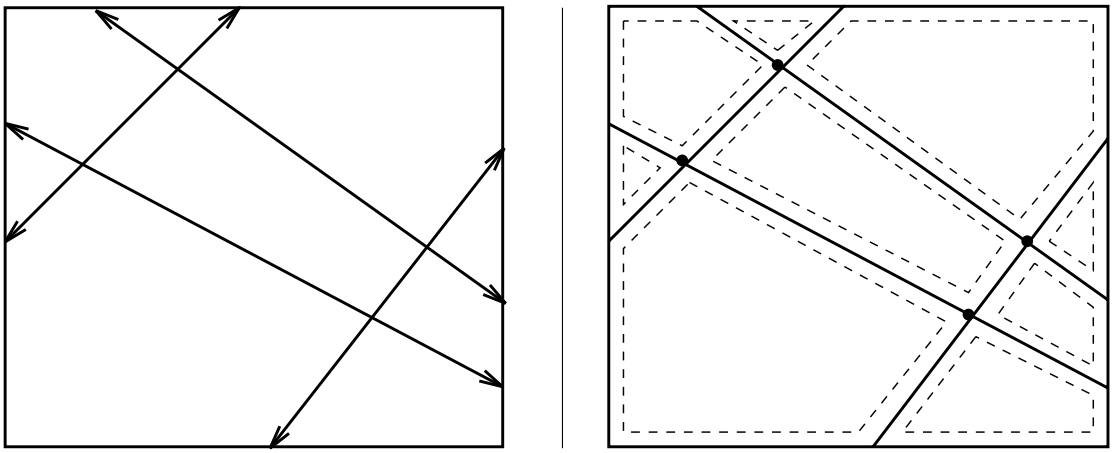


INPUT OUTPUT

17.15 Maintaining Line Arrangements

Input description: A set of lines and line segments $l_1, \ldots, l_n$.

Problem description: What is the decomposition of the plane defined by $l_1, \ldots, l_n$?

Discussion: A fundamental problem in computational geometry is explicitly constructing the regions formed by the intersections of a set of n lines. Many problems reduce to constructing and analyzing such an arrangement of a specific set of lines. Examples include:

- *Degeneracy testing* – Given a set of n lines in the plane, do any three of them pass through the same point? Brute-force testing of all triples takes $O(n^3)$ time. Instead, we can construct the arrangement of the lines and then walk over each vertex and explicitly count its degree, all in quadratic time.
- *Satisfying the maximum number of linear constraints* – Suppose that we are given a set of n linear constraints, each of the form $y \leq a_i x + b_i$. Which point in the plane satisfies the largest number of them? Construct the arrangement of the lines. All points in any region or *cell* of this arrangement satisfy exactly the same set of constraints, so we need to test only one point per cell to find the global maximum.

Thinking of geometric problems in terms of features in an arrangement can be very useful in formulating algorithms. Unfortunately, it must be admitted that

arrangements are not as popular in practice as might be supposed. Primarily, this is because some depth of understanding is required to apply them correctly. The computational geometry library CGAL now provides a general and robust enough implementation to justify the effort to do so. Issues arising in arrangements include

- *What is the right way to construct a line arrangement?* – Algorithms for constructing arrangements are incremental. Begin with an arrangement of one or two lines. Subsequent lines are inserted into the arrangement one at a time, yielding larger and larger arrangements. To insert a new line, we start on the leftmost cell containing the line and walk over the arrangement to the right, moving from cell to neighboring cell and splitting into two pieces those cells that contain the new line.

- *How big will your arrangement be?* – A geometric fact called the *zone theorem* implies that the kth line inserted cuts through k cells of the arrangement, and further that $O(k)$ total edges form the boundary of these cells. This means that we can scan through each edge of every cell we encounter on our insertion walk, confident that only linear total work will be performed while inserting the line into the arrangement. Therefore, the total time to insert all n lines in constructing the full arrangement is $O(n^2)$.

- *What do you want to do with your arrangement?* – Given an arrangement and a query point, we are often interested in identifying which cell of the arrangement contains the point. This is the problem of point location, discussed in Section 17.7 (page 587). Given an arrangement of lines or line segments, we are often interested in computing all points of intersection of the lines. The problem of intersection detection is discussed in Section 17.8 (page 591).

- *Does your input consist of points instead of lines?* – Although lines and points seem to be different geometric objects, appearances can be misleading. Through the use of *duality transformations*, we can turn line L into point p and vice versa:

  $$L : y = 2ax - b \leftrightarrow p : (a, b)$$

  Duality is important because we can now apply line arrangements to point problems, often with surprising results.

  For example, suppose we are given a set of n points, and we want to know whether any three of them all lie on the same line. This sounds similar to the degeneracy testing problem discussed above. In fact it is *exactly the same*, with only the role of points and lines exchanged. We can dualize our points into lines as above, construct the arrangement, and then search for a vertex with three lines passing through it. The dual of this vertex defines the line on which the three initial vertices lie.

It often becomes useful to traverse each face of an existing arrangement exactly once. Such traversals are called *sweepline algorithms*, and are discussed in some detail in Section 17.8 (page 591). The basic procedure sorts the intersection points by x-coordinate and then walks from left to right while keeping track of all we have seen.

Implementations: CGAL (*www.cgal.org*) provides a generic and robust package for arrangements of curves (not just lines) in the plane. This should be the starting point for any serious project using arrangements.

A robust code for constructing and topologically sweeping an arrangement in C++ is provided at *http://www.cs.tufts.edu/research/geometry/other/sweep/*. An extension of topological sweep to deal with the visibility complex of a collection of pairwise disjoint convex planar sets has been provided in CGAL.

Arrange is a package for maintaining arrangements of polygons in either the plane or on the sphere. Polygons may be degenerate, and hence represent arrangements of lines. A randomized incremental construction algorithm is used, and efficient point location on the arrangement is supported. *Arrange* is written in C by Michael Goldwasser and is available from *http://euler.slu.edu/∼goldwasser/publications/*.

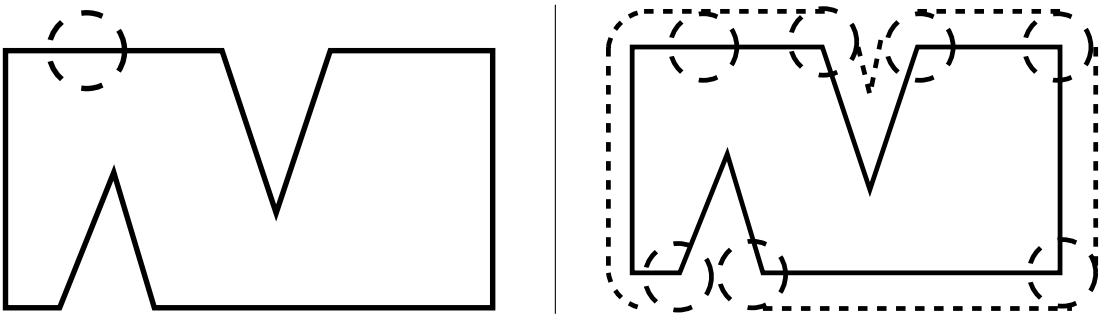
Notes: Edelsbrunner [Ede87] provides a comprehensive treatment of the combinatorial theory of arrangements, plus algorithms on arrangements with applications. It is an essential reference for anyone seriously interested in the subject. Recent surveys of combinatorial and algorithmic results include [AS00, Hal04]. Good expositions on constructing arrangements include [dBvKOS00, O'R01]. Implementation issues related to arrangements as implemented in CGAL are discussed in [FWH04, HH00].

Arrangements generalize naturally beyond two dimensions. Instead of lines, the space decomposition is defined by planes (or beyond 3-dimensions, *hyperplanes*). The zone theorem states that any arrangement of n d-dimensional hyperplanes has total complexity $O(n^d)$, and any single hyperplane intersects cells of complexity $O(n^{d-1})$. This provides the justification for the incremental construction algorithm for arrangements. Walking around the boundary of each cell to find the next cell that the hyperplane intersects takes time proportional to the number of cells created by inserting the hyperplane.

The history of the zone theorem has become somewhat muddled, because the original proofs were later found to be in error in higher dimensions. See [ESS93] for a discussion and a correct proof. The theory of Davenport-Schinzel sequences is intimately tied into the study of arrangements, which is presented in [SA95].

The naive algorithm for sweeping an arrangement of lines sorts the n^2 intersection points by x-coordinate and hence requires $O(n^2 \lg n)$ time. The *topological sweep* [EG89, EG91] eliminates the need to sort, and so traverses the arrangement in quadratic time. This algorithm is readily implementable and can be applied to speed up many sweepline algorithms. See [RSS02] for a robust implementation with experimental results.

Related Problems: Intersection detection (see page 591), point location (see page 587).



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