ted from JOURNAL OF COMPUTER AND SYSTEM SCIENCES
:hts Reserved by Academic Press, New York and London

Vol. 38, No. 1, February 1989
Printed in Belgium

Topologically Sweeping an Arrangement

HERBERT EDELSBRUNNER

University of Illinois, Urbana, Illinois

LEONIDAS J. GUIBAS

DEC/SRC, Palo Alto, California, and Stanford University, Stanford, California

Received October 29, 1986; revised November 11, 1987

Sweeping a collection of figures in the Euclidean plane with a straight line is one of the novel algorithmic paradigms that have emerged in the field of computational geometry. In this paper we demonstrate the advantages of sweeping with a topological line that is not necessarily straight. We show how an arrangement of n lines in the plane can be swept over in $O(n^2)$ time and $O(n)$ space by a such a line. In the process each element, i.e., vertex, edge, or region, is visited once in a consistent ordering. Our technique makes use of novel data structures which exhibit interesting amortized complexity behavior; the result is an algorithm that improves upon all its predecessors either in the space or the time bounds, as well as being eminently practical. Numerous applications of the technique to problems in computational geometry are given—many through the use of duality transforms. Examples include solving visibility problems, detecting degeneracies in configurations, computing the extremal shadows of convex polytopes, and others. Even though our basic technique solves a planar problem, its applications include several problems in higher dimensions. © 1989 Academic Press, Inc.

1. MOTIVATION

veeping a collection of figures in the Euclidean plane E^2 with an undirected
vertical) line is one of the novel algorithmic paradigms that have emerged in
ield of computational geometry [PS, NP]. In general, the sweep is supported
vo types of data structures: one that maintains the figures currently intersecting
:weeping line, and another that tells the sweeping line when to stop next. Such
s include the times when the set of intersected figures changes, as well as other
ts of interest. The stopping-times structure is most naturally implemented by a
:ity queue. This common solution, however, inherently entails the price of
ıtaining the priority queue, which is $O(\log n)$ per update if the queue has size n
.U]. The purpose of this paper is to demonstrate that, in certain situations,
: is a way to avoid having to pay this additional logarithmic cost factor. The
ıgs will be achieved by replacing the sweeping line by a "topological line," that
n unbounded simple curve that satisfies properties milder than straightness.

0022-0000/89 $3.00

Our specific problem will be that of sweeping a set of (infinite, straight) lines in E^2. Let H be a set of n lines in the plane. The set H dissects E^2 into a collection of convex *regions*, each bounded by *edges* which are segments of the lines in H. The boundaries of these segments, in turn, are points where the lines of H intersect. We shall term these endpoints *vertices*. We take the regions to be open, and the edges relatively open (with respect to the line they are on). The regions, edges, and vertices partition the plane into a subdivision known as the *arrangement* $\mathscr{A}(H)$ [G]. We will assume that $\mathscr{A}(H)$ is *simple*, in other words, that any two lines intersect at a vertex, but no three do so. We will also assume that none of the lines in H is vertical. In a later section we discuss how these restrictions can be removed. See Fig. 1.1 for an example of an arrangement.

Normally a subdivision is specified by listing all the *incidence relations* between its regions and edges, and its edges and vertices, in a way consistent with the natural cyclic orderings of the edges around a region and the edges around a vertex [GS]. A possible choice for an adequate representation of the arrangement consists of a set of vertex records, each containing the names of the four edges it is incident to, arranged in counterclockwise order, as well as a set of edge records, each containing the line the edge lies on, and the names of its left and right endpoints. See Guibas and Stolfi [GS] for a fuller account of the representation of planar subdivisions.

It is clear from the previous discussion that the size of the subdivision associated with $\mathscr{A}(H)$ is $\Theta(n^2)$. This subdivision can be constructed in $O(n^2 \log n)$ time by sweeping with a vertical straight line [EW]. Furthermore, the sweep uses only $O(n)$ storage in addition to the space needed to represent the arrangement. This is advantageous in applications where we are allowed to destroy the vertices, edges, and (implicitly) regions of the arrangement after creation and inspection. By a more intricate method it is possible to construct the arrangement in time $O(n^2)$, using an incremental approach [CGL, EOS] which involves introducing the lines of H one at a time. However, in this method $O(n^2)$ storage is intrinsic, since no part of the arrangement may be thrown away until all of it has been computed.

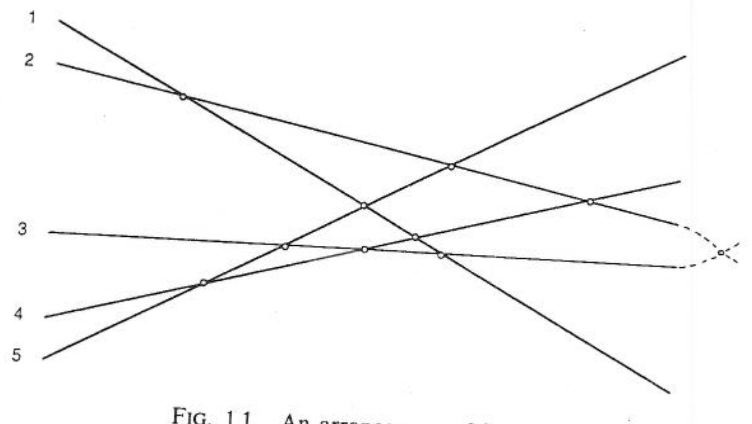


FIG. 1.1. An arrangement of five lines.

In this paper we will see how to use a "topological sweep" to compute $\mathscr{A}(H)$ in $O(n^2)$ time, *but with only $O(n)$ extra storage.* As we already mentioned, there are many applications where the elements of the arrangement (i.e., vertices, edges, and regions) need only be examined as they are built and then may be discarded immediately afterwards. Our method will allow these applications to run in $O(n^2)$ time and $O(n)$ space. There are also many problems that can be reduced to a number of two-dimensional sweeps (see problems (b), and (e)–(h) below, for instance). The time it takes to solve such problems is typically the number of required sweeps times $O(n^2)$. Some example applications where our technique improves existing bounds are listed below; E^k denotes Euclidean k-dimensional space:

(a) Compute the minimum area triangle spanned by three of n points in E^2.

(b) Compute a maximum subset of a given set of n points in E^2 whose elements define the vertices of a convex polygon; same question for an *empty* convex polygon, that is, one containing none of the given points in its interior.

(c) Compute the visibility graph of n non-intersecting segments in E^2.

(d) Given n segments in E^2, compute a line which intersects as many segments as possible.

(e) Enumerate all faces of an arrangement in E^d, $d \geqslant 2$.

(f) Test whether any $d+1$ points of a configuration of n points in E^d, $d \geqslant 2$, are in special position (do not span the full space).

(g) Given n non-zero vectors $v_1, ..., v_n$ in E^d, compute an assignment of $\{+1, -1\}$ to coefficients $\alpha_1, ..., \alpha_n$ such that $\sum \alpha_i v_i$ is longest.

(h) Compute the directions of minimum and maximum shadows for a convex polytope in E^d, $d \geqslant 3$.

It is remarkable that although our basic technique is strictly planar, there are many applications to problems in higher dimensions as well.

Besides the applications listed above, the method presented here is noteworthy for two additional reasons. One is that it is an illuminating example of amortized complexity analysis, a methodology that has recently become very popular in the analysis of algorithms [T]. Second, we have implemented our method and it works extremely well in practice, outperforming the straight-line sweep even for arrangements of only tens of lines.

Here is a quick summary of the structure of this paper: Section 2 contains various geometric preliminaries that we will employ throughout the exposition. Section 3 presents the topological plane sweep and its analysis. In Section 4 we deal briefly with a technique for handling degeneracies, and then Section 5 expands on the multifarious applications of the topological sweep, including all the problems mentioned above. Section 6 ends the paper with some open problems and conclusions.

2. Geometric Preliminaries

Let $l_1, l_2, ..., l_n$ denote the lines of an arrangement $\mathscr{A}(H)$. Without loss of generality we assume that when so written they are sorted according to slope, from smallest to largest. Our earlier assumptions about H being simple imply that all slopes are finite and distinct, so this ordering is well defined. The same assumptions allow us to define an "above" relation between elements of $\mathscr{A}(H)$. We will say that element A is *above* element B if A and B have intersecting projections on the x axis and, at each abscissa x of their intersection, all points of A are above all points of B. It is easy to check that, for any two distinct elemnets A and B with intersecting x-projections, either A is above B or B is above A. It is known that the "above" relation among the elements of a given subdivision is acyclic. For a discussion of this topic see, for example, the paper by Edelsbrunner *et al.* [EGS].

Lemma 2.1. *There is exactly one region that is not below any other region (denoted by $\mathscr{T}$ for "top") and exactly one region that is not above any other region (denoted by $\mathscr{B}$ for "bottom").*

Proof. Trivial. ∎

A (vertical) *cut* is a list $(c_1, c_2, ..., c_n)$ of edges of $\mathscr{A}(H)$ such that

(i) c_1 is an edge of $\mathscr{T}$ and c_n is an edge of $\mathscr{B}$, and

(ii) for each i, $1 \leqslant i \leqslant n-1$ we have that c_i and c_{i+1} are both incident upon region R_i such that c_i is above R_i and c_{i+1} is below R_i.

These conditions imply that no two edges of the cut lie on the same line of $\mathscr{A}(H)$, so there is a one-to-one correspondence between the edges of our cut and the lines of the arrangement. A cut will be our formal analog of the intuitive concept of a "topological line." Such a line cuts the arrangement along the edges of the cut, in the given sequence; see Fig. 2.1. We let above(l) and below(l) denote the open halfplanes bounded from below and from above by the non-vertical line l, respectively. Note that the region R_i referred to above is necessarily unique, as R_i is

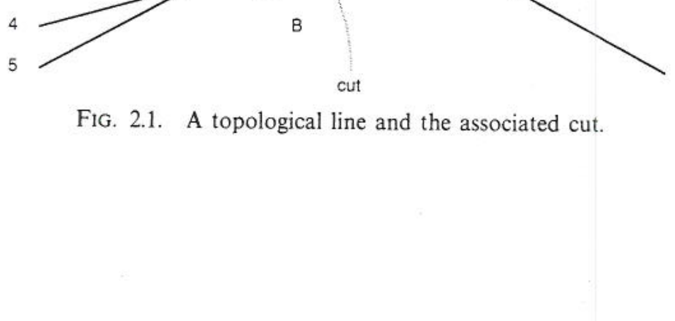


Fig. 2.1. A topological line and the associated cut.

below(c_i) and above(c_{i+1}). Since the "above" relation is acyclic, the same region cannot be reused in a cut.

We can define an appropriate "left-of" relation among cuts by considering that cut A is *left* of cut B if for every line l of the arrangement $\mathscr{A}(H)$, the edge of A on l is the same as, or to the left of, the edge of B on l. Among all cuts there is a "leftmost" one, consisting of the left-unbounded edges of each line $l_1, l_2, ..., l_n$, in this order; similarly, there is a "rightmost" cut, consisting of the right-unbounded edges of $l_1, l_2, ..., l_n$. Our topological sweep of the arrangement will be implemented by starting with the leftmost cut and pushing it to the right till it becomes the rightmost cut, in a series of elementary steps.

An *elementary* step is performed when the topological line sweeps past a vertex of the arrangement; it corresponds to a transposition in the underlying numbering of the lines as defined by the order in which they are intersected by the sweeping topological line. Obviously exactly $\binom{n}{2}$ elementary steps will be required to sweep the arrangement, in proceeding from the identity permutation of the lines to its reversal, between the leftmost and rightmost cuts. See Fig. 2.2 for an example.

We next state a lemma that shows that for any cut there always exists an elementary step that advances it to the right, unless it is the rightmost cut.

LEMMA 2.2. *There always exist two consecutive edges of the cut with a common right endpoint, unless we are considering the rightmost cut.*

Proof. An edge c_i terminates on the right at vertex v_i because an intersection occurs with another line l. Let c_j be the edge of the cut on l and v_j be the right endpoint of c_j. In fact there are two cases, as Fig. 2.3 shows, depending upon whether $i<j$ or $i>j$. In both cases we can conclude that either $v_i = v_j$, or v_j occurs to the left of v_i.

Now just consider the edge c_i of the cut with the leftmost right endpoint. Such an endpoint exists, because our cut is not rightmost. In this case we must have $v_i = v_j$ and in fact $j = i \pm 1$. ▮

The major dificulty in implementing the topological sweep is how to discover where in a cut an elementary step can be applied. To this end we introduce the axiliary notion of horizon trees. Before we do this let us make an attempt to find two edges of the cut with a common right endpoint without using any look-ahead structures. Indeed this is straightforward if each edge knows its right endpoint—after all two such edges are adjacent in the cut. But after the elementary step we are

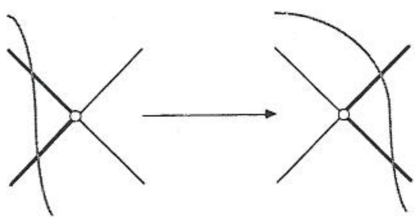


FIG. 2.2. An elementary step.

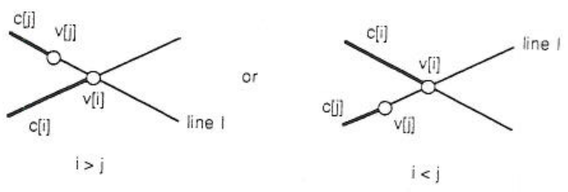


FIG. 2.3. The right endpoint of an edge of the cut.

facing the problem of computing the right endpoints of the two new edges in constant time. On the other hand, if we give up knowledge of the right endpoint for edges in the cut, then the computation of the two edges whose right endpoints coincide becomes nontrivial.

Let $(m_1, m_2, ..., m_n)$ denote the lines containing the edges $(c_1, c_2, ..., c_n)$, respectively. The *upper horizon tree* $T^+(C)$ of the cut C is constructed by starting with the edges of the cut and extending them to the right. When two edges come together at an intersection point, only the one of higher slope continues on to the right; the

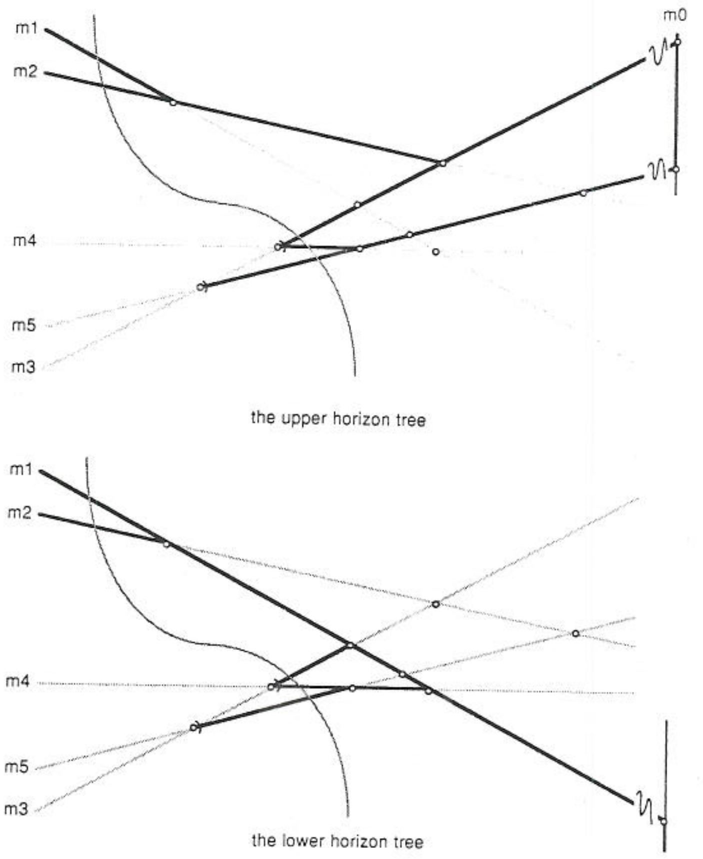


FIG. 2.4. The horizon trees of a cut.

other one stops at that point and is removed from further consideration. More formally, the upper horizon tree consists of one segment from each of the lines m_i, where a point p of m_i belongs to $T^+(C)$ if

(i) p is above all lines m_j with $j > i$, and

(ii) p is below all lines m_k satisfying both $k < i$ and having slope greater than the slope of m_i.

Figure 2.4 shows $T^+(C)$ for the cut of Fig. 2.1, as well as the symmetrically defined *lower horizon tree* $T^-(C)$ (where lines of lower slope are the winners). Observe that the edges of the cut belong to both trees, but the left endpoints of those edges belong to no tree.

There is an obvious defect in the definition of the upper horizon tree above, since it can turn out to be actually a forest and not a tree—consider the upper horizon tree of the rightmost cut in Fig. 2.4, for example. To rectify this minor problem we add the dummy vertical line m_0 at $x = +\infty$ and consider it as the last line in our list. For the lower horizon tree we add the same line again, but now consider it as the first in our list.

LEMMA 2.3. *The (rectified) definition of the upper horizon tree above truly defines a tree consisting of exactly one segment s_i^+ from each line m_i; furthermore, s_i^+ contains the edge c_i.*

Proof. We have

$$s_i^+ = \underset{\underset{\text{convex}}{\uparrow}}{m_i} \cap \underset{\underset{\text{convex}}{\uparrow}}{\bigcap_{j>i} \text{above}(m_i)} \cap \underset{\underset{\text{convex}}{\uparrow}}{\bigcap_{\substack{j<i \\ \text{slope}(m_j) > \text{slope}(m_i)}} \text{below}(m_j)}.$$

Each of the sets above obviously contains c_i. Their intersection is convex, and therefore a segment. Such a segment s_i^+ is terminated when it encounters another segment s_j^+ of higher slope. Thus the right endpoints of these segments naturally form a tree. ∎

We are interested only in what happens to the right of the topological line, as what is to the left has already been swept over. Consider two successive edges c_i and c_{i+1} of the cut. Let R_i^+ be the region of the plane to the right of the topological line and delimited by s_i^+ and s_{i+1}^+ in the subdivision defined by the upper horizon tree. Define R_i^- analogously.

Recall that R_i is the region of the arrangement that lies between c_i and c_{i+1} and let R_i' be R_i restricted to the right side of the topological line. We note that $R_i' = R_i^+ \cap R_i^-$, for to the right of the topological line we have

$$R_i^+ = \bigcap_{\substack{j \leqslant i \\ \text{slope}(m_j) \geqslant \text{slope}(m_i)}} \text{below}(m_j) \cap \bigcap_{j > i} \text{above}(m_j),$$

$$R_i^- = \bigcap_{j \leqslant i} \text{below}(m_j) \cap \bigcap_{\substack{j > i \\ \text{slope}(m_j) < \text{slope}(m_i)}} \text{above}(m_j),$$

$$R_i = \bigcap_{j \leqslant i} \text{below}(m_j) \cap \bigcap_{j > i} \text{above}(m_j).$$

The above relation follows, since the doubly indexed intersection in the formula for R_i^+ simplifies to below(m_i), and that in the formula for R_i^- simplifies to above(m_{i+1}). This relation, and its more obvious analog $C = T^+(C) \cap T^-(C)$ will be useful to us in the sequel.

3. The Topological Plane Sweep

We now come to the central part of this paper, which is the treatment of the updating required by the elementary steps of the previous section. Our goal is to design data structures for representing cuts and horizon trees, plus some auxiliary information needed for the implementation of the topological plane sweep. In the following we use a Pascal-like notation for expressing these structures. As it turns out, we need only very simple data structures for this problem:

$E[1:n]$	is the array of line equations: $E(i) = (a_i, b_i)$, if the ith line of H, l_i, is $y = a_i x + b_i$.
$HTU[1:n]$	is an array representing the upper horizon tree. $HTU[i]$ is a pair (λ_i, ρ_i) of indices indicating the lines that delimit the segment of l_i in the upper horizon tree to the left and to the right, respectively. If this segment is the leftmost on l_i we set $\lambda_i = -1$; if it is rightmost on l_i we set $\rho_i = 0$.
$HTL[1:n]$	represents the lower horizon tree and is defined similarly.
I	is a set of integers, represented as a stack. If i is in I, then c_i and c_{i+1} share a common right endpoint.
$M[1:n]$	is an array holding the current sequence of indices that form the lines $m_1, m_2, \ldots, m_n$ of the cut.
$N[1:n]$	is a list of pairs of indices indicating the lines delimiting each edge of the cut. $N[i]$ thus encodes the endpoints of the cut edge on l_i. The same convention as that above is used for missing endpoints.

Of course there is nothing categorical about these structures.[1] The same infor-

[1] In an actual implementation we need not store the left endpoints of segments in the horizon trees or the cut. This will save an additional $3n$ words of storage.

mation can be represented in many equivalent ways. For our example arrangement and cut in Fig. 2.1, the contents of our data structure would be as in Fig. 3.1.

E:	(a_1, b_1)	HTU:	$(-1, 2)$	HTL:	$(-1, 0)$
	(a_2, b_2)		$(-1, 5)$		$(-1, 1)$
	(a_3, b_3)		$(5, 4)$		$(5, 1)$
	(a_4, b_4)		$(5, 0)$		$(5, 3)$
	(a_5, b_5)		$(3, 0)$		$(3, 1)$
I:	4	M:	1	N:	$(-1, 2)$
	1		2		$(-1, 1)$
			5		$(5, 4)$
			3		$(5, 3)$
			4		$(3, 1)$

FIG. 3.1. The state of our structures for the arrangement and cut of Fig. 2.1.

We begin by describing how the upper and lower horizon trees can be constructed in $O(n)$ total time for the leftmost cut—under the assumption that the lines of H (i.e., the array E) have been given to us sorted in slope order. It is easy to see that, for the leftmost cut, the upper horizon tree consists of a segment on each line extending from left infinity till the first intersection with a line of larger slope is encountered.

This observation makes the construction of the upper horizon tree easy, if we insert the lines into our structure one at a time in order of decreasing slope. Assume that lines l_{i+1}, l_{i+2}, ..., l_n have already been inserted. These lines form an "upper bay" that l_i has to hit. See Fig. 3.2. We can compute where l_i hits this bay by traversing it in *counterclockwise* order. The advantage of doing this is that each edge we pass over ceases to be part of the bay, so it need never be looked at again. When we come to the edge that l_i hits, we simply have to break it into two parts and update the bay and HTU structures by inserting the appropriate segment of l_i into them. The linearity of this method is obvious.

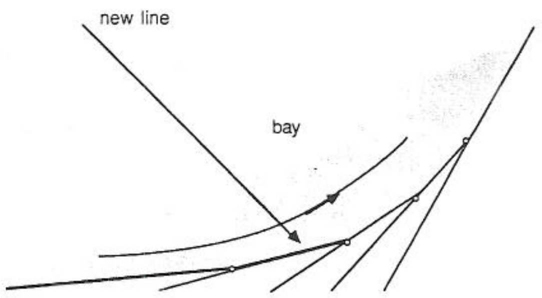


FIG. 3.2. Initialization of HTU.

Note that by combining the information contained in *HTU* and *HTL* for the leftmost cut we can easily obtain in linear time the structures *N* and *I*. The leftmost segment of each line—which is the appropriate edge for the leftmost cut—is the shorter of the segments of the line in *HTU* or *HTL*. Once *N* is known *I* can be trivially obtained. Thus initialization for all our structures is possible in $O(n)$ time.

How is an elementary step to be implemented using our structures? Suppose that we pop the stack *I* and get the index *i*. We know that c_i and c_{i+1} share a common right endpoint *V*, and therefore we can do an elementary step at *V*. Denote by σ the *s* segments *after* the elementary step. Let us first consider how *HTU* has to change; see Fig. 3.3. The change from s_{i+1}^+ to σ_i^+ is easy: the part of s_{i+1}^+ to the left of *V* is simply cut off. The change, however, from $s_i^+ = c_i$ to σ_{i+1}^+ requires a good deal of computation.

Just as in the initialization part, we have to compute where the extension of line m_i to the right hits the bay of the upper horizon tree delimited by c_{i+1} and c_{i+2}. And, as during initialization, we do this by traversing the bay starting at c_{i+2} and proceeding in a counterclockwise order, till an intersection of an edge of the bay with m_i is encountered. Note that m_i *must* hit this bay, since c_{i+2} is below c_i and therefore below m_i, yet the bay in question connects to σ_i^+ which is above the line m_i, as in Fig. 3.3. Once the proper intersection of m_i and the bay above is determined, updating the *HTU* can be done in $O(1)$ time. Thus *HTU* can be updated in a total cost of $O(n)$ per elementary step.

Since $c_i = s_i^+ \cap s_i^-$ (as follows from the remarks at the end of the previous section) the new *N* is easily obtained after the horizon trees are available. The same holds for *I*, while *M* can be trivially updated. Thus the overall cost in time for an elementary step is linear in the worst case (examples attaining this can be readily constructed).

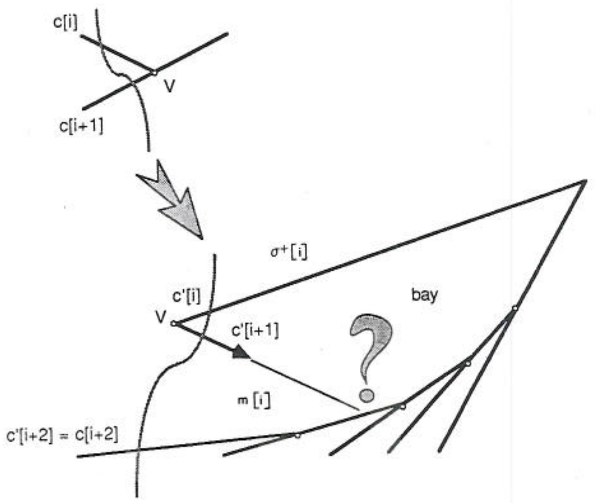


FIG. 3.3. Updating the upper horizon tree.

The structures for our example, after an elementary step at $i=4$ become as in Fig. 3.4.

E: same	I: 1	HTU: $(-1, 2)$	HTL: $(-1, 0)$	M: 1	N: $(-1, 2)$
		$(-1, 5)$	$(-1, 1)$	2	$(-1, 1)$
		$(4, 0)$	$(4, 1)$	5	$(4, 1)$
		$(3, 0)$	$(3, 1)$	4	$(3, 1)$
		$(3, 0)$	$(3, 1)$	3	$(3, 1)$

FIG. 3.4. The structures of Fig. 3.1, after an elementary step at 4.

What is the overall cost in time for pushing the leftmost cut all the way to the rightmost? Since there are $O(n^2)$ elementary steps and each step can cost $O(n)$, we get a total bound of $O(n^3)$, which is too large. We desire an $O(n^2)$ total bound—or an $O(1)$ bound per update, in the amortized sense.

To prove such a bound we do an amortized argument using the vertices of the horizon trees for our accounting. We only consider the upper horizon tree, the proof for the lower horizon tree being symmetrical. The vertices of the upper horizon tree are, by definition, the endpoints of the segments s_1^+ through s_n^+ as well as the "endpoints" of the artificial vertical segment which coincides with the dummy line at $x=+\infty$. For the counting argument it makes no difference whether a vertex is actually a finite point or whether it is the fictitious endpoint of an unbounded segment. The root of the tree whose nodes are these vertices is the fictitious upper endpoint of the dummy line, and its leaves are are the left endpoints of the segments s_i^+. We define the *depth* of a leaf to be the number of edges between itself and the root. The *external path length* of the tree is then defined as the sum of the depths of all leaves. Since the depth of a leaf cannot exceed n, the external path length is bounded above by n^2.

When we perform an elementary step we change s_{i+1}^+ to σ_i^+ and s_i^+ to σ_{i+1}^+. The transition from s_{i+1}^+ to σ_i^+ is not dramatic, as the depth of the left endpoint of σ_i^+ is only one less than that of the left endpoint of s_{i+1}^+. The depth of the left endpoint of σ_{i+1}^+ is at least that of s_i^+, and in fact it can be much longer (as, for example, in Fig. 3.3). But this will only be in favor of our argument. What is important is that the depth of any segment traversed when we search for the right endpoint of σ_{i+1}^+ increases by one. This is because the right endpoint of σ_{i+1}^+ lies on the path to the root of any such point. Again, there could be even more leaves whose depth increases by one—which works in our favor. Hence, the time spent in an elementary step is at most proportional to the increase in external path length caused by this step. The total amount of time to update the upper horizon tree is thus at most proportional to the difference is external path lengths between the initial and final trees. This difference is $O(n^2)$ since the external path length of the final tree is at most n^2 and that of the initial tree is non-negative.

THEOREM 3.1. *The total cost of updating HTU (or HTL) through all the elemen-*

tary steps is $O(n^2)$. Therefore the topological sweep can be carried out in $O(n^2)$ time and $O(n)$ working storage.

We remark that the same amortized bound will be obtained if we traverse each bay in the opposite direction. In the case of the upper horizon free, for example, we could start at the elementary step vertex V and then proceed clockwise around the bay, till the intersection of the bay with m_i is encountered—see Fig. 3.3. This requires different and more elaborate data structures, but the proof of the quadratic bound is comparably simple, as we now briefly explain. In the horizon tree, the number the bays associated with the cut as 1 to $n-1$, from top to bottom. Define the weight of an edge bounding a bay from below to be the number of the bay right above it. The weight of the whole horizon tree is then simply the sum of all the weights assigned to its edges. For the leftmost cut, the weight of the upper horizon tree is easily seen to be $O(n^2)$. At each elementary step, the traversed edges transfer to a bay numbered one less, except for the intersected edge that is split among the two bays. The cost of the step can then be accounted for by a fixed charge per step plus a decrease in the tree weight. We omit the details.

4. Coping with Degeneracies

This section proposes a method that eliminates all degenerate cases, such as parallel lines or multiple concurrent lines, thus relieving the programmer of the tedious task of coding these cases. Of course, we have to pay something for the elimination, and the price is carefully written primitive procedures that treat two parallel lines as non-parallel and three concurrent lines as non-concurrent. This entails the occurrence of zero-length edges and vertices at infinity in the arrangement. It is crucial for this method that this simulation of non-degenerate cases be done in a consistent way. For a more complete description of this idea see [E, Chap. 9; EM], where implementation issues are discussed.

The primitive procedures use the indices 1 through n assigned to the lines for their computations. Let line l_i be given by the equation

$$a_i x + b_i y + c_i = 0,$$

for $1 \leqslant i \leqslant n$ and $(a_i, b_i) \neq (0, 0)$. We define another line

$$l_i(\varepsilon): a'_i x + b'_i y + c'_i = 0,$$

with $a'_i = a_i + \varepsilon^{2^{3i}}$, $b'_i = b_i + \varepsilon^{2^{3i-1}}$, and $c'_i = c_i + \varepsilon^{2^{3i-2}}$, for $\varepsilon > 0$ small enough. All decisions, like whether or not l_i intersects l_j above l_k etc., are based on $l_i(\varepsilon)$ instead of on l_i. Consequently, the computation simulates the sweep of arrangement $\mathscr{A}(H(\varepsilon))$, with $H(\varepsilon) = \{l_i(\varepsilon) \mid 1 \leqslant i \leqslant n\}$ and $\varepsilon > 0$ but small enough. It is not hard to prove that $H(\varepsilon)$ contains no two parallel and no three concurrent lines.

All manipulations of coordinates can be reduced to determining the signs of determinants of the form

$$\det \begin{pmatrix} a'_i & b'_i & c'_i \\ a'_j & b'_j & c'_j \\ a'_k & b'_k & c'_k \end{pmatrix},$$

which can be done without specification of any particular value for ε. To this end, however, all multiplications must be performed without any loss in precision; equivalently, we need to compute the power series expansion in ε of the determinant above until we encounter the first non-zero term.

5. Applications

We expect that the idea of topologically sweeping a geometric scene (instead of sweeping it with a straight line) will have numerous applications in computational geometry; for instance, Nievergelt and Preparata [NP] have used a similar idea for intersecting two planar convex maps. It appears that the difficulty of applying the idea successfully to problems other than sweeping arrangements of lines is the design of efficient supporting data structures. In this section we address problems that can be formulated in terms of arrangements, or that relate to such problems by some geometric transformation; in all cases we are able to obtain an improvement over the previously known space or time bounds.

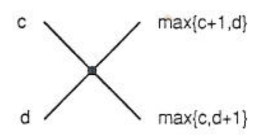
5.1. *Convex Subsets of Configurations and Paths in Arrangements*

Let H be a set of n lines in general position in E^2. A *monotone path* π *of* $\mathscr{A}(H)$ is a connected subset of alternating edges and vertices of $\mathscr{A}(H)$ such that every vertical line intersects π in exactly one point. Therefore π is unbounded. A vertex p of π is a *turn* if the two incident edges are not collinear. We define the *length* of π as the number of turns plus one.

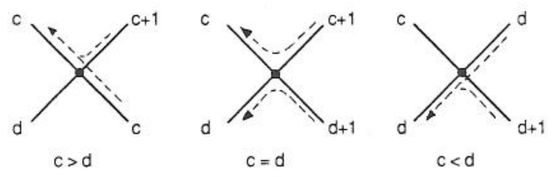
Problem 5.1.1. Compute a longest monotone path of $\mathscr{A}(H)$.

Sharir [Sh] has shown that there are arrangements of n lines with monotone paths of length $\Omega(n\sqrt{n})$; no non-trivial upper bound is currently known. To compute a longest path, we sweep $\mathscr{A}(H)$ topologically and for each edge e in the current cut we maintain a longest path which extends from e towards the left: the edge e holds the number of turns of this path and a pointer to its predecessor edge on this path. The rules for maintaining this information are illustrated in Fig. 5.1.

Theorem 5.1.1. *The length of the longest monotone path in an arrangement of n lines in E^2 can be found in $O(n^2)$ time and $O(n)$ storage.*



(a) c and d are the counts of the predecessor edges



(b) the backward pointers

FIG. 5.1. Rules for updating longest monotone paths.

It is interesting that the topological sweep does not maintain enough information to allow us to extract the actual longest path directly, for we cannot afford to keep around the predecessor pointers for all edges of the arrangement and still have linear storage. Without predecessor pointers, we can still backtrack by running a complete sweep up to each desired edge. This clearly is a time-consuming process, so what we do instead is to save various snapshots of the data structures used by the algorithm at certain moments; in this way we can avoid having to rerun the algorithm from the beginning for each step of backing up we need to do.

The specific method that we use can be formulated in terms of the problem of backing up from a state t of the sweep to an earlier state s in the linear ordering of all states visited by the algorithm. We identify each state with its rank in this ordering. Initially we have saved states s and t; we now wish to deduce the sequence of nodes defining the longest path between s and t. Note that if we have saved a state of the sweep, then we can easily extract the current (last) node of the longest path in this state. Let m denote the state halfway between s and t. In order to back up from t so s we proceed as follows: first go forward from s to m and save state m; then recursively back up from t to m; then output the current node on the longest path in snapshot m; and finally recursively back up from m to s. To find the longest path we apply this recursive backing up to the initial and final states of the sweep.

The storage used by this method is the maximal number of buffers needed to hold states at any one time. The recursive structure of the algorithm makes it clear that we need $O(\log n)$ buffers: this is the depth of the recursion stack and each invocation needs one buffer to store the halfway state. So the total space used by this method is $O(n \log n)$—assuming that the edges of the path are output as they are discovered so that we do not have to store them. The time for the whole procedure $O(n^2 \log n)$, as follows from a standard divide-and-conquer recurrence. It is possible to analyze the time cost of this backtracking method for any given number of buffers. For example, if we are willing to use (n^e) buffers $(O(n^{1+e})$ total storage), then we can do the backtracking in $O(n^2)$ time. This analysis turns out to be isomorphic to that of certain recurrences on trees considered by Bentley and Saxe [BS, p. 331].

THEOREM 5.1.1A. *The longest monotone path in an arrangement of n lines in* E^2 *can be computed in* $O(n^2 \log n)$ *time and* $O(n \log n + k)$ *storage, where k is the length of that path.*

A monotone path π in $\mathscr{A}(H)$ is called *concave* if each turn of π is a left turn when traversed by a particle moving from left to right.

PROBLEM 5.1.2. Compute a longest monotone concave path in $\mathscr{A}(H)$.

To solve Problem 5.1.2 algorithmically, we take the same approach as for Problem 5.1.1; we adjust only the rules for maintaining longest paths (see Fig. 5.2). Even though a concave (or convex) path has length at most $O(n)$, the previous difficulties with storage arise again and can be removed again at the expense of a $\log n$ factor in time and storage.

By the duality discussed in [CGL], Problem 5.1.2 corresponds to a problem studied in combinatorial geometry. Let S be a set of points in E^2 which is dual to H so that the slopes of the lines determine the x-coordinate of the points (i.e., map line $y = ax + b$ into point $(a, -b)$). Let $T = \{(a_1, b_1), \ldots, (a_k, b_k)\}$ be a subset of S such that $a_i < a_{i+1}$. We say T is a *concave* chain of length k if for any two consecutive points (a_i, b_i) and (a_{i+1}, b_{i+1}) all other points lie in the upper halfplane of the line joining these two points, and T is a *convex* chain if for any two consecutive points all other points lie in the lower halfplane through these two points.

PROBLEM 5.1.2A. Compute a longest concave (or convex) chain of S.

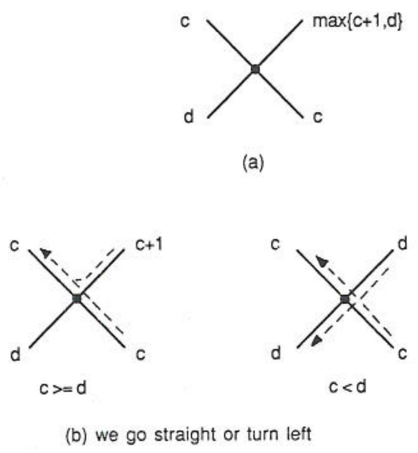


FIG. 5.2. Rules for updating longest concave paths.

Erdös and Szekeres [ES1, ES2] have shown that every non-degenerate set of at least $\binom{2k-4}{k-2}+1$ points in E^2 has a convex or concave chain of length k, but this is not true for $\binom{2k-4}{k-2}$ points. By duality, the points in a convex (or concave) chain correspond to the lines that support edges of a monotone concave (or convex) path in the dual arrangement.

THEOREM 5.1.2. *The length of the longest concave (or convex) chain in a set of n points in E^2 can be found in $O(n^2)$ time and $O(n)$ storage. To extract the actual path an extra* log *n factor must be paid in both time and storage. Alternatively, we can extract the actual path by paying an extra factor of $O(n)$ in time while keeping the storage linear.*

Building on this result, we attack a related problem. If S denotes a non-degenerate set of n points in E^2, we call a subset T of S *convex* if each point of T appears as a vertex of the convex hull of T. Again Erdös and Szekeres [ES1, ES2] have demonstrated the existence of 2^{k-2} points in E^2 without a convex subset of cardinality k. They conjecture that every non-degenerate set of $2^{k-2}+1$ points contains such a convex subset.

PROBLEM 5.1.3. Compute a largest convex subset of S.

Suppose that p is the topmost point of a convex subset T of S. Draw the horizontal line through p and remove all points above this line. Now move this line upward to infinity using a projective transformation. This projective transformation maps p into a point with infinite y coordinate and $T—\{p\}$ into a concave chain. Conversely, if subset U maps into a concave chain then $U \cup \{p\}$ is a convex subset of S with topmost point p. To solve Problem 5.1.3 we thus solve n instances of Problem 5.1.2A. To extract the actual largest convex subset, we use the brute-force backtracking strategy, but only for the "winning" leftmost point p. This backtracking takes $O(n^3)$ time, since the largest convex subet consists of at most n points. This improves the storage bound in the result of Chvatàl and Klincek [CK] who gave an algorithm that runs in $O(n^3)$ time and $O(n^2)$ storage.

THEOREM 5.1.3. *The largest convex subset of a set of n points in E^2 can be found in $O(n^3)$ time and $O(n)$ storage.*

A convex subset T of S is called *empty* if no point of S belongs to the interior of the convex hull of T. By a result of Harborth [Ha], every 10 points (no three collinear) in E^2 have an empty convex subset of size 5, and by Horton [Ho] there are sets with arbitrarily many points but without an empty convex subset of size 7. Using an $O(n^2)$ time and $O(n^2)$ storage algorithm, Avis and Rappaport [AR] found a set of 20 points without any empty convex hexagon. The current record holders are Fabella and O'Rourke [FO] who used other programs to find a set of 22 points without an empty convex hexagon.

PROBLEM 5.1.4. Compute a largest empty convex subset of S.

Let T be an empty convex subset of S with leftmost point p. By the same projective transformation as that used above, p is mapped into a point with infinite y coordinate and T is mapped into an *empty* concave chain, that is, the image of no point in S lies vertically above any edge of the chain. So we try to compute a longest empty concave chain of S. An edge $e=(p, q)$ is *forbidden* if there is a point r vertically above e. In dual space p, q, and r correspond to three lines p', q', r' such that p' and q' intersect above r' and the slope of r' lies between the slopes of p' and q'. To compute a longest empty concave chain, we use the same algorithm as for Problem 5.1.3 with one modification: a turn (in the dual arrangement) is taken only if it is allowed, that is, if it does not correspond to a forbidden edge in S. Figure 5.2A shows a forbidden turn and illustrates how we can efficiently distinguish between forbidden and allowed turns: for each line h remember the steepest line less steep than h whose intersection with h was processed—call it $f(h)$. A left turn from line g to line h is now forbidden if and only if $f(h)$ is steeper than g.

Maintaining this extra information for each line costs constant time per edge, which implies.

THEOREM 5.1.4. *The largest empty convex subset of a set of n points in* E^2 *can be found in* $O(n^3)$ *time and* $O(n)$ *storage.*

5.2. *Stabbing line segments*

Let S be a set of n closed and bounded line segments in E^2, not necessarily disjoint. For clarity and convenience, we assume that the $2n$ endpoints are in general position. We consider two stabbing problems for S:

PROBLEM 5.2.1. Find a line that cuts the maximal number of segments in S.

PROBLEM 5.2.2. Find a line that cuts no segment and such that the absolute value of the numbers of segments above the line minus the number of segments below is a minimum.

In dual space a segment (with two endpoints) corresponds to a pair of lines; a line cuts the segment if its dual point lies in a specified double wedge of the two lines. (If the mapping above is used, then the point belongs to the double wedge which avoids the vertical line through the intersection of the two lines; see Fig. 5.3.)

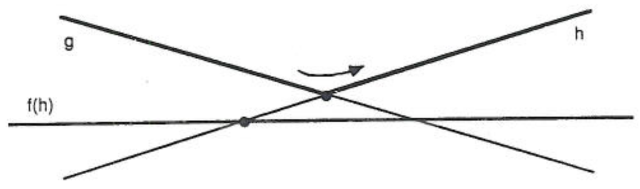


Fig. 5.2A. Forbidden turn from g to h.

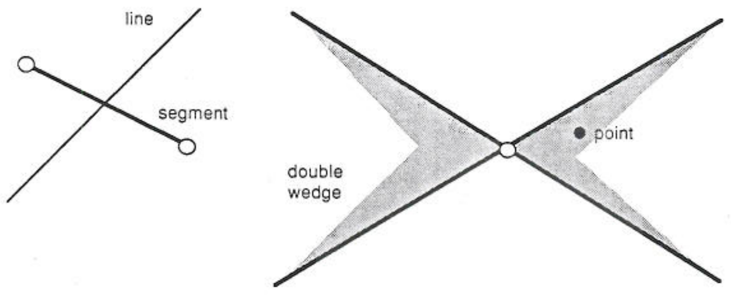


FIG. 5.3. The dual of a line segment.

Let H be the set of $2n$ lines dual to the endpoints of the segments in S. If the dual points of two lines l_1 and l_2 fall into the same region of arrangement $\mathscr{A}(H)$, then l_1 and l_2 intersect the same segments, and therefore the same number of segments. To solve Problem 5.2.1 we compute for each region of $\mathscr{A}(H)$ the number of segments cut by a line dual to a point of the region. This piece of information is best computed when the region is first encountered during a topological sweep (see Fig. 5.4 for the five cases that occur). Each manipulation can be carried out in constant time. To get the algorithm started, we compute the count for each region cut by the initial topological line. Even if we do this using a trivial method, this will take only $O(n)$ time per region and thus $O(n^2)$ time altogether. Therefore we can improve the results obtained in [EOW] by a factor of n in storage.

THEOREM 5.2.1. *A line that cuts the maximum number of n given line segments in R^2 can be found in $O(n^2)$ time and $O(n)$ storage.*

To solve Problem 5.2.2, we compute for each region the number of segments in S which lie above a corresponding line and the number which are cut by this line. Again, this information can be propagated in constant time from one region to the next during the topological sweep.

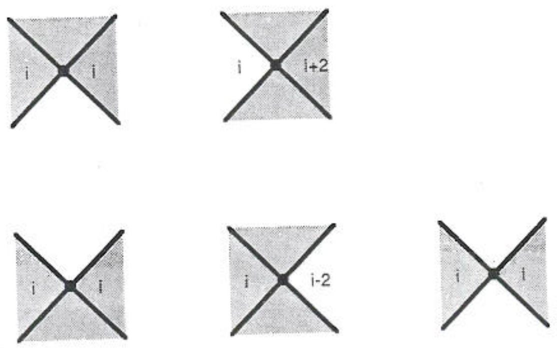


FIG. 5.4. The counts shown give the number of segments cut by a line whose duals falls in that region. The last case corresponds to "passing to the other side of the plane."

THEOREM 5.2.2. *A line that avoids all of a given set of n segments in E^2 and produces a best balance between the numbers of segments on either side can be found in $O(n^2)$ time and $O(n)$ storage.*

It is interesting to note that the methods above do not work for the following seemingly related problem originating in Lee and Preparata [LP]: For n segments in E^2 find a direction (if it exists) such that no two shadows intersect if light is shed from this direction. Sweeping the dual arrangement with a topological line, instead of a straight one, does not seem to be an adequate substitute in this case.

5.3 *Visibility Problems for Non-intersecting Line Segments*

For solving the problem in this section we make use of the fact that the topological sweep visits the vertices of any fixed line in the left to right order. In dual space this means that we scan the points around any fixed point p in the order they are met by a rotating line anchored at p. This order is not exactly the sorted order of points around p since the line extends from p to two sides. Nevertheless, we can "unmerge" and concatenate these sublists to get the points sorted around p in $O(n)$ steps. After this introductory remark we are ready to state and solve the segment visibility problem.

Let S be a set of n relatively open, bounded, and pairwise non-intersecting segments in E^2, and P the set of their $2n$ endpoints. We define the *visibility graph* V_s as follows:

(i) the endpoints of the segments are the nodes of V_s, and

(ii) for endpoints v, w the undirected edge $\{v, w\}$ is an edge of V_s if the straight segment connecting v and w avoids all segments in S (see Fig. 5.5).

Visibility graphs have been used for computing shortest paths between points that avoid all segments in S [L, AA]: the single-source shortest path algorithm of Dijkstra [AHU] gives a method for finding such a path between two arbitrary points in time proportional to the number of edges in V_s (see also [FT]). To construct V_s, we follow the approach of Welzl [W], which can be described intuitively as follows.

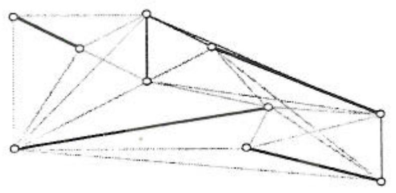


FIG. 5.5. The visibility graph for five segments.

Imagine that each point p in P is equipped with a ray $r(p)$ rooted at p and rotating around p counterclockwise through 180°, from pointing down to pointing up. At each point in time, p stores the segments $s(p)$ that intersects $r(p)$ closest to p. When $r(p)$ sweeps over a point q in P then $s(p)$ may possibly change. (If we assume that no collinearities are present among the points in P, then four cases can occur; see Fig. 5.6.)

Algorithmically, it is straightforward to distinguish the four cases (given p, q, $s(p)$, and $s(q)$) and to make the necessary changes in constant time, given that $s(p)$ and $s(q)$ are correct when the ray $r(p)$ reaches q. We are left with a scheduling problem: how to schedule the crossings of all $2n$ rays over all $2n$ points in such a way that $s(p)$ and $s(q)$ are correct when the crossing of $r(p)$ over q is processed. Fortunately, if we look at the dual, the consistency requirement becomes simply the left-to-right rule satisfied by the topological sweep.

We write $p \to q$ for the event that $r(p)$ crosses over q. Thus, the algorithm processes a sequence $(e_1, ..., e_m)$ of events, with $m \leqslant \binom{2n}{2}$. Let x be the last point encountered by $r(q)$ just before it becomes parallel to the line through p and q, and let y be the first one encountered after that. If $e_i = [q \to x]$ is scheduled before $e_j = [p \to q]$ and this is scheduled before $e_k = [q \to y]$ (that is, if $i < j < k$) then $s(q)$ will have the correct value when $p \to q$ is processed.

Let now H be the set of $2n$ lines which are dual to the points in P. An event $p \to q$ corresponds to the intersection of the lines that correspond to p and q. For a point p let $(l_1, ..., l_{2n-1})$ be the sequence of lines connecting p with other points such that the slope of l_i is smaller than that of l_{i+1}, for $1 \leqslant i \leqslant 2n-2$. This sequence corresponds to the sequence of intersections of the line dual to p with the other $2n-1$ lines in H, sorted from left to right. The restriction above on schedules of events thus translates to:

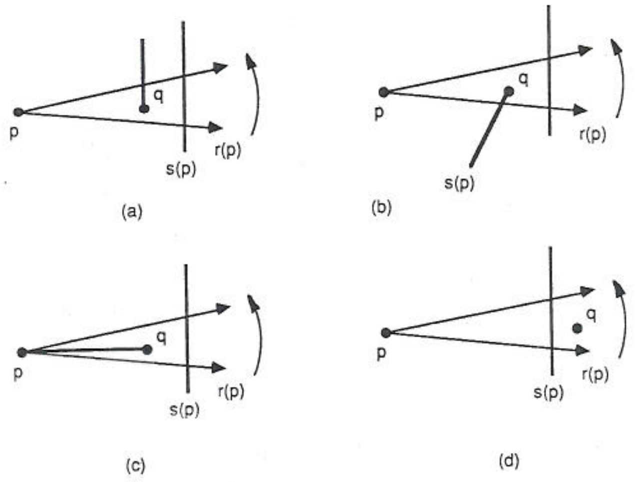


FIG. 5.6. First segment $s(p)$ changing over time.

If v and w are two vertices of $\mathscr{A}(H)$ which lie on a common line of H and v is further left than w, then the event defined by v is to be processed before the one defined by w.

Let G be the directed graph with the vertices of $\mathscr{A}(H)$ as nodes and an arc from v to w if v and w are endpoints of a common edge in $\mathscr{A}(H)$ and v is further left than w. Any topologically sorted sequence of G's nodes (see [Kn]) gives a sequence of events that allows us to process a single event in constant time, as illustrated above; the sweep algorithm of Section 3 provides exactly such a sequence. The theorem below imporves the result of Welzl and Asano *et al.* [W; AA] as far as the amount of storage is concerned: their methods need quadratic storage since they construct $\mathscr{A}(H)$ explicitly.

THEOREM 5.3.1. *The visibility graph of a set of n non-intersecting segments in* E^2 *can be constructed in* $O(n^2)$ *time and* $O(n)$ *storage (not including the storage needed for the edges of the graph).*

We now turn to a problem that can be solved by methods similar to the ones used to construct the visibility graph V_s:

PROBLEM 5.3.2. Identify the segments of S that are hidden from another segment s_0.

Formally, we say that a segment s is *hidden from* s_0 if there is no relatively open line segment which avoids S and has its endpoints on the closures of s and s_0. For an endpoint p of any segment in S and every angle α, $0 \leqslant \alpha < 2\pi$, we let $r_\alpha(p)$ be the ray that starts at p and forms an angle α with the x-axis. Then we define $s_\alpha(p)$ as the segment in $S \cup \{s_0\}$ which intersects $r_\alpha(p)$ closest to p (if such a segment exists).

LEMMA 5.3.2. *Segment s of S is not hidden from* s_0 *if and only if* s_0 *is visible from an endpoint of s or there is an endpoint p of another segment in S and angle* α *such that* $s = s_\alpha(p)$ *and* $s_0 = s_{-\alpha}(p)$.

sa.

Proof. Omitted. ∎

For each endpoint p of any segment in S, the algorithm maintains two rays leaving p in opposite directions together with the segments hit first by these rays. If s_0 is one of these segments then the other one is not hidden from s_0. All details in the maintenance of this information are as in the construction of the visibility graph. This process in fact lets us compute the visible pieces from s_0 of each segment.

THEOREM 5.3.3. *The segents of S hidden from segment* s_0 *can be identified in* $O(n^2)$ *time and* $O(n)$ *storage.*

5.4. *Minimum Area Triangles*

Let S be a set of n points in E^2. Any three points p, q, r of S define a triangle $t_{p,q,r}$ with area $A_{p,q,r}$. We consider

PROBLEM 5.4.1. Determine points p, q, r of S such that $A_{p,q,r}$ is minimum.

If points p and q are fixed then r is a point closest to the line l through p and q. It follows that l can be moved continuously into a position where it contains r without passing through any point of S. In the dual arrangement the line l containing p and q corresponds to a vertex v and point r corresponds to a line that bounds a region with v on its boundary. Following [CGL, EOS], we propose the following algorihtm: for each region of the dual arrangement, test all vertex-edge pairs on the boundary, that is, compute the corresponding triangle and record it if its area is the current minimum. As shown in [CGL, EOS], $O(n^2)$ triangles are tested. In terms of the sweep algorithm in Section 3, a region is examined when it is entered; at this point, the edges and vertices of the region can be derived in constant time each from the two horizon trees. This follows from the representation of the horizon trees and the remarks at the end of Section 2.

THEOREM 5.4.1. *The minimum area triangle defined by n points in E^2 can be determined in $O(n^2)$ time and $O(n)$ storage.*

This improves the $O(n^2)$ time and storage algorithms of [CGL, EOS] and the $O(n^2 \log n)$ time and $O(n)$ storage algorithm of [EW]. Note that the area of the determined triangle vanishes if S contains three collinear points. A more general approach to finding degeneracies in point sets follows in subsection 5.6.

5.5. *Visiting Faces in d-Dimensional Arrangements*

Even though we do not know how to generalize the topological sweep directly to higher dimensions, a number of problems in E^d can be attacked by using only the planar methods we have developed. In this subsection we look at the problem of listing all faces of various types in a d-dimensional arrangement. The methods we describe will be used in later sections to solve other geometrical problems.

Let H be a set of n hyperplanes in E^d. For convenience, we assume that H is non-degenerate, that is, each i-face of arrangement $\mathscr{A}(H)$ belongs to exactly $d-i$ hyperplanes and any j hyperplanes intersect in a $(d-j)$-flat (i.e., a $(d-j)$-dimensional linear subspace). It is also convenient to assume that H contains no hyperplanes parallel to any coordinate axis x_d. In particular, no hyperplane is parallel to the last coordinate axis x_d, which we visualize as being "vertical." Many problems in computational geometry, including those in the sections below, can be solved efficiently by visiting all faces of various dimensions in $\mathscr{A}(H)$ and computing some piece of information for each face. We will show how to use topological sweeps of two-dimensional arrangements for efficiently computing this information for each cell (i.e., d-face) of a d-dimensional arrangement. If i-faces $(2 \leqslant i \leqslant d)$ are to be visited, then we ay use the same method applied to all i-flats determined by the hyperplanes. Vertices and edges can be visited by sweeping all planes (i.e., 2-flats) in the arrangement.

To visit all cells of $\mathscr{A}(H)$, we sweep all two-dimensional subarrangements and make use of a correspondence between cells and vertices of $\mathscr{A}(H)$. For ξ a cell in $\mathscr{A}(H)$, define $v(\xi)$, the *canonical vertex* of ξ to be the vertex v of the boundary of ξ that has the smallest x_d coordinate among all points of ξ. Conversely, for a vertex v we define the *canonical cell* $c(v)$ so that $v(c(v)) = v$. This is possible since every vertex is the bottommost vertex of exactly one cell, as by assumption the arrangement is non-degenerate.

There is one unpleasant property of this association. It is clear that cells which are unbounded from below have no canonical vertex. We can deal with this problem by carrying out $2d$ runs of the sweep $R_1, ..., R_{2d}$, where the positive (negative) x_i-axis is treated as the positive x_d-axis in run $R_{2i-1}(R_{2i})$, thus assuring that all cells will be reached.

Suppose that we sweep a two-dimensional subarrangement in plane g which is the intersection of $d-2$ hyperplanes in H. Let v and w be two vertices in this arrangement incident to a common edge; then v and w belong to $d-1$ common hyperplanes (i.e., $(d-1)$-flats). Let h_v and h_w be the hyperplanes not common to v and w, but which contain v and w, respectively. Assume that v already stores the necessary information $I(v)$ for $c(v)$ (where I is application-dependent) and also that I can be updated in constant time if we cross any hyperplane and thus enter a new cell. Then the information $I(w)$ for $c(w)$ can be computed in constant time from $I(v)$, h_v, h_w, v, and w. As an example let $I(v)$ denote the number of hyperplanes below $c(v)$. Then $I(w)$ can differ from $I(v)$ by at most 2, since h_v and h_w are the only two hyperplanes that can separate $c(v)$ and $c(w)$. A simple example illustrating the various possibilities in the plane is shown in Fig. 5.7.

To initialize the sweep of g, we might compute the informaton for each vertex of the initial cut using a trivial method. If we assume that a single step in the sweep costs constant time and that $O(n)$ time is needed to compute I per initial vertex, then an entire sweep costs $O(n^2)$ time. The hyperplanes in H determine $\binom{n}{d-2} = O(n^{d-2})$ planes, which amounts to $O(n^d)$ time altogether. Thus it is posible to visit all faces of the arrangement $\mathscr{A}(H)$ in time proportional to its size. It is important to notice that this strategy works for all functions I that satisfy the computability requirements stated above. Specific examples of such functions are discussed in Sections 5.7, 5.8, and 5.9.

5.6. *Degeneracies in Configurations*

Geometrical algorithms often become complex when they have to deal with degenerate situations—the topological sweep is no exception. As we will see in this subsection, however, we can use the techniques we have developed to detect and even enumerate the degeneracies present in a configuration of points. This might be useful knowledge before our point set is processed by other algorithms. The relation

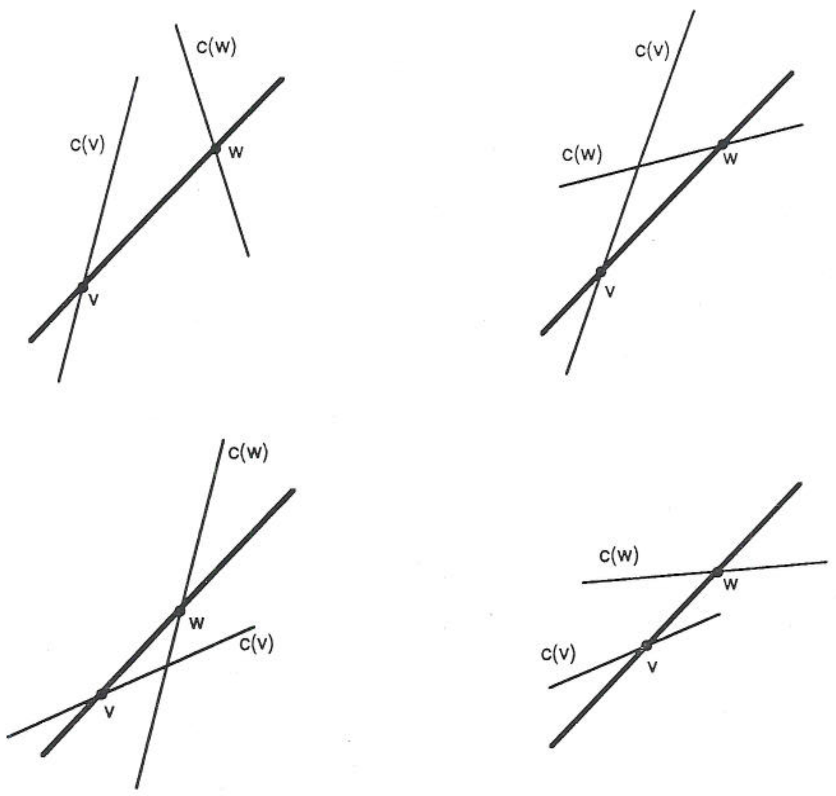


FIG. 5.7. Illustration of the four cases in the plane.

between this section and Section 4 is obvious: the objective in this section is to discover degeneracies, while that in Section 4 to "hide" them. We start with some definitions.

A set of $i+2$ points in E^d, $0 \leqslant i < d$, is called an *i-degeneracy* if it belongs to an i-flat but to no $(i-1)$-flat. A set S of n points in E^d, $d \geqslant 2$, is *degenerate* if for some i, $0 \leqslant i < d$, it contains an i-degeneracy.

PROBLEM 5.6.1. Decide whether or not S is degenerate.

To detect i-degeneracies for $i \leqslant d-2$, we simply test all subsets of size $i+2$ in $O(n^d)$ time; so assume that S contains no degeneracies other than possibly of dimension $d-1$. Assume also that no d points of S lie on a common vertical hyperplane (we can perform $d-1$ extra runs $R_1, \ldots, R_{d-1}$ of our algorithm, with coordinates x_d and x_j exchanged in run R_j, to guarantee that the common hyperplane is non-vertical at least once). If $d+1$ points of S belong to a common hyperplane (i.e., they are a $(d-1)$-degeneracy) then the corresponding $d+1$ hyperplanes in the dual arrangement (point $(p_1, \ldots, p_d)$ is mapped to hyperplane $x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} - p_d$) meet in a common point v. Let g be a plane obtained by intersecting $d-2$ of these hyperplanes. Then v appears as the intersection of at least three lines in the two-dimensional subarrangement in g. It is worthwhile to note that the technique of Section 4 which simulates non-degeneracy during a sweep can still be applied. To detect degeneracies, however, the technique must be accompanied by a test which checks for edges of length zero.

THEOREM 5.6.1. *In $O(n^d)$ time and $O(n)$ storage we can decide whether or not n points in E^d are degenerate, for $d \geq 2$.*

We can test for points lying on spheres by testing for planar degeneracies among certain transformed points. Call a set of $i+2$ points in E^d an *i-cosphericality* if these points are equidistant from a common point and belong to a common i-flat but to no $(i-1)$-flat, for $0 \leq i \leq d$. We can use the algorithm above in order to decide whether or not a set S of n points in E^d contains a cospherical subset: For point $p=(p_1, ..., p_d)$, define point $p'=(p_1, ..., p_d, p_1^2+\cdots+p_d^2)$ in E^{d+1}. A subset of $i+2$ points in S is an i-cosphericality if and only if the corresponding $i+2$ points in E^{d+1} are an i-degeneracy; see Guibas and Stolfi [GS] for more details on this lifting map.

We now address briefly the problem of reporting all degeneracies present in S. A subset T of S is *i-degenerate* if every $i+2$ points in T define an i-degeneracy; it is *proper* if there is no point p with $T-\{p\}$ $(i-1)$-degenerate; and it is *maximal* if it is not contained in another i-degenerate subset of S. The proper and maximal degenerate subsets of S imply in a trivial way all others. Furthermore the number of such subsets can be only $O(n^d)$ (see [E]), so there is hope of reporting them efficiently.

PROBLEM 5.6.2. For $0 \leq i < d$, enumerate all proper and maximal i-degenerate subsets of S.

Edelsbrunner [E] shows how to accomplish this in $O(n^d)$ time and space. We can decrease the storage cost to $O(n+k)$, where k is the total size of reported proper and maximal degenerate subsets, by using the techniques of this paper.

For example, when $d=2$, we are looking to report all lines containing at least three points of S. In the dual arrangement this corresonds to reporting all vertices where three or more lines are concurrent. We can find those by modifying the algorithm of Section 3 so that the stack I contains as entries ranges of indices, where range $[i, j]$ indicates that $(c_i, ..., c_j)$ have a common right endpoint. In the representation of *HTU* or *HTL* our convention will be that we record the highest slope line terminating a tree segment s_i^+ or s_i^-. This corresponds to perturbing the highest slope line by moving it parallel to itself a small distance to its left, then perturbing the next line in slope by moving it to its left by a much smaller distance, etc. An elementary step sweeping over such a multiple vertex $[i, j]$ is now easy to handle. In updating *HTU* we just find the intersection of the bay from s_j^+ to s_{j+1}^+ by propagating each of the lines supporting $c_i, ..., c_{j-1}$ in turn. In each case we can start the search from where the previous intersection was detected. We omit here the details of this method and its counterpart in higher dimensions; our plan is to report on it elsewhere.

5.7. *Computing Ranks of Points*

Let S be a non-degenerate set of n points in E^d, $d \geqslant 2$. For a point p in S and a halfspace U that contains p, we define $r(p, U)$ as the cardinality of $(S - \{p\}) \cap U$. Then $\rho(p) = \min\{r(p, U) \mid U$ is a halfspace that contains $p\}$ is called the *rank* of p. Applications of ranks of points can be found in statistics and other fields.

We now translate the notion of a rank into dual space. Let H be the set of dual hyperplanes, and let $h \in H$ correspond to point p in S. Consider the $(d-1)$-dimensional subarrangement of $\mathscr{A}(H)$ in h. For each facet (i.e., $(d-1)$-face) f of $\mathscr{A}(H)$ in h define

$$a(f) = |\{h \in H \mid f \in h^-\}|,$$

$$b(f) = |\{h \in H \mid f \in h^+\}|.$$

By definition of rank, we have $\rho(p) = \min\{a(f), b(f) \mid f$ is a facet of $\mathscr{A}(H)$ contained in $h\}$. The results in Section 5.5 now imply:

THEOREM 5.7.1. *In $O(n^d)$ time and $O(n)$ storage we can compute the ranks for each one of a set of n points in E^d.*

5.8. *Best assignment for vectors in E^d*

Let $V = \{v_1, ..., v_n\}$ be a set of n non-zero vectors in E^d, and let A be the set of all ordered n-tuples $(\alpha_1, ..., \alpha_n)$, termed *assignments*, with $\alpha_i \in \{+1, -1\}$ for $1 \leqslant i \leqslant n$. For an assignment $\alpha = (\alpha_1, ..., \alpha_n)$, we define

$$s(\alpha) = \sum_{i=1}^{n} \alpha_i v_i,$$

and let $|s(\alpha)|$ be the (Euclidean) length of vector $s(\alpha)$.

PROBLEM 5.8.1. Given V, a set of n non-zero vectors in E^d, determine an assignment α for V with $|s(\alpha)|$ maximum.

By reduction to arangements in E^{d-1}, we can show that out of the 2^n assignments only $O(n^{d-1})$ need to be considered. For vector v_i in V let h_i denote the hyperplane through the origin with normal v_i, and let h_i^+ and h_i^- denote the two (open) halfspaces bounded by h_i, of which h_i^+ is the one into which v_i points. Hyperplanes h_i, for $1 \leqslant i \leqslant n$, cut E^d into various cones all with apex at the origin. Let now $\mu = (\mu_1, ..., \mu_n)$ be an optimal assignment for V (i.e., $|s(\mu)|$ is maximal), and let C denote the cone that contains the endpoint $p = p(\mu)$ of $s(\mu)$.

LEMMA 5.8.1. *For $1 \leqslant i \leqslant n$, $p \in h_i^+$ if $\mu_i = +1$ and $p \in h_i^-$ if $\mu_i = -1$.*

Proof. Assume the existence of an index j such that p belongs to h_j^+ but

$\mu_j = -1$. Define $q = p + 2v_j$. Note that q is the endpoint of $s(\mu')$, where $\mu' = (\mu_1, ..., \mu_{j-1}, -\mu_j, \mu_{j+1}, ..., \mu_n)$. Since $p \in h_j^+$, we have a fortiori that $q \in h_j^+$. Now q is further from the origin than p since p lies in the positive halfspace h_j^+ and $q = p + 2v_j$. This contradicts the optimality of μ. A similar argument holds in the other case. ▌

It follows that only one assignment $\alpha_c = (\alpha_1, ..., \alpha_n)$ needs to be checked for each cone C, namely the one with $\alpha_i = +1$ if $C \subseteq h_i^+$ and $\alpha_i = -1$ if $C \subseteq h_i^-$. Furthermore, if $s(\alpha_c)$ is known and D is a cone separated from C only by hyperplane h_j so that, say, $C \subseteq h_j^+$ and $D \subseteq h_j^-$ then $s(\alpha_D) = s(\alpha_C) - 2v_j$. Therefore $s(\alpha_D)$ can be computed in constant time from $s(\alpha_C)$. Finally note that only one of each pair of two opposite cones needs to be considered and that a hyperplane that avoids the origin cuts each pair of opposite cones in a bounded $(d-1)$-face or two unbounded $(d-1)$-faces of an arrangement of n $(d-2)$-flats in h. Sweeping all two-dimensional subarrangements of this arrangement and inspecting all cells yields the theorem below; the $n \log n$ term is there to handle the sorting needed when $d = 2$.

THEOREM 5.8.2. *For n vectors in E^d, we can find in $O(n^{d-1} + n \log n)$ time and $O(n)$ storage an assignment α with $|s(\alpha)|$ maximum.*

It is interesting to note that the choice of assignments checked does not depend on the lengths of the vectors at all. It is not hard to see that exactly the vectors that give rise to vertices of the convex hull of $\{s(\alpha) | \alpha \in A\}$ are checked (this convex hull forms a *zonotope* about which more can be found in [E]). Consequently, the algorithm works even if $|s(\alpha)|$ no longer denotes the Euclidean length of $s(\alpha)$ but some other norm.

5.9. *Extremal Shadows of Convex Polytopes*

Let P be a fixed convex polytope in E^d; that is, P is the convex hull of some finite set of n points in a d-dimensional Euclidean space with $d \geqslant 3$. For x a non-zero vector, the orthogonal projection of P onto the hyperplane h through the origin with normal vector x is called P's *shadow* $S(x)$ *from direction* x. We define $\mu(x)$ as the $(d-1)$-dimensional measure of $S(x)$ (in E^3, $\mu(x)$ is the area of the two-dimensional shadow). Obviously, $\mu(x) = \mu(-x)$.

For each facet f of P, let v_f be the outward directed normal vector with length equal to the $(d-1)$-dimensional measure of f, and let h_f be the hyperplane through the origin with normal vector v_f. These hyperplanes cut E^d into cones with the origin as apex. Each cone can be thought of as a collection of directions of projection for which the "set of visible facets" is the same. More formally, let C be a cone thus defined and x a point in C distinct from the origin. We define $F(C)$ as the set of facets f of P with C in h_f^+ (where the latter is defined as the halfspace bounded by h_f on the side defined by v_f). It is not hard to verify that

$$\mu(x) = \frac{1}{|x|} \sum_{f \in F(C)} \langle x, v_f \rangle = \frac{1}{|x|} \left\langle x, \sum_{f \in F(C)} v_f \right\rangle.$$

Let us define $v_C = \sum_{f \in F(C)} v_f$ and let $\bar{C}$ be the cone such that v_C has maximum length. By a result of McKenna and Seidel [MS], $v_{\bar{C}}$ belongs to $\bar{C}$; consequently, $S(v_{\bar{C}})$ has maximum measure among all shadows. This maximizing cone $\bar{C}$ can be found by a method akin to that used in the previous application 5.8. Each crossing of a hyperplane to visit an adjacent cone either brings a new facet "into view," or takes one out, so v_C changes only locally. As a matter of fact, $v_{\bar{C}}$ is actually maximum even among all sums of the form $\sum_{f \in \mathscr{F}} v_f$, where $\mathscr{F}$ is *any* subset of the facets of P. Therefore we can also find this maximum by solving an optimal assignment problem on the vectors v_f by the method of the previous subsection (this is a $\{0, 1\}$ assignment which is linearly related to a $\{+1, -1\}$ assignment).

For the shadow with minimum measure we can restrict our attention to directions determined by the intersection of $d-1$ of the hyperplanes h_f, for otherwise we can move our direction onto some hyperplane and reduce the sum v_C. As in Section 5.8, the measure of all shadows in the specified directions can be computed in constant time per direction. The observations above improve the algorithms in McKenna and Seidel [MS] which take either $O(n^{d-1})$ time and storage or $O(n^{d-1} \log n)$ time and $O(n)$ storage.

THEOREM 5.9.1. *The minimum and maximum shadow of a convex polytope in E^d with n facets can be computed in $O(n^{d-1} + n \log n)$ time and $O(n)$ storage.*

A number of other variants of this problem are possible. For example, by analogous techniques we can compute in the same time bounds the direction(s) of view that maximize or minimize the number of "visible vertices." A slightly more challenging problem is that of computing the direction of view from which a convex polytope in E^3 has the most or least vertices *on its silhouette*. The only difficulty in these problems is that as we cross one of the hyperplanes discussed above, the information associated with the curent cone changes by more than a constant amount. For example, in the silhouette problem we are essentially "xor"ing into our current set of silhouette vertices those of the facet coming in/out of view. However, during the execution of the sweep through all cones, a particular facet cannot come in/out of view more than n times (a sweep crosses a line of a two-dimensional arrangement $n-1$ times), so it contributes to the total updating cost proportionally to n times its size. A simple argument now shows that this still leaves the total time cost of our algorithm $O(n^2)$.

6. Open Problems and Conclusions

There are several open questions associated with the problems discussed in this paper. Among them are the following:

(a) Can the vertices of an arrangement of n lines in E^2 be sorted in x-order from left to right in $O(n^2)$ time? This problem is at least as hard as the classical problem of sorting $X + Y$, which also remains open [Fr].

(b) Can the idea of topological sweep be extended to higher dimensions?

(c) Can the topological sweep yield improved results for the problem of computing all intersecting pairs among n line segments in E^2?

(d) Can the method of Section 5.1 of trading time for space be either completely avoided in that context, or improved?

(e) How fast can we compute the shadow of a polytope in E^3 of minimum/maximum perimeter?

In conclusion, this paper has presented a new technique for sweeping a two-dimensional arrangement that allows us to visit all elements of the arrangement in a consistent ordering. The technique is extremely simple to implement: nothing beyond simple arrays (or linked lists) is needed. It is fast in both practice and theory, where it improves either the space or the time performance of previously known methods. The technique has many applications to planar as well as higher dimensional problems is computational geometry. Since point sets are the duals of arrangements, many problems about collections of points in a Euclidean space can be attacked by using the topological sweep. Numerous examples have been given in this paper.

Acknowledgments

The authors wish to thank Raimund Seidel for suggesting the argument that we used to prove Theorem 3.1, Harald Rosenberger who implemented the topological sweep and compared it with a straight line sweep, the students who took the Stanford 1985 analysis of algorithms qualifying examination and suffered through a version of this problem, and finally Lyle Ramshaw and Cynthia Hibbard for their detailed reading and comments on the manuscript. The constructive criticism of an anonymous referee is also appreciated.

References

[AA] T. Asano, T., Asano, L. J. Guibas, J. Hershberger, and H. Imai, Visibility-polygon search and euclidean shortest paths, *in* "Proceedings, 26th Annual Found. of Comput. Sci. Symposium, 1985," pp. 155–164.

[AHU] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms," Addison–Wesley, Reading, MA, 1974.

[AR] D. Avis and D. Rappaport, Computing the largest empty convex subset of a set of points, *in* "Proceedings, 1st ACM Symposium on Comput. Geom., 1985, pp. 161–167.

[BS] J. Bentley and J. B. Saxe, Decomposable searching problems I. Static to dynamic transformation, *J. Algorithms* **1** (1980), 301–358.

[CGL] B. M. Chazelle, L. J. Guibas, and D. T. Lee, The power of geometric duality, *BIT* **25** (1985), 76–90.

[CK] V. Chvatàl and G. Klincsek, Finding largest convex subsets, *in* "Proceedings, 11th SE Conf. on Combin., Graph Theory and Comput, 1980"

[E] H. Edelsbrunner, "Algorithms in Combinatorial Geometry," Springer-Verlag, Heidelberg, 1987.

[EM] H. EDELSBRUNNER AND E. P. MÜCKE, Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms, manuscript, 1987.

[EGS] H. EDELSBRUNNER, L. J. GUIBAS, AND J. STOLFI, Optimal point location in a monotone subdivision, *SIAM J. Comput.* **15** (1986), 317–340.

[EOS] H. EDELSBRUNNER, J. O'ROURKE, AND R. SEIDEL, Constructing arrangements of lines and hyperplanes with appllications, *SIAM J. Comput.* **15** (1986), 317–340.

[EOW] H. EDELSBRUNNER, M. H. OVERMARS, AND D. WOOD, Graphics in flatland: A case study, *in* "Advances in Computing Research 1" (F.P. Preparata, Ed.), pp. 35–59. Jai, Greenwich, CT, 1983.

[ES1] P. ERDÖS AND G. SZEKERES, A combinatorial problem in geometry, *Composition Math.* **2** (1935), 463–470.

[ES2] P. ERDÖS AND G. SZEKERES, On some extremum problems in elementary geometry, *Ann. Univ. Sci. Budapest* **3** (1060), 53–62.

[EW] H. EDELSBRUNNER AND E. WELZL, Constructing belts in two-dimensional arrangements with applications, *SIAM J. Comput.* **15** (1986), 271–284.

[FO] G. FABELLA AND J. O'ROURKE, "Twenty-two points with No Empty Hexagon," Rep. JHU/EECS-8603, Dept. of Electri. Engin. and Comput. Sci. Johns Hopkins University, Baltimore, MD, 1986.

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

[FT] M. L. FREDMAN AND R. E. TARJAN, Fibonacci heaps and their uses in improved network optimization algorithms, *in* "Proceedings, 21st Annual Found. of Comput. Sci. Symposium, 1984," pp. 338–346.

[G B. GRÜNBAUM, "Arrangements and Spreads," Reg. Conf. Ser. Math., Amer. Math. Soc., Providence, RI, 1972.

[GS] L. J. GUIBAS AND J. STOLFI, Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Trans. Graphics* **4** (1985), 74–123.

[Ha] H. HARBORTH, Konvexe-Fünfecke in ebenen Punkmengen, *Elem. Math.* **33** (1978), 116–118.

[Ho] J. D. HORTON, Sets with no empty convex 7-gon, *Canad. Math. Bull.* **26** (1983), 482–484.

[Kn] D. E. KNUTH, "The Art of Computer Programming, Vol. I, Fundamental Algorithms," Addison–Wesley, Reading, MA, 1971.

[L] D. T. LEE, "Proximity and Reachability in the Plane," Ph. D. thesis, Dept. Elec. Engin., University of Illinois, 1979.

[LP] D. T. LEE AND F. P. PREPARATA, Euclidean shortest paths in the presence of rectilinear barriers, *Network* **14** (1984), 393–410.

[MS] M. MCKENNA AND R. SEIDEL, Finding the optimal shadow of a convex polytope, *in* "Proceedings, 1st ACM Symp. on Comput. Geom., 1985," pp. 24–28.

[NP] J. NIEVERGELT AND F. P. PREPARATA, Plane-sweep algorithms for intersecting geometric figures, *Comm. ACM* **25** (1982), 739–747.

[PS F. P. PREPARATA AND M. I. SHAMOS, "Computational Geometry, an Introduction," Springer-Verlag, New York/Berlin, 1985.

[Sh] M. SHARIR, personal communication, 1985.

[T] R. E. TARJAN, Amortized computational complexity, *SIAM J. Comput.*, in press.

[W] W. WELZL, Constructing the visibility graph for n line segments in $O(n^2)$ time, *Inform. Process. Lett.* **20** (1985), 167–171.

Printed by Catherine Press, Ltd., Tempelhof 41, B-8000 Brugge, Belgium