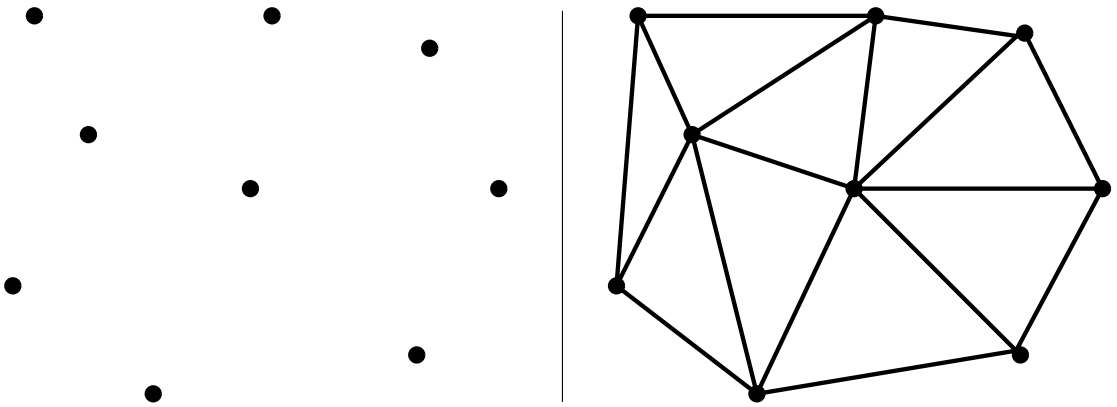

INPUT OUTPUT

17.3 Triangulation

Input description: A set of points or a polyhedron.

Problem description: Partition the interior of the point set or polyhedron into triangles.

Discussion: The first step in working with complicated geometric objects is often to break them into simple geometric objects. This makes triangulation a fundamental problem in computational geometry. The simplest geometric objects are triangles in two dimensions, and tetrahedra in three. Classical applications of triangulation include finite element analysis and computer graphics.

A particularly interesting application of triangulation is surface or function interpolation. Suppose that we have sampled the height of a mountain at a certain number of points. How can we estimate the height at any point q in the plane? We can project the sampled points on the plane, and then triangulate them. This triangulation partitions the plane into triangles, so we can estimate height by interpolating between the heights of the three points of the triangle that contain q. Furthermore, this triangulation and the associated height values define a mountain surface suitable for graphics rendering.

A triangulation in the plane is constructed by adding nonintersecting chords between the vertices until no more such chords can be added. Specific issues arising in triangulation include:

- *Are you triangulating a point set or a polygon?* – Often we are given a set of points to triangulate, as in the surface interpolation problem discussed

earlier. This involves first constructing the convex hull of the point set and then carving up the interior into triangles.

The simplest such $O(n \lg n)$ algorithm first sorts the points by x-coordinate. It then inserts them from left to right as per the convex-hull algorithm of page 105, building the triangulation by adding a chord to each point newly cut off from the hull.

- *Does the shape of the triangles in your triangulation matter?* – There are usually many different ways to partition your input into triangles. Consider a set of n points in convex position in the plane. The simplest way to triangulate them would be to add to the convex-hull diagonals from the first point to all of the others. However, this has the tendency to create skinny triangles.

  Many applications seek to to avoid skinny triangles, or equivalently, minimize small angles in the triangulation. The *Delaunay triangulation* of a point set minimizes the maximum angle over all possible triangulations. This isn't exactly what we are looking for, but it is pretty close, and the Delaunay triangulation has enough other interesting properties (including that it is dual to the Voronoi diagram) to make it the quality triangulation of choice. Further, it can be constructed in $O(n \lg n)$ time using implementations described below.

- *How can I improve the shape of a given triangulation?* – Each internal edge of any triangulation is shared between two triangles. The four vertices defining these two triangles form either (1) a convex quadrilateral, or (2) a triangle with a triangular bite taken out of it. The beauty of the convex case is that exchanging the internal edge with a chord linking the other two vertices yields a different triangulation.

  This gives us a local "edge-flip" operation for changing and possibly improving a given triangulation. Indeed, a Delaunay triangulation can be constructed from any initial triangulation by removing skinny triangles until no locally-improving exchange remains.

- *What dimension are we working in?* – Three-dimensional problems are usually harder than two-dimensional problems. The three-dimensional generalization of triangulation involves partitioning the space into four-vertex tetrahedra by adding nonintersecting faces. One important difficulty is that there is no way to tetrahedralize the interior of certain polyhedra without adding extra vertices. Furthermore, it is NP-complete to decide whether such a tetrahedralization exists, so we should not feel afraid to add extra vertices to simplify our problem.

- *What constraints does the input have?* – When we are triangulating a polygon or polyhedra, we are restricted to adding chords that do not intersect any of the boundary facets. In general, we may have a set of obstacles or

constraints that cannot be intersected by inserted chords. The best such triangulation is likely to be the so-called *constrained Delaunay triangulation*. Implementations are described next.

- *Are you allowed to add extra points, or move input vertices?* – When the shape of the triangles does matter, it might pay to strategically add a small number of extra "Steiner" points to the data set to facilitate the construction of a triangulation (say) with no small angles. As discussed above, *no* triangulation may exist for certain polyhedra without adding Steiner points.

To construct a triangulation of a convex polygon in linear time, just pick an arbitrary starting vertex v and insert chords from v to each other vertex in the polygon. Because the polygon is convex, we can be confident that none of the boundary edges of the polygon will be intersected by these chords, and that all of them lie within the polygon. The simplest algorithm for constructing general polygon triangulations tries each of the $O(n^2)$ possible chords and inserts them if they do not intersect a boundary edge or previously inserted chord. There are practical algorithms that run in $O(n \lg n)$ time and theoretically interesting algorithms that run in linear time. See the Implementations and Notes section for details.

Implementations: Triangle, by Jonathan Shewchuk, is an award-winning C language code that generates Delaunay triangulations, constrained Delaunay triangulations (forced to have certain edges), and quality-conforming Delaunay triangulations (which avoid small angles by inserting extra points). It has been widely used for finite element analysis and is fast and robust. Triangle is the first thing I would try if I needed a two-dimensional triangulation code. It is available at *http://www.cs.cmu.edu/~quake/triangle.html*.

Fortune's Sweep2 is a widely used 2D code for Voronoi diagrams and Delaunay triangulations, written in C. This code may be simpler to work with if all you need is the Delaunay triangulation of points in the plane. 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/*.

Mesh generation for finite element methods is an enormous field. Steve Owen's Meshing Research Corner (*http://www.andrew.cmu.edu/user/sowen/mesh.html*) provides a comprehensive overview of the literature, with links to an enormous variety of software. QMG (*http://www.cs.cornell.edu/Info/People/vavasis/qmg-home.html*) is a particularly recommended code.

Both the CGAL (*www.cgal.org*) and LEDA (see Section 19.1.1 (page 658)) libraries offer C++ implementations of an extensive variety of triangulation algorithms for two and three dimensions, including constrained and furthest site Delaunay triangulations.

Higher-dimensional Delaunay triangulations are a special case of higher-dimensional convex hulls. Qhull [BDH97] is a popular low-dimensional convex hull code, say 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: After a long search, Chazelle [Cha91] discovered a linear-time algorithm for triangulating a simple polygon. This algorithm is sufficiently hopeless to implement that it qualifies more as an existence proof. The first $O(n \lg n)$ algorithm for polygon triangulation was given by [GJPT78]. An $O(n \lg \lg n)$ algorithm by Tarjan and Van Wyk [TW88] followed before Chazelle's result. Bern [Ber04a] gives a recent survey on polygon and point-set triangulation.

The *International Meshing Roundtable* is an annual conference for people interested in mesh and grid generation. Excellent surveys on mesh generation include [Ber02, Ede06].

Linear-time algorithms for triangulating monotone polygons have been long known [GJPT78], and are the basis of algorithms for triangulating simple polygons. A polygon is monotone when there exists a direction d such that any line with slope d intersects the polygon in at most two points.

A heavily studied class of optimal triangulations seeks to minimize the total length of the chords used. The computational complexity of constructing this *minimum weight triangulation* was resolved when Rote [MR06] proved it NP-complete. Thus interest has shifted to provably good approximation algorithms. The minimum weight triangulation of a convex polygon can be found in $O(n^3)$ time using dynamic programming, as discussed in Section 8.6.1 (page 300).

Related Problems: Voronoi diagrams (see page 576), polygon partitioning (see page 601).