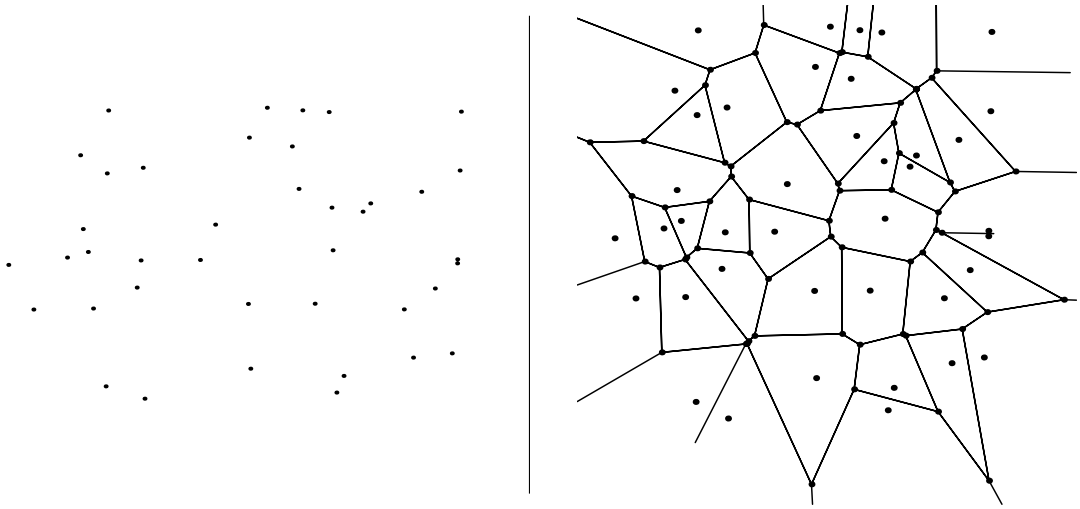

INPUT OUTPUT

17.4 Voronoi Diagrams

Input description: A set S of points $p_1, \ldots, p_n$.

Problem description: Decompose space into regions around each point such that all points in the region around p_i are closer to p_i than any other point in S.

Discussion: Voronoi diagrams represent the region of influence around each of a given set of sites. If these sites represent the locations of McDonald's restaurants, the Voronoi diagram partitions space into cells around each restaurant. For each person living in a particular cell, the defining McDonald's represents the closest place to get a Big Mac.

Voronoi diagrams have a surprising variety of applications:

- *Nearest neighbor search* – Finding the nearest neighbor of query point q from among a fixed set of points S is simply a matter of determining the cell in the Voronoi diagram of S that contains q. See Section 17.5 (page 580) for more details.

- *Facility location* – Suppose McDonald's wants to open another restaurant. To minimize interference with existing McDonald's, it should be located as far away from the closest restaurant as possible. This location is always at a vertex of the Voronoi diagram, and can be found in a linear-time search through all the Voronoi vertices.

- *Largest empty circle* – Suppose you needed to obtain a large, contiguous, undeveloped piece of land on which to build a factory. The same condition used to select McDonald's locations is appropriate for other undesirable facilities, namely that they be as far as possible from any relevant sites of interest. A Voronoi vertex defines the center of the largest empty circle among the points.

- *Path planning* – If the sites of S are the centers of obstacles we seek to avoid, the edges of the Voronoi diagram define the possible channels that maximize the distance to the obstacles. Thus the "safest" path among the obstacles will stick to the edges of the Voronoi diagram.

- *Quality triangulations* – In triangulating a set of points, we often desire nice, fat triangles that avoid small angles and skinny triangles. The *Delaunay triangulation* maximizes the minimum angle over all triangulations. Furthermore, it is easily constructed as the dual of the Voronoi diagram. See Section 17.3 (page 572) for details.

Each edge of a Voronoi diagram is a segment of the perpendicular bisector of two points in S, since this is the line that partitions the plane between the points. The conceptually simplest method to construct Voronoi diagrams is randomized incremental construction. To add another site to the diagram, we locate the cell that contains it and add the perpendicular bisectors separating this new site from all sites defining impacted regions. If the sites are inserted in random order, only a small number of regions are likely to be impacted with each insertion.

However, the method of choice is Fortune's sweepline algorithm, especially since robust implementations of it are readily available. The algorithm works by projecting the set of sites in the plane into a set of cones in three dimensions such that the Voronoi diagram is defined by projecting the cones back onto the plane. Advantages of Fortune's algorithm include that (1) it runs in optimal $\Theta(n \log n)$ time, (2) it is reasonable to implement, and (3) we need not store the entire diagram if we can use it as we sweep over it.

There is an interesting relationship between convex hulls in $d+1$ dimensions and Delaunay triangulations (or equivalently Voronoi diagrams) in d-dimensions. By projecting each site in E^d $(x_1, x_2, \ldots, x_d)$ into the point $(x_1, x_2, \ldots, x_d, \sum_{i=1}^{d} x_i^2)$, taking the convex hull of this $(d+1)$-dimensional point set and then projecting back into d dimensions, we obtain the Delaunay triangulation. Details are given in the Notes section, but this provides the best way to construct Voronoi diagrams in higher dimensions. Programs that compute higher-dimensional convex hulls are discussed in Section 17.2 (page 568).

Several important variations of standard Voronoi diagrams arise in practice. See the references below for details:

- *Non-Euclidean distance metrics* – Recall that Voronoi diagrams decompose space into regions of influence around each of the given sites. We have assumed that Euclidean distance measures influence, but this is inappropriate

for certain applications. If people drive to McDonald's, the time it takes to get there depends upon where the major roads are. Efficient algorithms are known for constructing Voronoi diagrams under a variety of different metrics, and for curved or constrained objects.

- *Power diagrams* – These structures decompose space into regions of influence around the sites, where the sites are no longer constrained to have all the same power. Imagine a map of radio stations operating at a given frequency. The region of influence around a station depends both on the power of its transmitter and the position/power of neighboring transmitters.

- *Kth-order and furthest-site diagrams* – The idea of decomposing space into regions sharing some property can be taken beyond closest-point Voronoi diagrams. Any point within a single cell of the kth-order Voronoi diagram shares the same set of k's closest points in S. In furthest-site diagrams, any point within a particular region shares the same furthest point in S. Point location (see Section 17.7 (page 587)) on these structures permits fast retrieval of the appropriate points.

Implementations: Fortune's Sweep2 is a widely used 2D code for Voronoi diagrams and Delaunay triangulations, written in C. This code is simple to work with if all you need is the Voronoi diagram. It is based on his sweepline algorithm [For87] for Voronoi diagrams and is available from Netlib (see Section 19.1.5 (page 659)) at *http://www.netlib.org/voronoi/*.

Both the CGAL (*www.cgal.org*) and LEDA (see Section 19.1.1 (page 658)) libraries offer C++ implementations of a variety of Voronoi diagram and Delaunay triangulation algorithms in two and three dimensions.

Higher-dimensional and furthest-site Voronoi diagrams can be constructed as a special case of higher-dimensional convex hulls. Qhull [BDH97] is a popular low-dimensional convex-hull code, useful for from two to about eight dimensions. It is written in C and can also construct Delaunay triangulations, Voronoi vertices, furthest-site Voronoi vertices, and half-space intersections. Qhull has been widely used in scientific applications and has a well-maintained homepage at *http://www.qhull.org/*. Another choice is Ken Clarkson's higher-dimensional convex-hull code, *Hull*, available at *http://www.netlib.org/voronoi/hull.html*.

Notes: Voronoi diagrams were studied by Dirichlet in 1850 and are occasionally referred to as *Dirichlet tessellations*. They are named after G. Voronoi, who discussed them in a 1908 paper. In mathematics, concepts get named after the last person to discover them.

The book by Okabi, et al. [OBSC00] is the most complete treatment of Voronoi diagrams and their applications. Aurenhammer [Aur91] and Fortune [For04] provide excellent surveys on Voronoi diagrams and associated variants such as power diagrams. The first $O(n \lg n)$ algorithm for constructing Voronoi diagrams was based on divide-and-conquer and is due to Shamos and Hoey [SH75]. Good expositions of both Fortune's sweepline algorithm [For87] for constructing Voronoi diagrams in $O(n \lg n)$ and the relationship

between Delaunay triangulations and $(d+1)$-dimensional convex hulls [ES86] include [dBvKOS00, O'R01].

In a kth-order Voronoi diagram, we partition the plane such that each point in a region is closest to the same set of k sites. Using the algorithm of [ES86], the complete set of kth-order Voronoi diagrams can be constructed in $O(n^3)$ time. By doing point location on this structure, the k nearest neighbors to a query point can be found in $O(k + \lg n)$. Expositions on kth-order Voronoi diagrams include [O'R01, PS85].

The smallest enclosing circle problem can be solved in $O(n \lg n)$ time using $(n-1)$st order Voronoi diagrams [PS85]. In fact, there exists a linear-time algorithm based on low-dimensional linear programming [Meg83]. A linear algorithm for computing the Voronoi diagram of a convex polygon is given by [AGSS89].

Related Problems: Nearest neighbor search (see page 580), point location (see page 587), triangulation (see page 572).