A Whirlwind Tour of Computational Geometry

RON GRAHAM, *AT & T Bell Laboratories, Murray Hill, NJ 07974*

FRANCES YAO, *Xerox PARC, Palo Alto, CA 94304*

RON GRAHAM, well known to our readers, has won numerous distinctions in mathematics. He is a member of the National Academy of Sciences, has received the Pólya prize in combinatorics, is an editor of numerous journals, is a trustee of the AMS, etc. His research is in combinatorics, graph theory, number theory and algorithms.

FRANCES YAO is now Principal Scientist and Manager of the Theoretical Computer Science Area at Xerox Palo Alto Research Center. She received her Ph.D. in mathematics from MIT in 1973 and has taught at several universities, including Stanford (1976–1979). Her research interests span theoretical computer science, with emphasis on computational geometry.

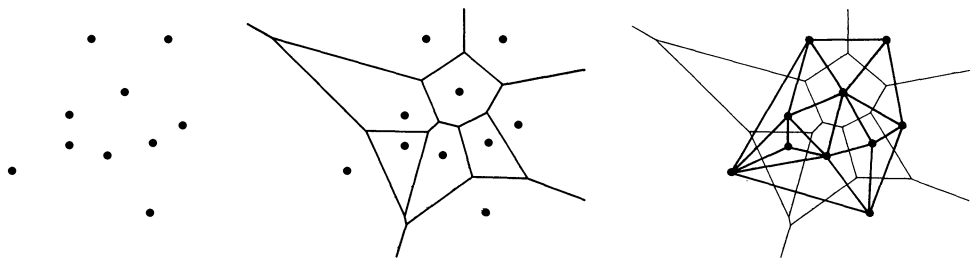
Computational geometry is concerned with finding efficient *algorithms*, or computational procedures, for solving a wide spectrum of geometric problems. Such problems can arise from computer graphics, robotics and motion planning, computer-aided design and manufacturing, or one of the many other applied areas where geometry comes into play. In spite of the practical flavor of these problems, results from classical and modern geometry quite often can play key roles in the design of efficient methods for solving them. Indeed, Euclidean geometry could be thought of as perhaps the earliest systematic study of algorithms: the ruler-and-compass constructions in Euclidean geometry are just algorithms based on a well defined set of elementary operations. In this venerable subject also occur some of the first provably unsolvable problems, such as squaring the circle, trisecting the angle and doubling the cube, much in the same spirit as the more recent negative solution of Hilbert's tenth problem (implying the nonexistence of an algorithm for solving general diophantine equations) [**9**] and the celebrated (but still unresolved) P vs. NP question [**13**].

We begin our discussion of computational geometry with several basic concepts.

Voronoi Diagrams

Consider the following problem, known as the *post office problem*: We are given a set S of n points in the plane (considered as *post offices* or *sites*). When an arbitrary new point (x, y) (say, a residence) is given, we must find out which post office is closest to (x, y).

In the example above, one expects many queries, with different values of (x, y), to be asked with respect to the same set of post offices. Thus it pays to "preprocess" the set S, that is, to have some useful computation about S done in advance, so that when a query arrives, the answer can be determined quickly. In this particular case, precomputing a map of "postal regions" would seem to be a good idea. For each post office p, the locus of points (x, y) that are closer to p than to any other post office in S is a convex region $V(p)$, called the *Voronoi region* or *Voronoi polygon* associated with p. The n polygons $V(p)$ form a partition of the plane, called the *Voronoi diagram* of S, denoted by $Vor(S)$. (FIGURE 1(b).)



(*a*) A point set S (*b*) $Vor(S)$ (*c*) $DT(S)$

FIG. 1

The earliest significant use of Voronoi regions seems to have occurred in the work of Gauss, Dirichlet and Voronoi [**29**] in their investigations on the reducibility of positive definite quadratic forms (see [**21**], [**17**] for a summary). However, the same concept has evolved independently in a variety of other scientific disciplines. For example, in crystallography, one simple model of crystal growth starts with a fixed collection of sites in 2- or 3-space, and allows crystals to begin growing from each site, spreading out at a uniform rate in all directions, until all space is filled (see [**14**]). The "crystals" (called "Wirkungsbereiche" or "domains of action") then consist of all points nearest to a particular site, and consequently are just the Voronoi regions for the original set of points (FIGURE 2).

Similarly, the statistical analysis of meteorological data led to the formulation (around 1910) of Voronoi regions under the name "Thiessen polygons." The reader is referred to [**17**] for a survey of these applications, and to [**2**] for a more complete discussion of the uses of Voronoi regions in statistics, geography, interpolation, physical chemistry, and many other fields.

The Voronoi diagram enjoys many interesting properties. First of all, the straight-line dual of the Voronoi diagram is a triangulation of S, called the *Delaunay triangulation* and denoted by $DT(S)$ (FIGURE 1(c).) It is easy to see that:

P1. If p is a nearest neighbor of q, then polygons $V(p)$ and $V(q)$ are adjacent in the Voronoi diagram. This means that p and q are connected by an edge in $DT(S)$.

P2. Points p, q, r form a triangle in $DT(S)$ if and only if the circle through p, q, and r has no point of S in its interior.

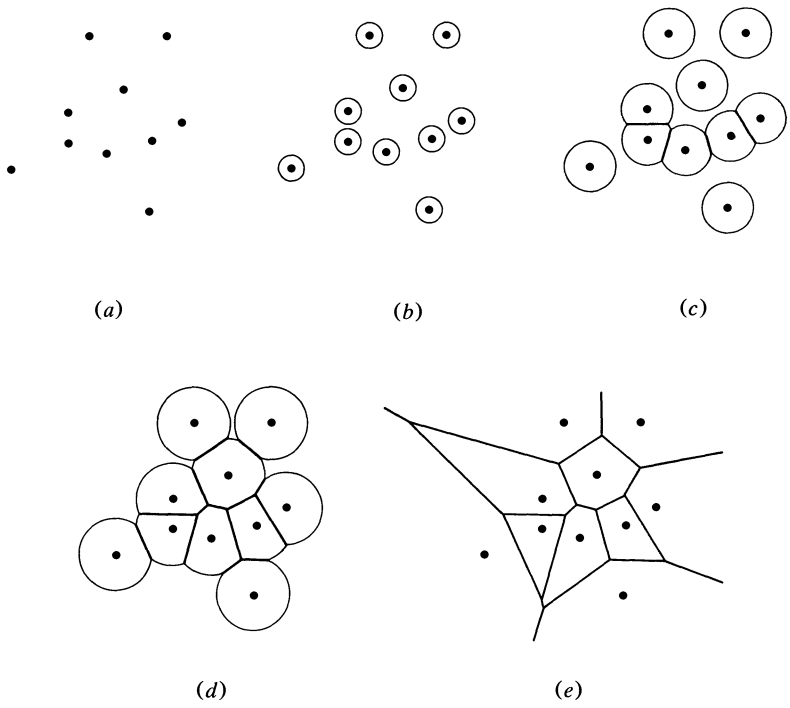


FIG. 2. Model of 2-dimensional crystal growth.

It follows from property P1 that, to find a *closest pair* of points occurring in S, we need only consider the $O(n)$ pairs which are joined by edges in the Delaunay triangulation for S.

Some further useful properties of these diagrams can be established without much difficulty. For example, the Delaunay triangulation contains all *minimum spanning trees* of the point set. A minimum spanning tree (MST) is a tree (i.e., a connected graph with no cycles) that connects all points of V, such that the sum of the edge lengths in the tree is as small as possible. The MST of a point set S may not be unique, but any such tree must use only edges present in $DT(S)$. We state this as:

P3. $MST(S) \subseteq DT(S)$.

Proof. Let $e = (p, q) \in MST(S)$. If $e \notin DT(S)$, then by property P2, the circle with (p, q) as diameter must contain some point of S, say r, in its interior. When we remove the edge e from $MST(S)$, the set S breaks up into two connected components. But the two components can be rejoined by using one of the edges (p, r) or (q, r); since either edge is shorter than the original edge (p, q), we have a contradiction. ■

Property P3 implies that we can find a MST for a set of points by first constructing its Delaunay triangulation $DT(S)$ and then, since the triangulation is a planar graph, apply a particularly efficient (e.g., linear-time) MST algorithm for planar graphs to $DT(S)$.

For a set S of n points in the plane, there may be exponentially many different triangulations. Depending on the application, one may be interested in triangula-

tions that have certain special characteristics. Not surprisingly, the Delaunay triangulation $DT(S)$ satisfies a number of extremal properties, one of which is that it maximizes the minimum angle among all triangulations of S. In other words, $DT(S)$ does the best possible job in avoiding triangles with small angles (see [11] for a proof).

Construction of Voronoi Diagrams

How difficult is it to construct the Voronoi diagram $Vor(S)$ from S? A straightforward approach is to compute the polygons $V(p)$ one by one. Write $S = \{p_1, \ldots, p_n\}$. To find $V(p_1)$, say, we can compute, for successively larger values of j, the polygon U_j which is the locus of all points that are closer to p_1 than to any point in $\{p_2, \ldots, p_j\}$. Once U_j is known, we can obtain U_{j+1} by intersecting the convex polygon U_j with the perpendicular bisector between p_1 and p_{j+1}. This step can be accomplished by a fast binary (or logarithmic) search, because the vertices of a convex polygon admit a nice ordering (first increasing and then decreasing in their x-coordinates), and a line can intersect this polygon in at most two places. Thus, the polygon $V(p_1) = U_n$ can be computed in $O(n \log n)$ steps, and therefore the whole diagram $Vor(S)$ in a total of $O(n^2 \log n)$ steps.

To compute $Vor(S)$ more efficiently, we must try to construct several polygons $V(p_i)$ simultaneously. Here we can invoke a general principle, called the *sweep technique*, which is often useful in the design of geometric algorithms. Imagine that a horizontal line is being swept up the plane from $y = -\infty$ to $y = +\infty$. Any cross-section of the two-dimensional diagram $Vor(S)$ as defined by the sweep-line is a simple one-dimensional configuration. Furthermore, the configuration undergoes essential (i.e., topological) changes rather infrequently—only when the sweep-line first enters some new region $V(p_i)$, or has just left such a region; in either case, the change is local. The difficult part, of course, is in knowing when a change should occur, since the sweep-line will enter a region $V(p_i)$ *before* it actually encounters the point p_i. It turns out that this difficulty can be overcome by a clever idea due to Fortune [12]. One first performs a non-linear transformation T of the plane, with T depending on the set S. Under this transformation, the new diagram $T(Vor(S))$ has the property that the lowest point of the Voronoi region $T(V(p_i))$ is always at the site $T(p_i)$ itself (FIGURE 3). Thus, we will create a Voronoi region only when the sweep-line reaches a new site, and delete that region when its topmost point is reached. Finally, the real Voronoi diagram can be easily reconstructed from its transformed image. The total running time of this algorithm is $O(n \log n)$, since each insertion/deletion operation can be done in $O(\log n)$ steps with a standard one-dimensional search structure.

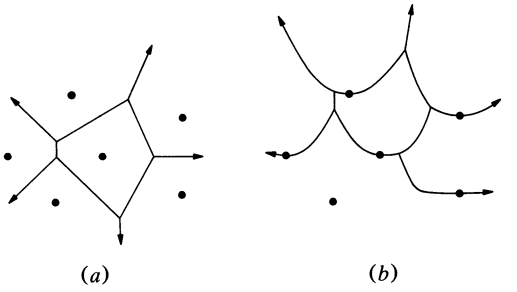


FIG. 3. Fortune's transformation.

The transformation used by Fortune maps a point r in the region $V(p)$ to $r + (0, y)$ where $y = d(r, p)$, the distance from r to p; thus, lines are transformed into hyperbolic arcs. This transformation can be intuitively understood as follows. Think of the ordinary Voronoi diagram as being generated by a physical process in which an *obstacle* is placed at each site, and an expandable disk is moved around so that at any instant, the disk has the maximum possible radius permitted by the obstacles. The locus of the center of the disk, then, defines the Voronoi diagram. If, instead of tracing the center, we trace out the locus of the *topmost* point of the disk at all times, the result would be precisely the transformed Voronoi diagram defined above.

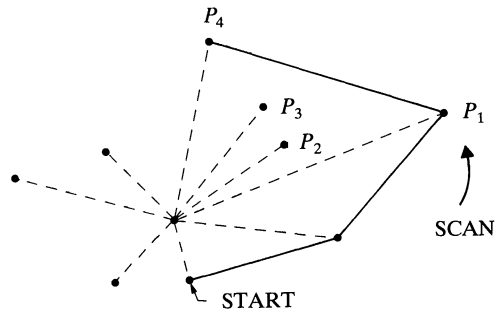
Convex Hulls

It is known that the Voronoi diagram for a finite *convex* planar set, that is, a set of n points forming the vertices of a convex polygon, can be computed in only $O(n)$ time [**1**]. Indeed, convexity is a strong geometric condition that often leads to particularly efficient solutions to computational problems. Consequently, a common strategy used in practice for solving problems on a general point set may involve either first decomposing the set into convex pieces, or approximating the set with its *convex hull*, i.e., the smallest convex set containing the given set. This makes the construction of the convex hull of a set a basic problem in computational geometry.

The first $O(n \log n)$ convex hull algorithm in two dimensions is due to Graham [**15**]. The algorithm first sorts the points by polar angle about an interior point, and then scans the sorted list to discard any point that would cause a reflex angle to occur (FIGURE 4(a)). Note that one may occasionally have to back up a long way and discard many points in order to restore convexity. However, as can be easily argued, the overall cost of the scanning step is still only linear in the total number of points. The convex hull can also be constructed by a divide-and-conquer algorithm ([**25**]). In this approach, we partition the points into two equal subsets P_L and P_R by a vertical line, find their convex hulls recursively, and merge them to form the desired convex hull. The merging step is accomplished by finding the *upper bridge* b_1 and the *lower bridge* b_2 between P_L and P_R (FIGURE 4(b)). To find the upper bridge, we take the line segment l connecting the rightmost point p of P_L and the leftmost point q of P_R, and move its endpoints upward iteratively along the boundaries of P_L and P_R, until l reaches the highest possible position, which now defines b_1. The lower bridge b_2 is found analogously. The time $T(n)$ required satisfies $T(n) \leq 2T(n/2) + O(n)$ which implies $T(n) = O(n \log n)$. If the input points are sorted, or more generally, form the (ordered) vertices of a simple polygon (i.e., a polygon without self-intersections), the convex hull can in fact be found in *linear* time (Graham and Yao [**16**], Lee [**20**]).

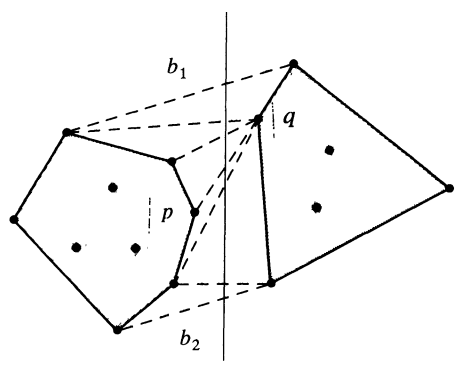
How fast can we compute the convex hull of a set in *three* dimensions? The divide-and-conquer algorithm described above, as it turns out, can be made to work in three dimensions within the same time bound $O(n \log n)$. In the crucial merging step of the algorithm, the hull of the union of two non-intersecting polyhedra can be formed by "gift wrapping" the two objects in one package, so to speak. The merging cost is proportional to the total size of the incidence graphs of the two polyhedra (i.e., the graphs induced by the vertices and edges of the polyhedra), and is $O(n)$ since these incidence graphs are planar [**24**].

For dimensions $d \geq 4$, the situation becomes more complex since a d-polytope with n vertices can have as many as $Cn^{\lfloor d/2 \rfloor}$ faces ([**17**]). There are two types of



Vertex p_2 is eliminated
because $\angle p_1 p_2 p_3$ is reflex;
then p_3 is eliminated because
$\angle p_1 p_3 p_4$ is reflex.

(*a*)
Graham's scan



(*b*)
Divide and conquer

FIG. 4

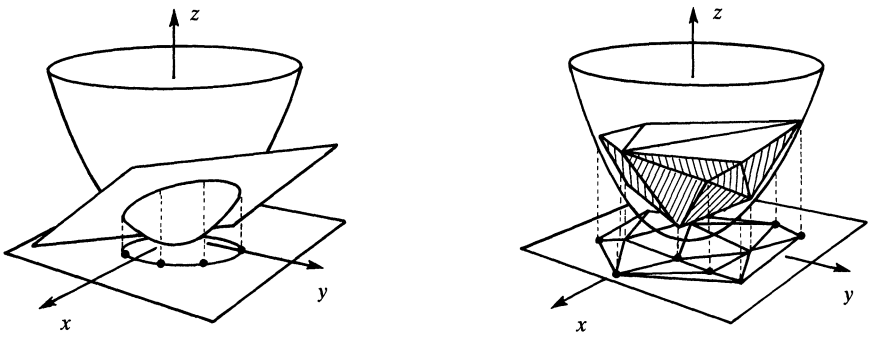
convex hull algorithms, depending on whether the algorithm only enumerates the *facets*, i.e., the $(d-1)$-faces, of the convex hull or produces the complete *facial lattice*, i.e., a description of all faces and incidence relationships of the convex polytope. Studies of these convex hull algorithms can be found in Chand and Kapur [**6**] and Seidel [**26**]. We note that, through duality, the problem of enumerating the facets of the convex hull of a point set is equivalent to the problem of enumerating the vertices of a polytope defined by a set of linear inequalities; in the latter form this problem has also been widely explored in the optimization literature (see Dyer [**10**] for a survey).

Connection between Voronoi Diagrams and Convex Hulls

Brown [5] was the first to observe that, through an appropriate transform the Voronoi diagram of a set S in R^d corresponds to the convex hull of the transformed set in R^{d+1}. We illustrate this for the simple case $d = 2$. Consider the mapping

$$\alpha: (x, y) \mapsto (x, y, x^2 + y^2)$$

which "lifts" a point (x, y) in the plane onto the paraboloid $z = x^2 + y^2$. Note that cocircular points in the plane are mapped by α into coplanar points in space (FIGURE 5(a)). Given n sites S in the plane, their image points under α form the vertices of a convex polyhedron P (since the paraboloid is convex). This means, for any "downward-looking" face f of the polyhedron P, its defining plane must lie "below" all other vertices of P. Therefore, by our earlier observation, the projection of f corresponds to a circle with no points of S in its interior. This implies that the projection of all the downward-looking faces of the polyhedron P gives precisely the faces of the Delaunay triangulation of the planar set S (FIGURE 5(b)). This relationship readily generalizes to higher dimensions, and enables one to use convex hull algorithms in $d + 1$ dimensions to obtain Voronoi diagrams in d dimensions!



(a) (b)

FIG. 5. Voronoi diagrams and convex hulls.

Combinatorial Geometry

The analysis of geometric algorithms often requires detailed combinatorial knowledge of the geometric structures involved. The book by Edelsbrunner [11] discusses many topics which overlap both computational and combinatorial geometry. We give one such example here.

Many algorithms in computational geometry involve calculation of the lower envelope (i.e., pointwise minimum) of a collection of functions. For example, suppose we are to render the 2-dimensional image of a 3-dimensional *terrain*, which is a polyhedral surface with the property that any line parallel to the z-axis intersects the surface at most once. From any viewpoint v, the "upper rim" (also called the *silhouette*) of the terrain as seen from v can be expressed as the lower envelope of a set of line segments in two dimensions (FIGURE 6).

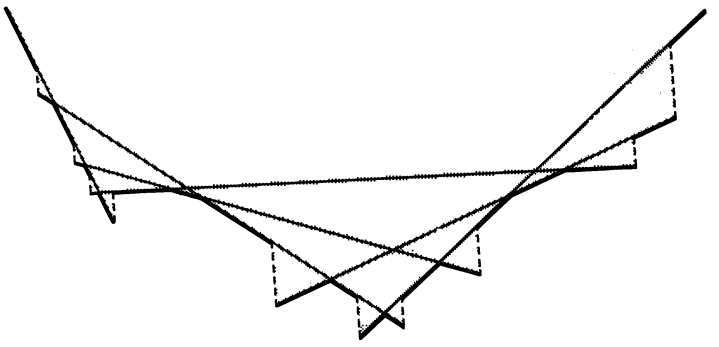


FIG. 6. The lower envelope of a set of line segments.

Lower envelopes are closely related to *Davenport-Schinzel sequences* (or *DS* sequences, for short). Specifically, an (n, s)-Davenport-Schinzel sequence is a sequence composed of n symbols satisfying: (i) no two adjacent elements are equal, and (ii) the sequence does not contain as a (not necessarily consecutive) subsequence of length $s + 2$ any alternation of two distinct symbols. The parameter s is called the *order* of the *DS* sequence. In the above example, the lower envelope of a set of n segments $\{l_1, \ldots, l_n\}$ in the plane is a *DS* sequence of order 3, since no alternation of the form $\cdots l_i \cdots l_j \cdots l_i \cdots l_j \cdots l_i \cdots$ can occur in the lower envelope. It has recently been shown that the maximum length $\lambda_3(n)$ of an $(n, 3)$-*DS* sequence is $\lambda_3(n) = \Theta(n\alpha(n))$, where $\alpha(n)$ is the extremely slowly growing functional inverse of the Ackermann function ([**19**]). In fact, this maximum value can be realized by a rather intricate arrangement of n line segments [**30**]. Thus the silhouette of a general polyhedral surface has complexity $\Theta(n\alpha(n))$, and can be computed in $O(n\alpha(n)\log n)$ time [**8**]. Other geometric problems where Davenport-Schinzel sequences naturally arise include the calculation of Euclidean shortest paths in 3-space amidst polyhedral obstacles, hidden-surface removal from a varying viewpoint, and the computation of other time-varying geometric configurations ([**4**], [**27**]).

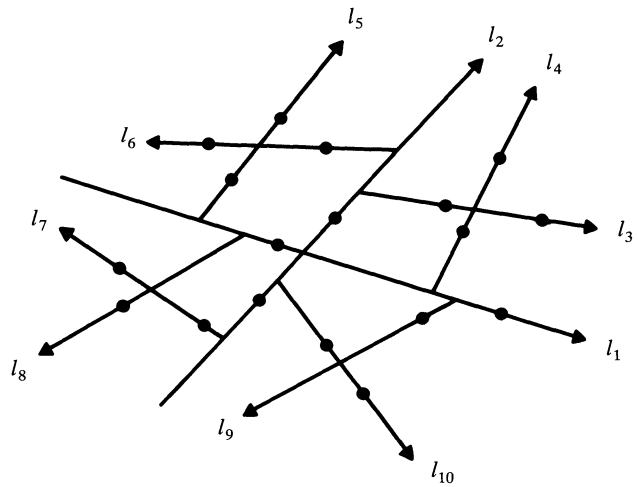
Space Partitions and Range Search

The post office problem mentioned earlier belongs to a class of problems studied in computational geometry known as "range search problems". In the most general setting, the database S to be searched may contain higher order objects than points, and the queries can be of a more complex nature. Still, let us confine our discussion here to the case when S is a set of points in R^d. An important type of query is the "half-space" query, in which an arbitrary linear inequality is specified, and all the points in S that satisfy the inequality should be identified quickly. As an example of a half-space query, consider a database where the first two coordinates contain, respectively, the income and the number of children of a household, and a query asks for all households with monthly income exceeding \$1000 plus \$200 for each child.

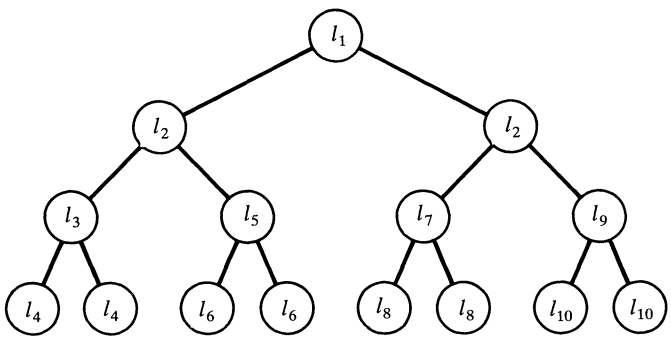
The first nontrivial solution to this problem was proposed by Willard [**31**]. It makes use of the following well-known fact: a density function over a bounded region can be partitioned by two straight lines l_1, l_2 into four quadrants with equal

mass. Translating this result into the discrete case, it says that a set S of n points in the plane can be partitioned by two lines into four (open) quadrants such that at most $n/4$ points are contained in the interior of each quadrant. One can then further partition the points in each quadrant, and repeat this process until all points are separated. This gives a tree structure for representing the set S (see FIGURE 7). For a query half-plane Q, since the line defining Q can intersect at most three of the four quadrants in each partition, one of the quadrants is either entirely inside or outside of Q, and hence can be exempted from further search. The search time $T(n)$ thus satisfies a recurrence relation $T(n) = 3T(n/4) + O(1)$, which yields $T(n) = O(n^{\alpha})$ with $\alpha = \log_4 3 \cong 0.774$.

Can we extend such 4-*partitions* in R^2 to 8-*partitions* in R^3? The answer turns out to be yes. Given a set S of n points in R^3, one can always find three planes



(*a*)
A 4-partition for 14 points



(*b*)
The corresponding tree structure

FIG. 7

that form an 8-partition in the sense that at most $n/8$ points of S lie in each of the eight open regions. Thus, we can represent a point set in R^3 by recursive 8-partitions. Since an arbitrary plane in R^3 can intersect at most seven of the eight open regions defined by an 8-partition, this leads to a retrieval time of $O(n^\alpha)$ with $\alpha = \log_8 7 \cong 0.936$ for half-space queries.

The 8-*partition* proposition stated above is considerably more difficult to prove than the two-dimensional analogue [**34**]. However, we note that a weaker version of it actually suffices to guarantee the same query time $O(n^{\log_8 7})$. In this version, the partition consists of a bisecting plane H for S, together with two *separate* 4-partitions for the sets S^+, S^- above and below H, with the condition that the half-axes of the two partitions are lined up exactly (FIGURE 8).

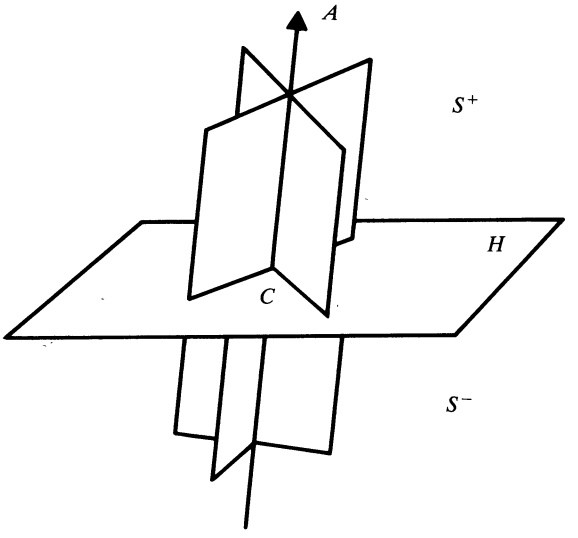


FIG. 8. An 8-partition with axis A and center C.

To show that such a partition exists, we invoke the well known Borsuk-Ulam theorem.

THEOREM (Borsuk-Ulam, [**23**]). *Let* f: $S^d \mapsto R^d$ *be a continuous, antipodal map, i.e.,* $f(-\nu) = -f(\nu)$ *for* $\nu \in S^d$, *the unit d-sphere. Then there is a point* $\nu \in S^d$ *such that* $f(\nu) = 0$.

To obtain the desired partition, consider $p_\nu(S^+)$, the projection of S^+ onto H along direction ν, where ν is any vector in the southern hemisphere of S^2. We find a (unique) 4-partition (L_1, L_2) for the planar set $p_\nu(S^+)$ with the constraint that L_1 is parallel to the x-axis. Call the intersection point of L_1 and L_2 the *center* of $p_\nu(S^+)$, denoted by $C(\nu)$. Similarly define the center $C'(\nu)$ for $p_{-\nu}(S^-)$, which is the projection of S^- along the direction $-\nu$. Applying the Borsuk-Ulam theorem for $d = 2$, it follows that the function $f(\nu) = C(\nu) - C'(\nu)$ must vanish for some vector ν_0. This gives us what we want: separate 4-partitions for S^+ and S^- which share a common axis A, where A has direction ν_0 and goes through the point $C(\nu_0) = C'(\nu_0)$. See [**32**] for complete details.

It is easy to see that the resulting 8-partition does have the property that any query plane can intersect at most 7 out of the 8 regions. Furthermore, this version can be generalized to higher dimensions by an inductive argument, thus allowing us to solve half-space queries in any number of dimensions.

It is an interesting open question whether the (strong) 8-partition proposition can be generalized to four dimensions; that is, given a density function over a bounded region in four dimensions, whether it can always be partitioned by 4 hyperplanes into 16 orthants with equal mass. However, it is not hard to show that (strong) 2^d-partitions are not always possible in dimensions $d \geq 5$. For example, take thirteen small balls of equal mass in R^5, and let the balls be in general position, so that no hyperplane can intersect more than five of them. Then, in any 2^5-partition, each ball must be cut by at least two hyperplanes, since otherwise some orthant would contain at least half of a ball, and thus at least $1/26$ of the total mass, which is more than the required fraction $1/32$. Therefore, for all thirteen balls, at least 26 instances of hyperplane-ball intersections are needed. Since the balls are in general position, five hyperplanes can provide at most 25 such instances and we have a contradiction.

Lower Bounds

An important, and often difficult, challenge in algorithm design is to know whether an algorithm can still be improved upon, or in fact one has already found the fastest algorithm possible. To resolve such questions, we must establish *lower bounds* to the computational complexity of a problem in some fairly general model.

For geometric problems, a natural model for studying computational complexity is the *algebraic computation tree model*. In this model, an algorithm can in one step either perform an arithmetic operation, or make a comparison and then branch according to the result of the comparison. (Thus, each comparison in fact corrresponds to a polynomial test of arbitrary degree evaluated at the input.) FIGURE 9 gives an example of an algebraic computation tree for testing whether a point (x, y) lies in the region

$$W = \{(x, y) | xy > 1,\ -2.5 < x + y < 3\}.$$

Let us consider the problem of finding a closest pair among n points in the plane. As we saw before, this problem can be solved in $O(n \log n)$ time by first constructing the Voronoi diagram and then performing a search in linear time. It turns out that $\Omega(n \log n)$ is also a lower bound for this problem in the algebraic computation tree model. In fact, the simpler problem of *deciding* whether the input set contains a *repeated* point, i.e., whether $p_i = p_j$ for some $i \neq j$, has complexity $\Omega(n \log n)$ even in *one* dimension! This result is due to Ben-Or [**3**], and the main idea of the proof is the following. The upper bound follows simply by sorting the n points and testing adjacent pairs for equality. The lower bound is obtained by mapping the polynomial inequalities to a suitable algebraic variety and then associating the number of computational steps with the Betti numbers of the algebraic variety. Let $x = (x_1, x_2, \ldots, x_n) \in R^n$ denote the input, and let T be an algebraic computation tree that decides whether $x \in W$ where W is defined by

$$W = \left\{ (x_1, x_2, \ldots, x_n) \,\middle|\, \prod_{i \neq j} (x_i - x_j) \neq 0 \right\} \subseteq R^n.$$

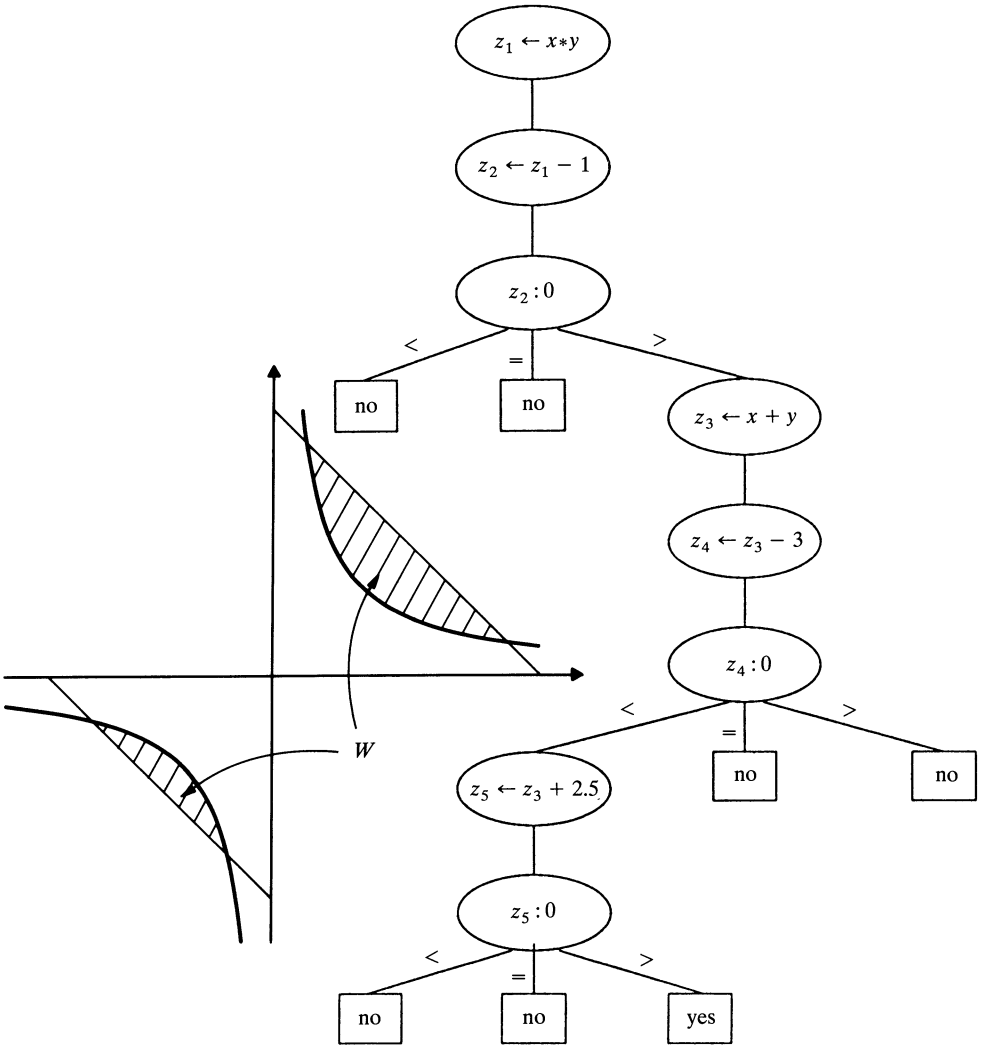


FIG. 9. An algebraic computation tree for testing "Is $(x, y) \in W$."

It is easy to see that W has $n!$ connected components. At the same time, by invoking a theorem of Milnor-Thom ([**22**], [**28**]), one can show that the algebraic variety V associated with a leaf of T has at most 3^{n+h} components, if the path to that leaf involves h operations. For a leaf associated with a "yes" answer, each connected component of V must be completely contained in some connected component of W. We thus conclude that the maximum path length h has to satisfy $3^h 3^{n+h} \geq n!$, which implies $h = \Omega(n \log n)$.

Common Techniques

As we have seen, computational geometry uses concepts and results from classical geometry, topology, combinatorics, as well as standard algorithmic techniques such as sorting and searching, graph manipulations, and linear program-

ming. In addition, certain special techniques and paradigms have emerged and found repeated application in computational geometry. We briefly summarize some of the most common ones below. Much more comprehensive surveys can be found in [11], [25] and [33].

1. Divide and conquer. The basic idea is to divide the problem into two or more subproblems, solve them recursively and then merge the results to obtain a solution to the original problem. Although this technique is common in algorithm design, it seems particularly well suited for solving geometric problems: the dividing step can naturally be accomplished by splitting along a line (or a hyperplane in d dimensions), and the merging step can take advantage of reduced problem complexity across the dividing hyperplane. The convex hull algorithms in two and three dimensions are good examples.

2. The sweep technique. This is a technique designed to reduce the dimension of a (static) geometric problem at the cost of changing its mode from static to dynamic. For example, in the construction of Voronoi diagrams, the sweep-line induces one-dimensional cross-sections which change dynamically (but only locally) when the sweep-line moves from one (discrete) position to the next. The sweep technique is also quite useful in higher dimensions.

3. Geometric transformations. Through a geometric transformation, one can often state a given problem in an alternative form which, though equivalent, may shed new light on the problem. One of the transforms used most often in computational geometry is the *dual transform* between points and hyperplanes. Other examples include the hyperbolic transform of Fortune for Voronoi diagrams and the correspondence between Voronoi diagrams and convex hulls (both discussed earlier), the use of *Plücker* coordinates to represent a line in three dimensions as a point in five dimensions, and the *skewed projection* determined by a pair of lines in three dimensions.

4. The locus approach. This approach applies to geometric query problems and it works as follows. We interpret the query as a point in some space (through an initial transformation, if necessary), and then partition that space into a set of non-overlapping regions so that the answer is invariant for all query points that fall in the same region. Several ingredients are involved in this approach: transformation of the query into a point, definition and construction of a suitable space partition, and an algorithm for point-location. Using the Voronoi diagram to solve the nearest-neighbor query is a basic example of this approach.

5. Random sampling. The idea is to use random sampling of the input objects to carry out divide-and-conquer efficiently, that is, to split up the original problem into subproblems each guaranteed to be of small size [7]. The expected performance of the resulting algorithm is with respect to the built-in randomization process, and does not depend on any assumption about the input distribution. This technique often yields simple algorithms with good expected performance.

Conclusion

We have tried here to give a brief sketch of some of the main ideas underlying the dynamically growing field of computational geometry. It is the natural convergence of ideas from many areas of mathematics such as topology, combinatorics,

algebra, probability and, of course, geometry, with those from computer science, such as graph algorithms, data structures, and optimization. We feel confident that the current trend of studying computational questions on geometric objects will continue to suggest new classes of problems which in turn will enrich and become a permanent part of the glorious tradition of geometry.

REFERENCES

1. A. Aggarwal, L. J. Guibas, J. Saxe and P. Shor, A linear time algorithm for computing the Voronoi diagram of a convex polygon, *Proc. 19th Ann. ACM Symp. Theory Comput.*, (1987) 39–45.
2. F. Aurenhammer, Voronoi diagrams—a survey, Inst. for Inf. Prc., Graz Tech. Univ. Tech. report, 1988.
3. M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Ann. Symp. Theory Comput.*, (1983) 80–86.
4. M. Bern, D. Dobkin, D. Eppstein and R. Grossman, Visibility with a moving point of view, *Proc. 1st Ann. ACM-SIAM Symposium on Discrete Algorithms* (1990) 107–117.
5. K. Q. Brown, Voronoi diagrams from convex hulls, *Inform. Process. Lett.*, 9 (1979) 223–228.
6. D. R. Chand and S. S. Kapur, An algorithm for convex polytopes, *J. ACM*, 17 (1970) 78–86.
7. K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete and Comput. Geometry*, 2 (1987) 195–222.
8. R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *J. Symbolic Computation*, 7 (1989) 11–30.
9. M. Davis, Y. Matijasevič and J. Robinson, Hilbert's Tenth Problem. Diophantine Equations: Positive aspects of a negative solution, in Mathematical developments arising from Hilbert Problems (F. Browder, ed.), *Proc. Symp. Pure Math 28, part 2*, (1976), 323–378.
10. M. E. Dyer, The complexity of vertex enumeration methods, *Math. Oper. Res.*, 8 (1983) 381–402.
11. H. Edelsbrunner, Algorithms in Combinatorial Geometry, Springer-Verlag, Heidelberg, Germany, 1987.
12. S. J. Fortune, A sweepline algorithm for Voronoi diagrams, *Proc. 2nd Ann. ACM Symp. Comput. Geom.* (1986) 313–322.
13. M. R. Garey and D. S. Johnson, Computers and Intractability: A Guide to the Theory of NP-completeness, W. H. Freeman and Co., San Francisco, 1979.
14. E. N. Gilbert, Random subdivisions of space into crystals, *Annals of Math. Stat.*, 33 (1962) 958–972.
15. R. L. Graham, An efficient algorithm for determining the convex hull of a finite planar set, *Inform. Process. Lett.*, **1** (1972) 132–133.
16. R. L. Graham and F. F. Yao, Finding the convex hull of a simple polygon, *J. Algorithms*, 4 (1983) 324–331.
17. B. Grünbaum and G. C. Shephard, Tilings and Patterns, W. H. Freeman and Co., New York, 1987.
18. L. J. Guibas and J. Stolfi, Ruler, compass and computer: The design and analysis of geometric algorithms, DEC Sys. Res. Center Tech. Dep. 37 (1989), 55 pp.
19. H. Hart and M. Sharir, Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes, *Combinatorica*, 6 (1986) 151–177.
20. D. T. Lee, On finding the convex hull of a simple polygon, *Int. J. Comput. Inform. Sci.*, 12 (1983) 87–98.
21. C. G. Lekkerkerker, Geometry of Numbers, Wiley Interscience, New York, 1969.
22. J. Milnor, On the Betti numbers of real algebraic varieties, *Proc. AMS*, 15 (1964) 275–280.
23. J. R. Munkres, Elements of Algebraic Topology, Addison-Wesley, Reading, MA, 1984.
24. F. P. Preparata and S. J. Hong, Convex hulls of finite sets of points in two and three dimensions, *Comm. ACM*, 2 (1977) 87–93.
25. F. P. Preparata and M. I. Shams, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.
26. R. Seidel, A convex hull algorithm optimal for point sets in even dimensions, Univ. of B.C., CS Tech Rep. 81–14, 1981.
27. J. T. Schwartz, M. Sharir and J. Hopcroft, Planning, Geometry and Complexity of Robot Motion, Ablex, Norwood, NJ, 1986.

28. R. Thom, Sur l'homologie des variétés algébriques réelles. Differential and Combinatorial Topology (ed., S. S. Cairns), Princeton Univ. Press, 1965.
29. M. G. Voronoi, Nouvelles applications des parametres continus a la théorie des formes quadratiques, *J. Reine u. Agnew. Math.*, 134 (1908) 198–287.
30. A. Wiernik and M. Sharir, Planar realizations of nonlinear Davenport-Schinzel sequences, *Discrete and Comput. Geometry* 3 (1988), 15–48.
31. D. E. Willard, New data structures for orthogonal range queries, *SIAM J. Computing*, 14 (1985) 232–253.
32. A. C. Yao and F. F. Yao, A general appraoch to d-dimensional geometric queries, Proc. 17th Ann. ACM Symp. on Theory of Computing, 1985, 163–168.
33. F. F. Yao, Computational Geometry (to appear as a chapter in Handbook of Theoretical Computer Science, North Holland, Amsterdam, 1990).
34 F. F. Yao, D. P. Dobkin, H. Edelsbrunner and M. S. Paterson, Partitioning space for range queries, *SIAM J. Computing*, 18 (1989) 371–384.