined in order to determine the diameter. How many antipodal pairs of vertices can P possess ? We will exhibit a data structure which permits not only the rapid enumeration of all antipodal pairs, but also the determination of the pair of vertices through which lies of support parallel to a given direction pass.

The cyclic ordering of the vertices of P induces a direction on each edge of P. Treating the edges as vectors, translate them to the origin. (figure 2) In this mapping, edges go to vectors and vertices go to sectors. In order to find the antipodal pair corresponding to some direction determined by a line ℓ, translate the line so that it passes through the origin of the vector diagram. The sectors through which it passes indicate the points of the antipodal pair. In the example, the dotted lines passes through sectors one and four.

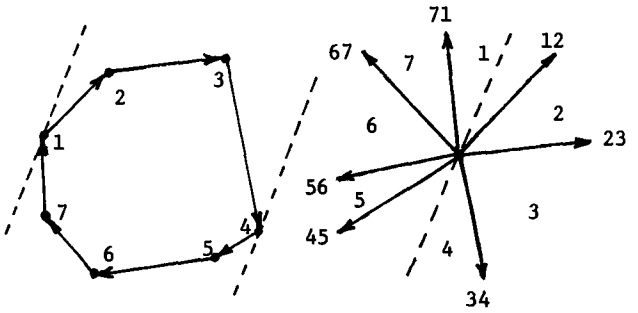


Figure 2. Determining antipodal points.

Determining the sectors through which ℓ passes can be done in O(log n) time by binary search. To find all antipodal pairs, imagine rotating line ℓ clockwise. The antipodal pair does not change until ℓ passes through some vector of the diagram. In figure 2, pair (1,4) turns into pair (1,5) as ℓ passes through vector $\overline{45}$. Since there are exactly n vectors to pass, there are exactly n antipodal pairs. Furthermore, they can all be found in O(n) time by scanning sequentially around the vector diagram. Since the distance between any two points can be found in constant time, we have

Theorem 5. The diameter of a convex polygon can be determined in O(n) time. □

Although it is possible for ℓ to pass through two vectors simultaneously, this complication does not affect the result.

Theorem 6. The diameter of a set of n points in the plane can be found in O(n log n) time.

Proof : diam(S) = diam(hull(S)). The hull can be found in O(n log n) time. Since the hull is a convex

polygon, its diameter can be found in an additional O(n) time by theorem 5. □

The optimality of the above algorithm is an open question.

Theorem 7. The diameter of a polygon (not necessarily convex) can be found in O(n) time.

Proof : The convex hull of a simple plane polygon can be found in O(n) time [7] and then theorem 5 applies. □

Theorem 8. Finding a simple closed polygonal path through n points of the plane must take O(n log n) time in the worst case.

Proof a : A simple closed polygonal path (SCPP) is a polygon. Since the hull of a polygon can be found in O(n) time, if any SCPP could be found in less than O(n log n) time we could find convex hulls in less than O(n log n) time, contradicting theorem four. □

Proof b : Consider a set consisting of n-1 points on the x-axis and another point not on the x-axis. Any SCPP effectively sorts the points on x-coordinate. cf.[8] □

Since a Euclidean traveling salesman tour is an SCPP, theorem 8 gives a non-linear lower bound for the traveling salesman problem.

Theorem 9. The L^1 and L^∞ diameter of a finite plane set can be found in O(n) time. See [7]. □

The L^1 distance, also called the rectilinear or "Manhattan" distance between points p_1 and p_2 is given by $d^1(p_1,p_2) = |x_1 - x_2| + |y_1 - y_2|$. The distance in the infinity metric is just $d^\infty(p_1,p_2) = \max(|x_1 - x_2|, |y_1 - y_2|)$. An O(n) algorithm obtains under any metric whose unit ball is a convex polygon.

Problem 4. *Given a convex polygon* P, *preprocessing allowed, determine whether a new point* x *is interior or exterior to* P.

This question can be decided in O(n) time by examining the intersection of the boundary of P with any line through x. If there are two intersection points which bracket point x, then x is interior, otherwise x is exterior. But this algorithm does not take advantage of preprocessing.

Theorem 10. The inclusion question for a convex polygon can be answered in O(log n) time, after O(n) preprocessing.

Proof : Choose any point r interior to P (the centroid of any three non-collinear points will suffice) and consider r to be the origin of polar coordinates. Rays drawn through the vertices of P originating at r partition the plane into n sectors. Each edge of P divides a sector into two regions, one interior to P and the other exterior. (see figure 3) Given the new point x, its polar coordinates relative to r can be found in constant time. The sector containing x can be found in O(log n) time by binary search. Once the sector is determined, whether x is inside or outside can be found by testing x against the edge of P contained in the sector. The preprocessing consists of arranging the sectors, which can be done in O(n) time since the vertices of P are available in order by angle. □

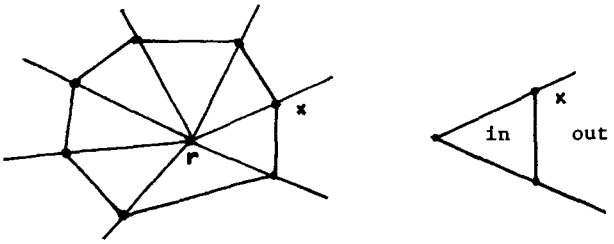


Figure 3. Inclusion in a convex polygon.

Problem 5. *Given two convex n-gons,* A *and* B, *find their intersection.*

The intersection is a convex polygon having at most 2n vertices. An $O(n^2)$ algorithm comes to mind immediately : check every edge of A against every edge of B, looking for intersections. While this algorithm is optimal for non-convex polygons, the convex case is restricted enough to permit a fast algorithm.

1. Preprocess polygon B as for inclusion testing (theorem 10). Let b be the origin for the sectors.
2. If b is exterior to polygon A, go to step 6.
3. Refer to figure 4. For each vertex of A, determine which sector of B it lies in. Although the first such search may cost O(log n), all n vertices can be located in a total of O(n) time by proceeding sequentially around polygon A. The scan never backs up, so no sector of B is examined more times than once plus the number of vertices of A that lie within it. Hence this step requires only O(n) time.
4. Once it is known which sector a vertex lies in, it can be determined in constant time (as in theorem 10) whether the vertex is interior or exterior to B. Scan around A once, examining all pairs of consecutive vertices a_i , a_{i+1} . If both vertices are in the same sector and interior to B,

then by convexity the edge joining them is also interior to B and it cannot intersect B. If one vertex is inside and the other is outside, then exactly one intersection occurs between the edge $a_i a_{i+1}$ and the bounding edge of the sector. If both vertices are outside B but in the same sector, no intersection occurs. The situation is the same even if a_i and a_{i+1} lie in different sectors of B, except in the case where both are outside. It is then necessary to check $a_i a_{i+1}$ against the edges bounding all intervening sectors. Since no backtracking is done, this step can be performed in O(n) time. (Separate treatment is required if a point of A lies on an edge of B, but the substance of the algorithm is not affected.)

5. The intersection consists of chains taken alternately from polygons A and B, with the intersection points in between. STOP.
6. If b is exterior to polygon A then the search in step three must be modified. At b, polygon A subtends some angle α in which all relevant sectors fall. These may be determined by finding the range of polar angles of the vertices of A. Let θ_i be the polar angle at b of vertex a_i. Then let $u = \min \{\theta_i\}$ and $v = \max \{\theta_i\}$. Now, vertices u and v partition A into two chains of vertices, each of which may be searched as in step three, separately.

We have proved

Theorem 11. The intersection of two convex n-gons can be found in O(n) time. □

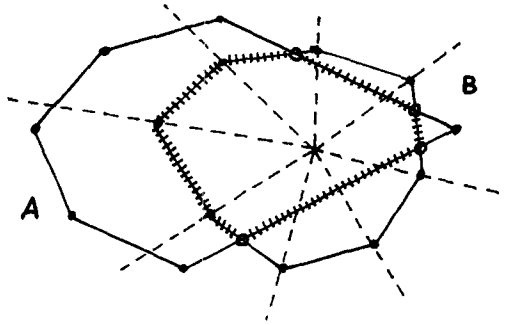


Figure 4. Intersection of convex polygons.

If the polygons are not convex, their intersection may not even be connected.

Theorem 12. Finding the intersection of two n-gons requires $O(n^2)$ time in the worst case.

Proof : Figure 5 shows two n-gons in which every edge of one intersects every edge of the other. The intersection consists of $n^2/4$ disjoint quadrilaterals which have a total of n^2 edges. Merely writing out the answer requires $O(n^2)$ time. The obvious algorithm discussed in problem 5 shows that this is also an upper bound. □

The convex intersection algorithm is a useful building block for more complicated algorithms. In the following discussion we consider star-shaped polygons.

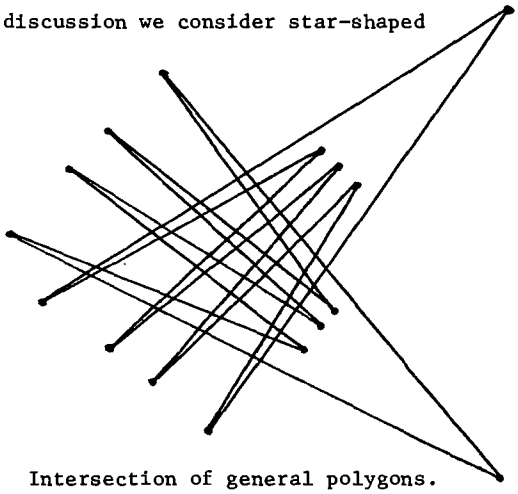


Figure 5. Intersection of general polygons.

Given a polygon P, the set of points which can "see" all points of P is called the kernel of P. More precisely, $\ker(P) = \{x \in P \mid \forall y \in P, \overline{xy} \in P\}$, where $\overline{xy}$ denotes the line segment from x to y. A polygon whose kernel is non-null is said to be star-shaped. The kernel is a convex polygon having no more than n edges.

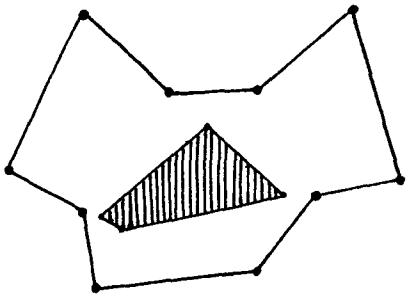


Figure 6. A polygon and its kernel.

Problem 6. *Given a polygon, find its kernel.*

Each side of P determines two half-planes. The one to the right (in the directed sense of P) is said to be the interior half-plane, owing to our earlier definition of a polygon. The kernel of P is just the intersection of all its interior half-planes. This shows that the kernel is convex.

Theorem 13. The intersection of n half-planes, hence the kernel of an n-gon, can be found in O(n log n) time.

Proof : Begin by intersecting the n half-planes

in pairs. This can be done in time cn/2, for some constant c. The result is at worst n/2 angles, or "2-gons." These can be intersected in pairs in time 2cn/4 = cn/2 to form n/4 quadrilaterals, and so forth. This process continues for a maximum of log n steps. That each step only requires O(n) time follows from theorem 11. Thus the entire algorithm can be performed in O(n log n) time. [9] □
This algorithm has not been shown to be optimal.

Using the above result and the method of theorem 10 we can obtain an O(log n) algorithm for inclusion in a star-shaped polygon. It suffices to choose r to be inside the kernel of the polygon. O(n) storage and O(n log n) preprocessing time are required.

Theorem 14. Whether two plane sets of n points are separable can be determined in O(n log n) time.

Proof : Applying theorem 2, the convex hulls of the sets can be found in O(n log n) time [4]. By theorem 11, the intersection of the hulls can be found in an additional O(n) time. If the intersection is null, the sets are separable. □

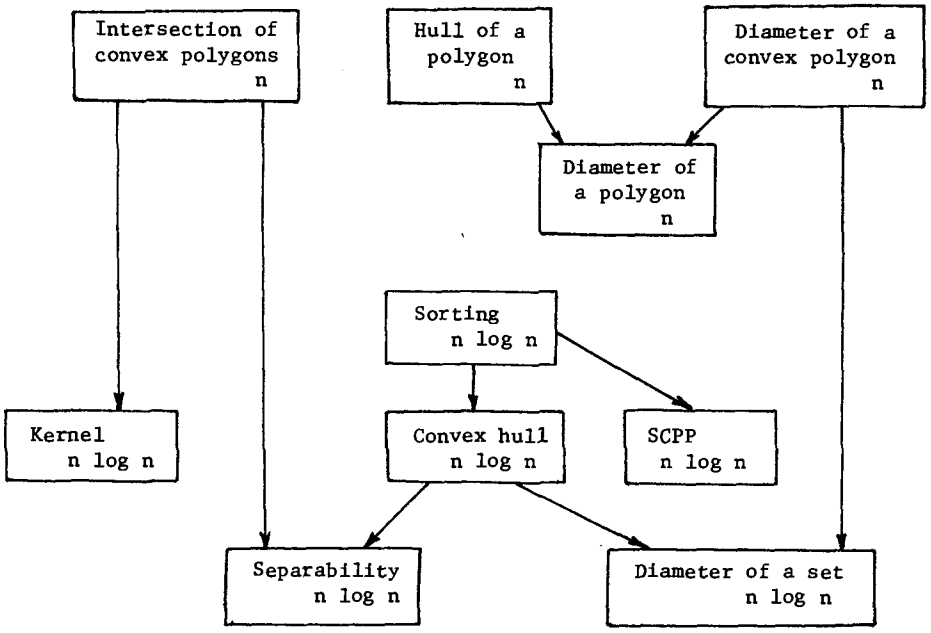
That the above algorithms are closely related can be seen from the following chart in which there is an arrow from algorithm A to algorithm B if A is used by B.

VI. Closest-point problems.

Problem 7. *Given* n *points in the plane, with preprocessing allowed, how quickly can the point closest to a new given point be found ?*
This problem is mentioned by Knuth [10] under the name "post-office search," but no solution is given. We will develop a data structure that solves this problem and a number of related ones.

Surrounding each of the original points p_i there is a convex polygon V_i, called the Voronoi polygon associated with p_i that has the following property : p_i is the closest of the given points to any $x \in V_i$.
The Voronoi polygon surrounding p_i is composed of pieces of the perpendicular bisectors of p_i and the other given points. If $h(p_i,p_j)$ denotes the half-plane containing p_i determined by the perpendicular bisector of p_i and p_j, then $V_i = \bigcap_{i \neq j} h(p_i,p_j)$,
which shows that V_i is a convex polygon having at most n-1 edges. The Voronoi polygons partition the plane, with semi-infinite polygons corresponding to points on the convex hull of the given set. Refer to figure seven. To solve the closest-point problem it is only necessary to determine in which Voronoi polygon the new point lies.

INTERPLAY OF GEOMETRIC ALGORITHMS



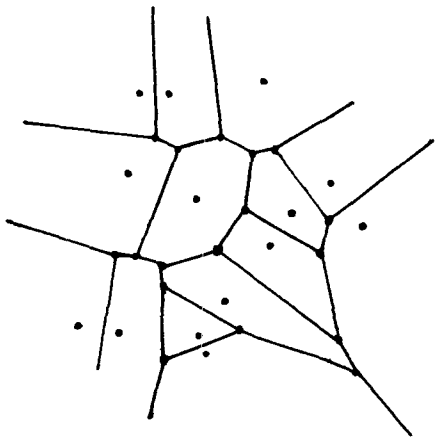


Figure 7. Voronoi polygons.

The Voronoi diagram has many interesting and useful properties, which are treated in detail elsewhere [12]. We mention only those features relevant to the closest-point problem.

Theorem 15. The Voronoi diagram V(S) of a set S containing n points has at most 3n-6 edges and 2n-4 vertices.

Proof : Consider the n given points as vertices of a graph in which there is an edge from v_i to v_j iff V_i and V_j share a common edge. This graph, D(S), is the geometric dual of V(S). Take as the

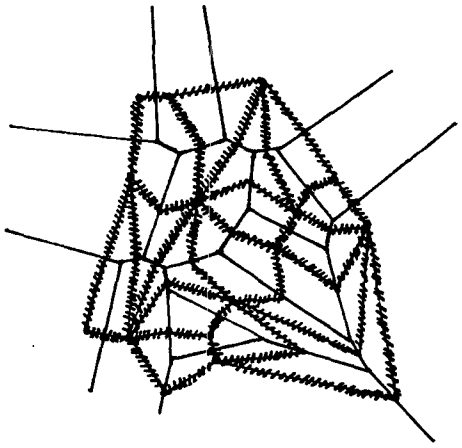


Figure 8. Dual of the Voronoi diagram.

edge v_iv_j a broken line segment from p_i to the midpoint of the common edge, then to p_j. By the convexity of the Voronoi polygons, none of these edges intersect, thus D(S) is planar. D(S), being a planar graph on n vertices, has at most 3n-6 edges. Since the edges of D(S) and V(S) are in one-to-one correspondence, V(S) has at most 3n-6 edges.

The vertices of V(S) are points at which three perpendicular bisectors meet; they are the circumcenters of triangles. So the degree of every vertex of V(S) is three. (If d>3 bisectors meet at a point, that point has multiplicity d-2.) Let V(S) have k vertices. Then $3k \leq 2(3n-6)$, or $k \leq 2n-4$. □

Theorem 16. The closest point problem can be solved in O(log n) time and $O(n^2)$ storage after $O(n^2)$ preprocessing.

Proof : We will give an algorithm. Consider drawing a horizontal line through each Voronoi point. These lines partition the plane into slabs. If the Voronoi points are pre-sorted by y-coordinate, then the slab containing a new given point x can be found in O(log n) time by binary search. There are at most 2n-3 slabs. The situation within each slab is very attractive. The line segments occurring within a slab do not intersect (except possibly at slab boundaries) and are thus totally ordered by the "right-left" relation. Areas between slab segments belong wholly to one Voronoi polygon. (see figures 9 and 10)

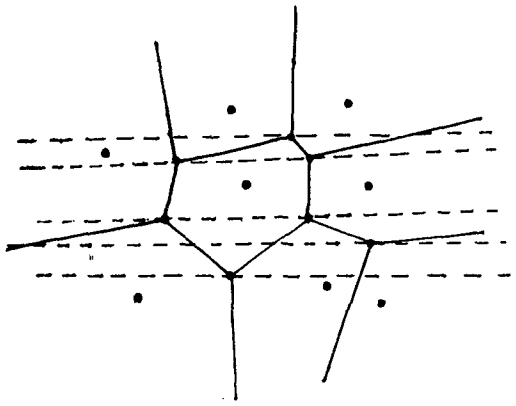


Figure 9. Voronoi polygons and slabs.

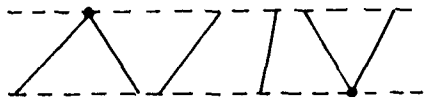


Figure 10. A single Voronoi slab.

Inside a slab there can be at most O(n) segments, since there are only that many in the whole diagram ! The polygon to which a point belongs can be found in O(log n) time by binary search, which solves the problem. Since there are O(n) slabs and no more than O(n) segments in any slab, $O(n^2)$ storage suffices. [13] □

The upper bound on Voronoi preprocessing follows from

Theorem 17. The Voronoi diagram V(S) can be constructed in O(n log n) time, and this is optimal.

Proof : see [12]. A divide-and-conquer approach is used based on the fact that two Voronoi diagrams for separable sets each having n/2 points can be merged in O(n) time to form the complete Voronoi diagram. Recursive application of this principle yields an O(n log n) algorithm. □

In many contexts, $O(n^2)$ storage is too expensive and we would be willing to settle for a slightly slower algorithm if it could reduce the storage requirement drastically. The basic idea of the next algorithm is to spend O(log n) time to reduce the problem to one that is not larger than half the size of the original.

Theorem 18. The closest-point problem can be solved in $O(\log^2 n)$ time and O(n) storage after O(n log n) preprocessing.

Sketch of proof : Suppose we have sorted the given points so that they are ordered by x-coordinate. We seek a decision boundary that separates the leftmost n/2 points from the rightmost n/2 such that if a point x is to the left of the boundary then it is closest to one of the leftmost n/2 points. If this test can be made in O(log n) time, the problem can be solved recursively. The decision boundary is easy to construct, given the Voronoi diagram. Color the Voronoi polygons of points in the left set blue, of those in the right set yellow. At the edges separating the sets, the paint will run together, forming a green decision boundary. This boundary is single-valued in y, so it may be searched by the slab method in O(log n) time. At the next level of the algorithm there will be <u>two</u> decision boundaries, then four, etc. Construction of the boundaries at successive levels proceeds as at the first level except that no edge of the Voronoi diagram is ever used twice. Once the given point has been compared to an edge, it is never necessary to test against that edge again. Complications arise since the decision boundaries are not required to be connected, and testing against a boundary with k pieces may achieve a (k+1)-fold splitting. A Voronoi polygon, hence a point, becomes a terminal node of the search tree when all of its edges have appeared as parts of decision boudaries. The depth of the tree is at most $\log_2 n$, so $O(\log^2 n)$ time suffices for the entire algorithm. Since no edge of the Voronoi diagram appears more than once in the search structure, only O(n) storage is required. The preprocessing upper and lower bound is a corollary of theorem 17. [14] □

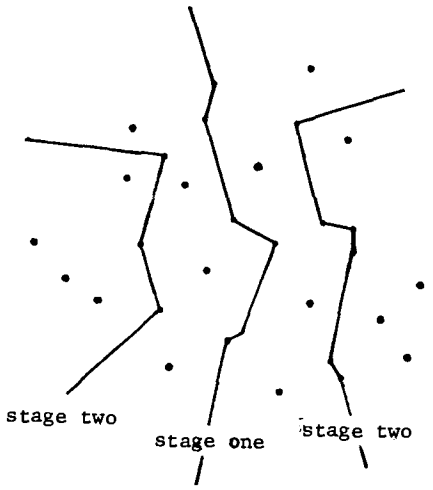


Figure 11. The first and second stages of division.

The Voronoi diagram is a very powerful structure. It can be used to find the two closest of n points in O(n log n) time and to find a Euclidean minimum spanning tree in O(n log n) time. This follows from the fact that an MST of the set S is an MST of D(S), the dual of the Voronoi diagram. [12] This result is attractive because it says that a minimum spanning tree can be constructed without even examining all the edges of the underlying graph. The diagram also solves

Problem 8. *Given* n *points in the plane, find a new point* x, *interior to the convex hull of the originals, whose smallest distance to any point is a maximum.*

The best previously known algorithm for this problem required $O(n^3)$ time [15] but the Voronoi diagram leads to an O(n log n) solution, since the required point x is one of the vertices of the diagram. There are only O(n) vertices to examine, and they can all be generated in O(n log n) time.

These fast algorithms demonstrate that the proper attack on a geometry problem is to construct those geometric entities that delineate the problem, such as the vector diagram of theorem 5, or the Voronoi polygons, and order these for rapid searching. In many cases such well-known algorithmic techniques as divide-and-conquer can be applied directly.

VII. Parallels between Geometry and Complexity

The connection between geometry and the theory of computation is not limited to the design of fast algorithms. Many questions arose during the development of classical geometry that are inherently complexity questions which the mathematicians of the day did not possess the formal tools to solve. In spite of this, geometry seems to have anticipated many of the ideas and investigations of modern computational complexity. By studying the work of the geometers we may discover fruitful areas for complexity research.

The geometric concept most closely related to the theory of computation is that of a Euclidean construction. Euclid realized in his constructions those algorithmic elements that we regard as indispensible today : finiteness, clarity, and termination. His constructions were always accompanied by proofs of correctness; in fact, the algorithm and its proof were often intertwined, a goal that seems even more desirable today. The question of the completeness of the Euclidean ruler and compass operations, that is, whether they suffice to perform all possible constructions, was raised by the Ancients. For centuries, considerable effort went into the problem of finding algorithms with the Euclidean tools for trisecting an angle, constructing a square equal in area to a given circle, and finding the side of a cube whose volume is twice that of a given cube. The existence of such algorithms is a computability question. An analogous (unsolved) problem in computer science is to determine whether the "operations" of a linear bounded automaton suffice to recognize all context-sensitive languages.

In 1796, Gauss proved that the Euclidean tools were not adequate to inscribe a regular p-gon in a circle for any prime p unless p is a Fermat prime (of the form $2^{2^n} + 1$). He did this by algebraic methods that make use of the fact that a ruler and compass construction is equivalent to a computation on the coordinates of the given points with a finite number of the operations $\{+, -, \times, \div, \sqrt{}\}$. With the advent of Galois theory in the early nineteeth century a complete characterization of those problems solvable with ruler and compass became available. The theory of Euclidean constructibility provides a natural framework for the extension of arithmetic complexity, which deals primarily with the field $\{+, -, \times, \div\}$, to include the square root operation. Many other models of geometric construction have been propounded, in much the same vein as the restricted or enhanced Turing machine models we now study. It was known, for example, that the ruler alone has strictly less power than the ruler and scale together (a scale is a ruler with two fixed marks), and that the ruler and scale have strictly less power than the ruler and compass. The unexpected result of Mohr and Mascheroni (before Gauss) is that, insofar as geometric objects are given and determined by points, the compass alone suffices to perform all Euclidean constructions. The compass and scale are more powerful still, since they can be used to trisect an angle. What is of interest to us about the Mohr-Mascheroni theorem is that it was proved by simulation; that is, it was demonstrated how a single compass could emulate, in a much more complicated way, any operation performed with ruler and compass [16]. The technique is reminiscent of the proof that a single-tape Turing machine can simulate a k-tape machine. Thus geometry possesses some elementary notions of hierarchy and the power of computing features.

So far we have discussed geometric computability but not complexity. While concise and elegant constructions were always regarded as desirable, no systematic study of the complexity of Euclidean constructions was undertaken until the work of Lemoine early in this century. [17] He recognizes five distinct Euclidean operations :

1. Placing a compass leg on a point.
2. Placing a compass leg arbitrarily on a line.
3. Placing the ruler edge through a point.
4. Drawing a circle.
5. Producing a line.

The total number of these operations performed during a construction is called its simplicity. Although Lemoine was able to improve greatly the simplicity of many famous constructions, the reason we do not now possess a complete theory of Euclidean complexity is that he was unable to prove any lower bounds. Apparently upper bounds were also easier to come by in those days ! Still, Lemoine's work is the only known attempt to obtain operation counts in geometry.

Hilbert, in his Foundations of Geometry [18], gives a specific criterion for an expression over $\{+, -, \times, \div, \sqrt{a^2 + b^2}\}$ to require exactly n square root operations in its evaluation. The restriction

is that each radicand be a sum of previously computed squares. Hilbert shows that the above field represents those constructions performable with ruler and scale. While the complexity of approximating $\sqrt{x}$ in terms of other arithmetics has been studied, the complexity of computations involving the radical as a given primitive operations does not appear to have been examined. Hilbert's work may prove to be a valuable starting point for such an investigation. Such a project would not be without practical application, as machines exist in which a square root can be performed as rapidly as multiplication. The obscurity of Hilbert's result is puzzling, especially in view of its apparent importance.

VIII. Summary

Geometry and complexity are complementary areas of research in the sense that the techniques and investigations of one are relevant to the other. It is clear that complexity theory provides the proper arena in which to study classical and computational geometry problems, while a review of the historical development of geometry suggests many avenues for research in complexity theory. Geometric problems may be attacked by the same set of fast algorithm techniques that have been successful on other problems in computer science and a collection of basic algorithms can be assembled, each of them optimal and well-understood, that are useful in solving more complex questions. These algorithms can be used as building blocks for the development of efficient procedures for higher-dimensional problems.

Acknowledgement

The results in this paper would never have been obtained had it not been for countless hours of discussions with David Dobkin, Stanley Eisenstat, and Daniel J. Hoey. I am grateful for their continued enthusiasm over geometric problems.

References

1. Über Tschebyschefsche Annäherungsmethoden. Math. Ann. 57 (1903) pp. 509-540.
2. R. Duda and P. Hart. Pattern Classification and Scene Analysis. Wiley, 1973. p. 378.
3. M. Shamos. On computing the area of a plane polygon. Submitted for publication.
4. R. L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. Info. Proc. Lett. 1(1972) pp.132-133.
5. The proof of this theorem was suggested by Daniel J. Hoey.
6. Yaglom and Boltyanskii. Convex Figures. Holt, Rinehart, and Winston, 1961.
7. M. Shamos. Computational Geometry. PhD. Thesis, Yale University, 1975.
8. M. Gemignani. On Finite Subsets of the Plane and Simple Closed Polygonal Paths. Math. Mag. Jan.-Feb. 1966. pp.38-41.
9. The kernel algorithm is due to Stanley C. Eisenstat.
10. D.E.Knuth. The art of computer programming. Vol. III, Sorting and searching. Addison-Wesley, 1973. p.555.
11. After G. Voronoi. Consult C.A.Rogers, Packing and Covering. Cambridge University Press, 1964.
12. M. Shamos and D. Hoey. Closest-point Problems. In preparation.
13. This is an application of a technique of Dobkin and Lipton, Sixth SIGACT Symposium.
14. This method of attack was suggested by David Dobkin.
15. B. Dasarathy and L. White. Some maximin and pattern classifier problems : theory and algorithms. Talk presented at the Computer Science Conference, February , 1975.
16. For a lucid discussion of the power of geometric construction tools, see H. Eves, A survey of geometry, Allyn and Bacon, 1972.
17. Lemoine, Géométrographie, 1907.
18. D. Hilbert. Foundations of Geometry, 1899. Edited and reprinted by Open Court, 1971.