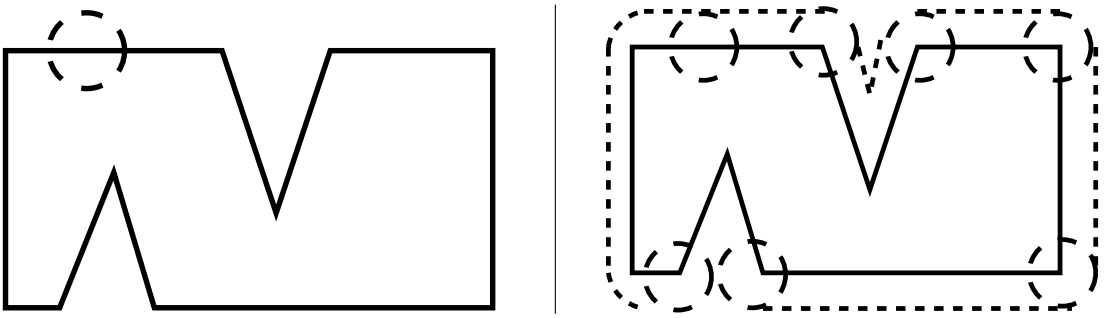

INPUT OUTPUT

17.16 Minkowski Sum

Input description: Point sets or polygons A and B, containing n and m vertices respectively.

Problem description: What is the convolution of A and B—i.e. , the Minkowski sum $A + B = \{x + y | x \in A, y \in B\}$?

Discussion: Minkowski sums are useful geometric operations that can *fatten* objects in appropriate ways. For example, a popular approach to motion planning for polygonal robots in a room with polygonal obstacles (see Section 17.14 (page 610)) fattens each of the obstacles by taking the Minkowski sum of them with the shape of the robot. This reduces the problem to the (more easily solved) case of point robots. Another application is in shape simplification (see Section 17.12 (page 604)). Here we fatten the boundary of an object to create a channel around it, and then let the minimum link path lying within this channel define the simplified shape. Finally, convolving an irregular object with a small circle will help smooth out the boundaries by eliminating minor nicks and cuts.

The definition of a Minkowski sum assumes that the polygons A and B have been positioned on a coordinate system:

$$A + B = \{x + y \mid x \in A, y \in B\}$$

where $x+y$ is the vector sum of two points. Thinking of this in terms of translation, the Minkowski sum is the union of all translations of A by a point defined within B. Issues arising in computing Minkowski sums include

- *Are your objects rasterized images or explicit polygons?* – The definition of Minkowski summation suggests a simple algorithm if A and B are rasterized images. Initialize a sufficiently large matrix of pixels by determining the size

of the convolution of the bounding boxes of A and B. For each pair of points in A and B, sum up their coordinates and darken the appropriate pixel. These algorithms get more complicated if an explicit polygonal representation of the Minkowski sum is needed.

- *Do you want to fatten your object by a fixed amount?* – The most common fattening operation expands a model M by a given tolerance t, known as *offsetting*. As shown in the figures above, this is accomplished by computing the Minkowski sum of M with a disk of radius t. The basic algorithms still work, although the offset is not a polygon. Its boundary is instead composed of circular arcs and line segments.

- *Are your objects convex or non-convex?* – The complexity of computing Minkowski sums depends in a serious way on the shape of the polygons. If both A and B are convex, the Minkowski sum can be found in $O(n+m)$ time by tracing the boundary of one polygon with another. If one of them is nonconvex, the *size* of the sum alone can be as large as $\Theta(nm)$. Even worse is when both A and B are nonconvex, in which case the *size* of the sum can be as large as $\Theta(n^2m^2)$. Minkowski sums of nonconvex polygons are often ugly in a majestic sort of way, with holes either created or destroyed in surprising fashion.

A straightforward approach to computing the Minkowski sum is based on triangulation and union. First, triangulate both polygons, then compute the Minkowski sum of each triangle of A against each triangle of B. The sum of a triangle against another triangle is easy to compute and is a special case of convex polygons, discussed below. The union of these $O(nm)$ convex polygons will be $A+B$. Algorithms for computing the union of polygons are based on plane sweep, as discussed in Section 17.8 (page 591).

Computing the Minkowski sum of two convex polygons is easier than the general case, because the sum will always be convex. For convex polygons it is easiest to slide A along the boundary of B and compute the sum edge by edge. Partitioning each polygon into a small number of convex pieces (see Section 17.11 (page 601)), and then unioning the Minkowski sum for each pair of pieces, will usually be much more efficient than working with two fully triangulated polygons.

Implementations: The CGAL (*www.cgal.org*) Minkowski sum package provides an efficient and robust code to find the Minkowski sums of two arbitrary polygons, as well as compute both exact and approximate offsets.

An implementation for computing the Minkowski sums of two convex polyhedra in $3D$ is described in [FH06] and available at *http://www.cs.tau.ac.il/∼efif/CD/*.

Notes: Good expositions on algorithms for Minkowski sums include [dBvKOS00, O'R01]. The fastest algorithms for various cases of Minkowski sums include [KOS91, Sha87].

The practical efficiency of Minkowski sum in the general case depends upon how the polygons are decomposed into convex pieces. The optimal solution is not necessarily the

partition into the fewest number of convex pieces. Agarwal et al. [AFH02] give a through study of decomposition methods for Minkowski sum.

The combinatorial complexity of the Minkowski sum of two convex polyhedra in three dimensions is completely resolved in [FHW07]. An implementation of Minkowski sum for such polyhedra is described in [FH06].

Kedem and Sharir [KS90] present an efficient algorithm for translational motion planning for polygonal robots, based on Minkowski sums.

Related Problems: Thinning (see page 598), motion planning (see page 610), simplifying polygons (see page 604).