"To keep a moving line straight costs resources,
so we build it crooked right from the beginning?"

18 Topological sweep

Topological sorting means ordering n items according to an acyclic relation defined by m pairs. If m is proportional to n or less, there is an algorithm computing such an ordering in time $O(n)$ [5]. This is a factor $\log n$ faster than required to sort n items. The speed-up is in part possible because of the non-deterministic nature of the problem: any one of the linear orderings satisfying the m relations will do. A similar relaxation of the plane sweep idea leads to the topological sweep method. In a plane sweep, events are scheduled according to the time they are encountered by a straight line. Sometimes the relative order of two events is crucial, but in many cases events encountered in succession are independent and a decision of their order in time is in fact unnecessary. Of course, to distinguish the two cases may be difficult and more expensive than maintaining the priority queue needed to schedule events according to time stamps. A notable exception is the sweep of a line arrangement where the $\log n$ overhead for the priority queue can be reduced to a constant [2]. This is explained in this section.

Sweeping with a straight line. Let H be a set of n lines in $\mathbf{R}^2$, and let ℓ be a vertical line sweeping across $\mathbf{R}^2$ from left to right. For convenience, assume the lines are in general position, which includes no line in H is vertical. The sweep is accommodated by two data structures:

(i) the *cross-section*, which is an ordered list of the lines intersecting ℓ from bottom to top, and

(ii) the *event schedule*, which stores pairs of contiguous lines in the cross-section, ordered by the time ℓ encounters their intersection points.

An *elementary step* advanced ℓ over the next intersection point. This point is determined using the event schedule, which may be implemented as a priority queue. The corresponding two lines are switched in the cross-section. The switch breaks up to two adjacencies in the cross-section, each reflected by a possible deletion from the event schedule. Symmetrically, up to two new adjacencies are created, each possibly causing an insertion into the event schedule. There are $\binom{n}{2}$ elementary steps, one per vertex, requiring time $O(n^2 \log n)$ to maintain the priority queue. It is not clear whether or not this amount of time is necessary to perform the straight line sweep, or whether it is possible to exploit geometric dependencies to schedule the events in less than $O(\log n)$ time per step. This question is related to the open problem of sorting $X + Y$ in time asymptotically less than $O(n^2 \log n)$, see [4]. X and Y are real vectors of length n each, and $X + Y$ is the n-by-n array obtained by adding components pairwise.

Topological line. We replace ℓ by a topological line. This is a simple unbounded curve, t, that meets every line in one point, where it crosses the line. We require the unbounded ends of t lie vertically above and below all lines in H, see figure 18.1. Conceptually, an elementary step advances t across a single vertex to its right. For this to happen,

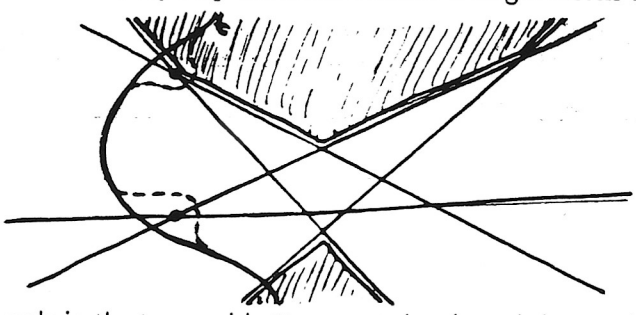

Figure 18.1: Since t starts and ends in the top- and bottommost chambers, it is meaningful to think of the topological sweep as directed from left to right.

a local change in the shape of t is sufficient. In the general case, there may be a number of possible elementary

steps, namely up to $\frac{n}{2}$, and any one will do. In figure 18.1, 2 of the 10 vertices to the right of t correspond to elementary steps. It is important that as long as there are any intersection points to the right of t, there is at least one elementary step. Take for example the leftmost vertex, u, to the right of t. Its 2 lines are necessarily contiguous along t, for otherwise, there would be vertices between t and u contradicting u is leftmost. It follows u corresponds to an elementary step, which implies the topological line never gets stuck before finishing the sweep. The important question is how to find an elementary step efficiently.

Horizon trees. Observe that two contiguous lines along t do not necessarily define an elementary step, even if they meet in a point, v, to the right of t. Other lines may interfere and intersect one or both lines between t and v. As an example, consider the bottommost two lines meeting t in figure 18.1. To detect such interferences, we store H in two tree structures, called the *upper* and the *lower horizon tree* of H and t. They are symmetric and we only explain the upper horizon tree. Intuitively, it is obtained by clipping each line to the left of t and to the right of any intersection with a line of larger slope, see figure 18.2. We call $b \in H$ a *child* of $a \in H$ if $a \cap b$ belongs to the

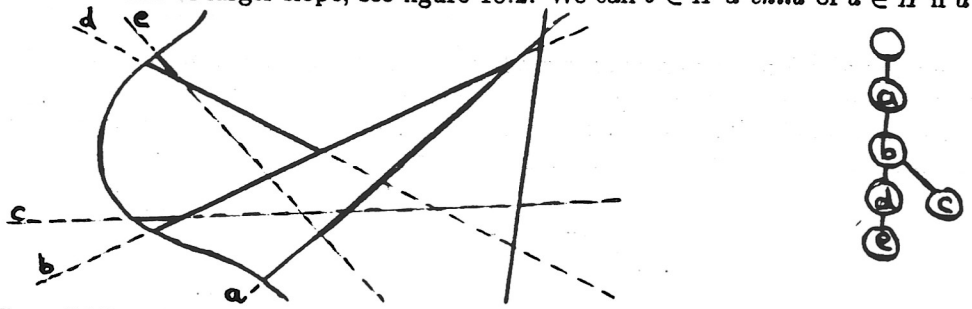

Figure 18.2: The solid lines indicate the upper horizon tree geometrically. The corresponding data structure is a rooted tree whose nodes are the lines in H and one artificially added line, which is the root.

geometric version of the tree and b is clipped to the right of this point. The thus defined structure is certainly a forest, and to get a tree in all cases we artificially add a line with infinitely large slope that meets each line to the right or all other lines. Observe that if a, b are contiguous along t and b is a child of a in the upper horizon tree, then no line below a can intersect a or b between t and $a \cap b$. To get the symmetric protection from above, we also store H in the symmetric lower horizon tree, see figure 18.3.

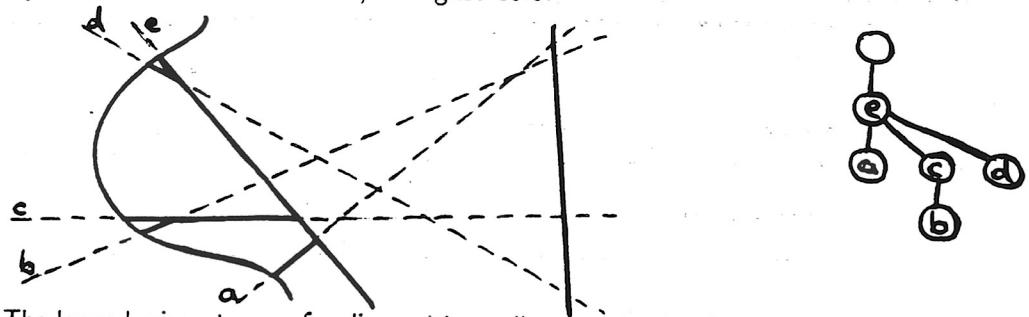

Figure 18.3: The lower horizon tree prefers lines with smaller slope in the direct comparison at intersection points.

LEMMA A. A contiguous pair $a, b \in H$ along t defines an elementary step iff b is a child of a in the upper horizon tree and a is a child of b in the lower horizon tree.

Initialization. At the beginning of time, t meets the lines to the left of all intersection points. The ordering of the lines along t at this moment is the same as the ordering by slope, which can be computed in time $O(n \log n)$ by sorting. Once the lines are sorted, the upper horizon tree can be built incrementally by adding the lines in the order of decreasing slope. Let $h_1, h_2, \ldots, h_n$ be this ordering and consider adding h_i to the upper horizon tree of the first $i-1$ lines, see figure 18.2. The tree decomposes the plane into i convex *bays* ordered from bottom to top. Because the slope of h_i is less than the slope of any line in the tree, the left part of h_i lies in the topmost bay. Follow h_i from left to right until it hits the boundary of this bay. It does this by intersecting a line of larger slope,

so h_i ends at this point. The horizon tree can thus be updated by finding the point where h_i hits the boundary of the topmost bay, which is determined by following the edges of this bay in a ccw order. Each line checked in this linear search, except for the last one whose edge meets h_i, no longer contributes an edge to the new topmost bay and will therefore not be checked again. It follows the time for the search can be charged in constant units to the n lines and $n+1$ bays. The total amount of time to initially construct the two horizon trees is thus $O(n)$, after sorting.

Algorithm. The topological sweep algorithm uses a linear array of lines to represent the cross-section of t. The horizon trees provide enough information to distinguish contiguous line pairs defining elementary steps from others. The event schedule is represented by a stack storing all current elementary steps. An entry in the stack is a pair of pointers into the linear array of lines. Each line in the array has direct links to its nodes in the two horizon trees. The sweep is started by initializing the upper and lower horizon trees as described. The stack is initialized by pushing all contiguous line pairs $a, b \in H$ satisfying the conditions of lemma A. From the array we get the $n-1$ contiguous line pairs, and the links to the trees allow testing the conditions of lemma A in constant time per pair. A high-level description of the main part of the algorithm follows.

```
while the stack is non-empty do
      pop the topmost elementary step, defined by a below b along t;
      clip a and b to the left of a ∩ b;
      extend b into the next lower bay in the upper horizon tree;
      extend a into the next higher bay in the lower horizon tree;
      push new elementary steps created by swapping a and b along t
endwhile.
```

The crucial part of the algorithm is how a and b are extended into the new bays to form the new horizon trees. This is explained for b and the upper horizon tree, see figure 18.4. b reaches into the bay below a. We need to find

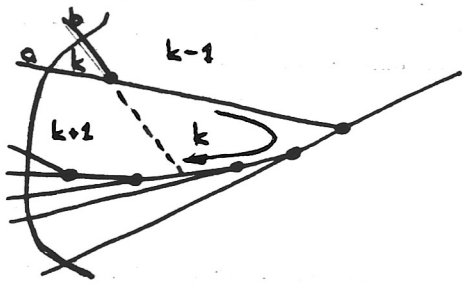

Figure 18.4: After the elementary step, b reaches into the bay below a. The new parent of b is the line supporting the edge of this bay intersected by b.

the edge of this bay intersected by b; the line containing this edge is the new parent of b in the upper horizon tree. This edge is determined by a linear search traversing the edges of this bay in a cw order, following child pointers from the parent of a downward.

REMARK. Alternatively, we could do the linear search by traversing the edges of the bay in a ccw order. This is indeed the way the algorithm is originally described [2]. In this case, the data structure for the horizon trees can be simplified because pointers to children are not needed. The proof that each elementary step takes only constant amortized time is more involved than the one for cw traversal explained below.

Amortized analysis. We present an argument implying a single elementary step takes only constant amortized time. This is certainly not true for the worst-case time because we can find examples where the linear search looks at almost all lines to find the proper extension of b. The argument uses chocolate chip cookies as the basic time or cost unit assigned to lines and intersection points. The lines are labeled from 1 through $n+1$, top to bottom in the upper horizon tree and bottom to top in the lower horizon tree. To state the invariant maintained by the argument, consider a line h, and let h_u and h_ℓ be its parents in the upper and lower horizon trees.

(I) The number of chocolate chip cookies carried by h is $k_u + k_\ell$, where k_u is the label of the bay right above $h \cap h_u$ in the upper horizon tree and k_ℓ is the label of the bay right below $h \cap h_\ell$ in the lower horizon tree.

This invariant can be satisfied initially by distributing fewer than n^2 cookies per horizon tree. Consider the elementary step defined by the kth line a and the $(k-1)$st line b from the top along t. In the upper horizon tree, a and b share bay k. By clipping a and b to the left of $a \cap b$, bay k disappears and reappears to the right of $a \cap b$, now below a and above b, see figure 18.4. Before the elementary step, b carries $k-1$ cookies justifying its position in the upper horizon tree. After the step, b needs k cookies to satisfy (I). It picks up the extra cooky at the intersection with a. The number of cookies required for a in the upper horizon tree remains unchanged. Each line traversed during the cw linear search of the old bay $k+1$ frees up a cooky, which is used to pay for its traversal.

The entire sweep can thus be understood as a cooky monster kept alive by cookies provided by the initial horizon trees and the intersection points. Two cookies per intersection point suffice to maintain (I), one for b in the upper horizon tree, the other for a in the lower horizon tree. This adds fewer than n^2 cookies to the fewer than $2n^2$ cookies required by the initial horizon trees. This implies fewer than 3 edges or lines are traversed on the average per elementary step.

THM. 18.1 The topological sweep of an arrangement of n lines in $\mathbf{R}^2$ takes time $O(n^2)$ and storage $O(n)$.

REMARKS. Sweeping a line arrangement is a process that generates no output by itself; it is useful only in combination with operations performed at the faces of the swept arrangement. Such applications can be found in [2, 3, 6]. There are difficulties in extending the topological sweep method to 3 and higher dimensions. As explained in [1, chapter 10.4], it is possible a topological plane gets stuck in a plane arrangement without elementary step left to advance it across the remaining intersection points.

Homework exercises

18.1 Let H be a set of n lines in general position in $\mathbf{R}^2$. Argue the sum of degrees of all unbounded chambers in $\mathcal{A}(H)$ is less than $8n$.

18.2 Let H be as in exercise 18.1. Show how to construct the unbounded chambers in time $O(n)$ assuming the lines are given in order of increasing slope.

18.3 Call a set of k points in $\mathbf{R}^2$ a *convex chain* if each point is a lower vertex of the convex hull of the set. Let S be a set of n points in $\mathbf{R}^2$. Show that topological sweep can be used to determine in time $O(n^2)$ and storage $O(n)$ the largest k so S contains a convex chain of size k.

References

[1] A. BJÖRNER, M. LAS VERGNAS, B. STURMFELS, N. WHITE AND G. ZIEGLER. *Oriented Matroids.* Cambridge Univ. Press, Cambridge, 1993.

[2] H. EDELSBRUNNER AND L. J. GUIBAS. Topologically sweeping an arrangement. *J. Comput. System Sci.* **38** (1989), 165–194. Corrigendum. *J. Comput. System Sci.* **42** (1991), 249–251.

[3] H. EDELSBRUNNER AND D. L. SOUVAINE. Computing least median of squares regression lines and guided topological sweep. *J. Amer. Statist. Assoc.* **85** (1990), 115–119.

[4] M. L. FREDMAN. On the information theoretic lower bound. *Theoret. Comput. Sci.* **1** (1976), 355–361.

[5] D. E. KNUTH. *Fundamental Algorithms. The Art of Computer Programming, Vol. 1.* Addison-Wesley, Reading, 1968.

[6] M. H. OVERMARS AND E. WELZL. New methods for computing visibility graphs. *In* "Proc. 4th Ann. Sympos. Comput. Geom., 1988", 164–171.