BIT 25 (1985), 76–90

THE POWER OF GEOMETRIC DUALITY

BERNARD CHAZELLE [1]), LEO J. GUIBAS [2]) and D. T. LEE [3])

[1]) *Department of Computer Science, Brown University, Box 1910, Providence, R.I., U.S.A.*
[2]) *Computer Science Laboratory, Xerox PARC, Palo Alto Research Center, Palo Alto, California 94304, U.S.A.*
[3]) *Northwestern University, Evanston, Illinois 60201, U.S.A.*

Abstract.

This paper uses a new formulation of the notion of duality that allows the unified treatment of a number of geometric problems. In particular, we are able to apply our approach to solve two long-standing problems of computational geometry: one is to obtain a quadratic algorithm for computing the minimum-area triangle with vertices chosen among n points in the plane; the other is to produce an optimal algorithm for the half-plane range query problem. This problem is to preprocess n points in the plane, so that given a test half-plane, one can efficiently determine all points lying in the half-plane. We describe an optimal $O(k+\log n)$ time algorithm for answering such queries, where k is the number of points to be reported. The algorithm requires $O(n)$ space and $O(n \log n)$ preprocessing time. Both of these results represent significant improvements over the best methods previously known. In addition, we give a number of new combinatorial results related to the computation of line arrangements.

1. Introduction.

1.1. *Generalities on the concept of duality.*

A natural duality between points and lines in the Cartesian plane has long been known to geometers. Under this duality a point p with coordinates (a, b) corresponds to the line T_p with equation $ax+by+1=0$. The point and the corresponding line are called *dual* of each other; the point belongs to the *primal* plane, and the line to the *dual* plane. Geometrically, if d is the distance from the origin O to point p, the dual T_p of p is the line perpendicular to Op at distance $1/d$ of O and placed on the other side of O (Fig. 1). In order to handle all cases, this definition requires that the plane be extended (by the addition of points at infinity) in such a way that it becomes topologically equivalent to a non-orientable surface, the so-called *projective plane*. As a consequence of this non-orientability, there is no consistent way to define the "left side" of an oriented line, or to define a "counterclockwise sense of rotation" for all points of the plane.

We solve this problem by using a different extension of the plane, that is endowed with the topology of a sphere (and therefore is orientable). Informally

Received December 1983. Revised May 1984.

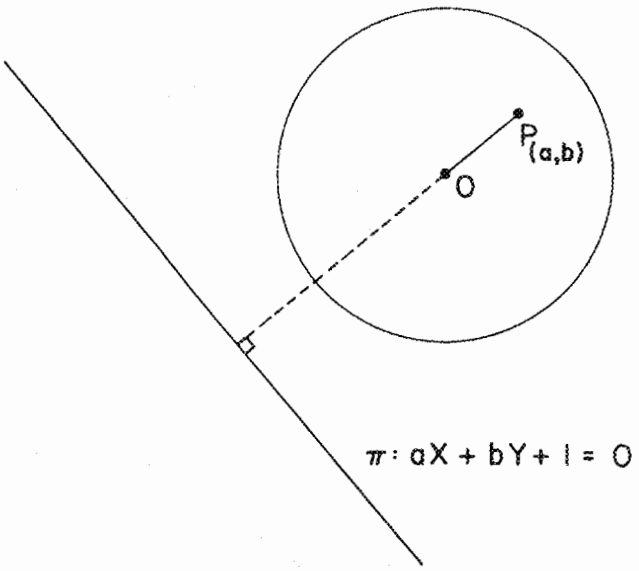

Fig. 1. The dual transform T.

speaking, this extension is made by distinguishing a point on the "top" side of the plane from its *antipode*, a point with the same position but on the "bottom" side of the plane. For such a point, the meaning of "counterclockwise" is defined with respect to an observer standing on the same side of the plane. So, if a car describes a counterclockwise loop on the top side of the plane, a second car on the bottom side that passes through the same positions in the same order will describe a clockwise loop. We call this extension the *two-sided plane* (2SP). One can view this construction algebraically by thinking about homogeneous coordinates. In classical homogeneous coordinates we identify the points $[x,y,z]$ and $[\lambda x, \lambda y, \lambda z]$ for each real $\lambda \neq 0$. In the new manifold we only do so if $\lambda > 0$. Multiplying by a negative constant gives us the antipode of our point.

A straight line extends to both sides of the 2SP, and therefore passes through a point if an only if it passes through its antipode. An oriented line has the same direction vector on both sides; however, since the notions of "left" and "right" are relative to a local observer, a point is to the left of an oriented line L if and only if its antipode is to the right of L. The left and right "half-planes" of L are well-defined concepts, and their union is the whole 2SP minus the line L.

By working on the 2SP, we can define the duality between points and lines so as to preserve relative orientations. Under this mapping, a point p *is transformed to an oriented* line T_p, and vice versa: the line T_p is oriented counterclockwise (as seen from the origin O) if and only if p lies on the top side of the plane. Note that the dual of a line passing through the origin is a *point at infinity*, and the dual of the origin is the *line at infinity*. It follows from this

construction that a point p lies to the left of an oriented line L if and only if the dual point T_L lies to the left of the line T_p. As is well-known, the transformation T has also the property of preserving incidence relations, i.e., a point *p lies on a line L* if and only if the line T_p contains the point T_L.

The concept of duality is a powerful tool for the description, analysis, and construction of algorithms. By applying it to theorems, proofs, operations, and algorithms we can obtain twice as many results with practically the same effort [1, 15]. For example, an algorithm for computing the convex hull of n points in the plane *automatically* becomes an algorithm for computing the intersection of n half-planes. A more formal development of the theory underlying this new duality, due to Guibas, Ramshaw, and Stolfi, is given in [12].

1.2. *Organization of the paper.*

The purpose of this paper is to use the concept of duality to study a number of problems involving points in the plane. In Section 2, we describe an optimal algorithm for computing the planar graph G formed by n lines in the plane, and we use this result in Section 3 to improve on the best algorithm previously known for finding the smallest-area triangle among n points. In the same section, we prove a property of G that is fairly unusual for a planar graph, and we use this fact to derive an improved algorithm for computing the *empty wedges* of a point set. Finally, we give the main result of this paper in Section 4, i.e., the first space and time optimal algorithm for the *half-plane range query problem*. This problem is to preprocess n points in the plane, so that given a test half-plane, one can efficiently determine all points lying in the half-plane. We describe an $O(k+\log n)$ time algorithm for answering such queries, here k is the number of points to be reported. The algorithm requires $O(n)$ space and $O(n \log n)$ preprocessing time.

2. Computing line arrangements.

Since the dual of a point set is a set of lines, it is important for the following developments to devise an efficient method for computing the planar graph formed by a set of lines. Let $T_1, \ldots, T_n$ be n lines in the plane and let G denote the subdivision of the plane induced by the lines. For simplicity, we will assume that all the infinite rays of the graph meet at a common point at infinity*. This will ensure that all the faces of G are properly enclosed by edges. Let p (resp. q) be the number of vertices (resp. edges) of the planar graph formed by G. It is easy to see that $p(1) = 1$, $q(1) = 1$, $p(n) \leqq p(n-1)+n-1$, and

* This convention is equivalent to assuming that our construction takes place on the upper hemisphere of a sphere whose equator has been pinched to a point.

$q(n) \leqq q(n-1)+2n-1$, therefore

$$p(n) \leqq \frac{n(n-1)}{2}+1, \quad q(n) \leqq n^2.$$

We now turn to the actual computation of the graph G. We represent the graph by means of adjacency lists, i.e., we associate with each vertex a list of its adjacent vertices in, say, clockwise order. This includes a list for the vertex at infinity. This representation is called a *vertex-to-edge* representation of G. Since generally the degree of each vertex is four, this representation allows us to traverse any face of the graph in time proportional to its number of vertices in both the clockwise and counterclockwise directions. To handle singularities such as the collinearity of several points in the primal plane and still retain the capability of efficient face traversal, we convert the traditional adjacency-list representation into the *doubly-connected-edge-list* representation or the *quad-edge* structure [11]. These are linear time transformations which involve only simple pointer manipulations, so we omit the details and instead refer the reader to the aforementioned sources.

We compute G iteratively, inserting each line T_i in turn. Assume that we have already inserted $T_1, \ldots, T_{k-1}$. To insert the line $L = T_k$, we start by determining which edge on T_1 it intersects. This can be done by keeping track of a reference edge on T_1, from which we can begin traversing the line T_1 towards its intersection I with L. To do so, we can assume that we keep the ordered sequence of edges on T_1 in a special list. We begin by inserting the intersection I as a new vertex in G. Note that if L and T_1 are parallel, this new vertex is actually the vertex at infinity.

This preliminary step in the procedure involves adding a new vertex-to-edge list and updating the lists corresponding to I's neighbors. We then start traversing the edges of G around each face crossed by L. We first do it in one direction, then restart in the other direction. There is no difficulty doing this traversal, since it involves only taking *rightmost* (or *leftmost*) turns at each vertex (Fig. 2). Note that by backing up one edge at each new face, we can carry out the complete traversal, while staying entirely on one side of L. This is indeed possible, because of the vertex at infinity which ensures the effective *boundedness* of all the faces. Of course, appropriate updating should take place during the traversal of G. This involves straightforward graph manipulation, and we omit the details. Note that when first encountering two consecutive edges adjacent to the vertex at infinity and lying on distinct side of L, the procedure should stop after inserting the last edge. This edge will point to infinity.

The entire traversal requires $O(t)$ operations, where t is the total number of vertices on all the faces of G intersecting L. The next result establishes an upper bound on t.*

* A similar result has been recently discovered independently by Edelsbrunner et al. [7].

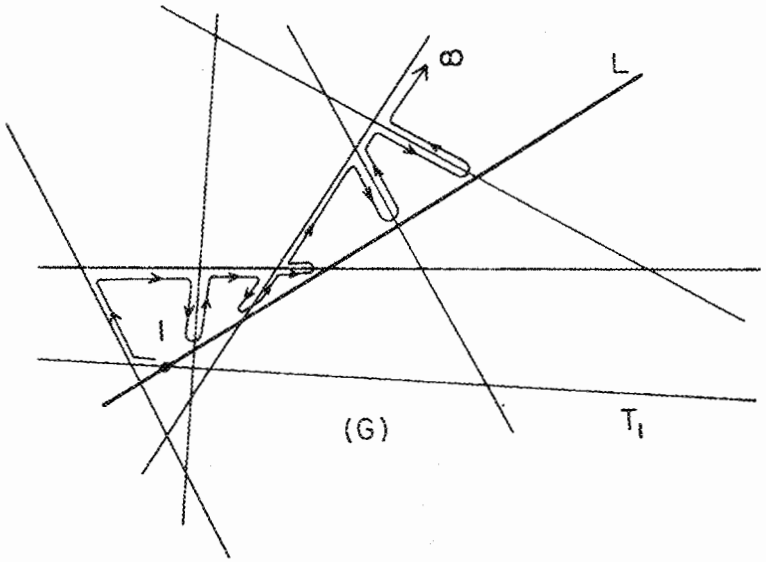

Fig. 2. Computing a line arrangement.

LEMMA 1. *The total number of edges on the faces of G intersecting the line L does not exceed* $8n$.

PROOF. For convenience we may assume that L is horizontal. We can then distinguish the faces adjacent to L *above* from the faces adjacent *below*. Since the two cases are strictly identical, we will restrict our attention to the former. In clockwise order, a face f consists of a horizontal or *base* edge, followed by a sequence of first *left* then *right* edges. An edge is said to be left (resp. right) if its supporting line intersects L on the left (resp. right) hand side of the base. Note that the edges adjacent to L, called *grounded* edges, are right edges for the face on their left-hand side and left edges for the face on their right-hand side. Let us charge each left but non-grounded edge to the intersection of the line supporting the previous (in the clockwise sense) left edge with L (Fig. 3). We define a symmetric charging scheme for the right non-grounded edges. It is easy to see that each vertex on L can be charged at most once from a left edge and at most once from a right edge. Actually the leftmost vertex cannot be charged from a left edge and the rightmost vertex cannot be charged from a right edge. Thus the number of traversed edges above L is bounded by $4k-2$, where k is the number of intersections of L by the rest of the lines. This bound is in fact the best possible. Since the same reasoning can be applied to the faces below L as well, the proof is complete. ■

From Lemma 1, we find that each insertion takes $O(n)$ time and therefore, as claimed earlier, the overall algorithm is quadratic.

THEOREM 1. *It is possible to compute an arrangement of n lines in* $O(n^2)$ *time.*

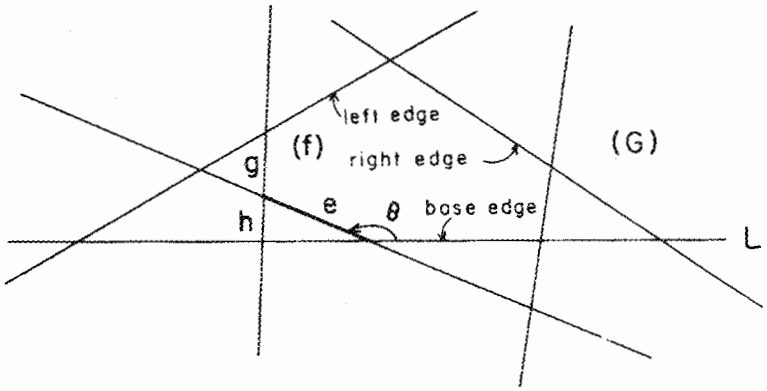

Fig. 3. Inserting a new line.

3. Smallest-area triangle, empty wedges, and other applications.

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in the plane. We will apply Theorem 1 to the problem of determining which of the $\binom{n}{3}$ triangles with vertices in S has the smallest area. The difficulty of this problem resides essentially in the absence of *locality*. Indeed, the vertices of the smallest triangle may be arbitrarily distant from one another, if they happen to be collinear (this illustrates the fact that solving this problem will allow us to determine whether three points in S are collinear). Using duality, we are able to strenghten the time complexity of the best algorithm known for this problem, i.e., the $O(n^2 \log n)$ time and $O(n)$ space algorithm of [8].

For the sake of convenience, we will use a slightly different dual transform, denoted D, and defined as follows. A point p: (a, b) is mapped to the line D_p: $Y = aX + b$, and a line L: $Y = kX + d$ is mapped to the point D_L: $(-k, d)$. We should observe that two points are mapped to two parallel lines if and only if these points have the same X-coordinates. We will assume that no points of S share the same X-coordinates. This may possibly entail rotating the axes by a small angle, easily determined in $O(n \log n)$ time. We define G as the planar graph formed by the lines $\{D_{p_1}, \ldots, D_{p_n}\}$.

Let $L(i, j, k)$ denote the line passing through p_k that is parallel to $p_i p_j$, and let $h(i, j, k)$ be the vertical distance between $p_i p_j$ and $L(i, j, k)$, i.e., the length of a vertical segment joining $p_i p_j$ and $L(i, j, k)$. It is clear that the smallest-area triangle with $p_i p_j$ as an edge minimizes $h(i, j, k)$ for all $k \neq i, j$; $1 \leqq k \leqq n$. We can express this property in the dual plane. The line through $p_i p_j$ becomes the intersection point I of D_{p_i} and D_{p_j}, and $L(i, j, k)$ is mapped to the point on D_{p_k} which has the same X-coordinate as I. It is now easy to check that the vertical distance between I and D_{p_k} is precisely ($x(\ )$ and $y(\ )$ denote the X and Y coordinates respectively):

$$h(i, j, k) = y(p_k) + \frac{x(p_k)[y(p_j) - y(p_i)] + x(p_j)y(p_i) - x(p_i)y(p_j)}{x(p_i) - x(p_j)}.$$

To determine the smallest-area triangle among the points of S, we simply mimic the computation of G described in the previous section. We traverse each line D_{p_k} in turn, examining all the vertices v_i of each face incident to D_{p_k}. Note that each v_i is the intersection of two lines D_{a_i} and D_{b_i}. From the remarks above, it follows that the smallest triangle with vertex p_k will be of the form (p_k, a_i, b_i), so it will be effectively found with this procedure. Finally, Theorem 1 shows that the entire computation will take $O(n^2)$ time. We conclude:

THEOREM 2. *It is possible to compute the smallest-area triangle among n points in $O(n^2)$ time and space.*

We can use Lemma 1 in order to derive a fairly curious result about arbitrary arrangements of lines in the plane. Consider the dual graph of G (in the graph-theoretic sense), denoted H, and defined as follows: each face of G is a vertex of H and two vertices of H are connected by an edge if and only if the corresponding faces in G share a common edge. Let V (resp. E) denote the number of vertices (resp. edges) in H. Clearly $V = O(n^2)$, and since H is a planar graph with no multiple edges, we have $E = O(V)$. Let $w_1, \ldots, w_V$ be the vertices of H, and let d_i denote the degree of vertex w_i. It is easy to see that

$$\sum_{1 \leq k \leq V} d_k = 2E = O(V).$$

We can strenghten this relation and prove a relation that is not true for general planar graphs.

LEMMA 2. *Let G be any arrangement of n lines with $V = \Omega(n^2)$ vertices. The vertex-degrees of the dual graph H satisfy the relation:*

$$\sum_{1 \leq k \leq V} d_k^2 = O(V).$$

PROOF. Let $f_1, \ldots, f_l$ be all the faces of G incident to a given line L of the arrangement. Without loss of generality, assume that $w_1, \ldots, w_l$ are the corresponding vertices in H. From Lemma 1, we have

$$\sum_{1 \leq i \leq l} (d_i - 1) \leq 8n.$$

Summing the left-hand side of the inequality over all the lines in G will have the effect of duplicating each term $(d_i - 1)$ exactly d_i times, therefore

$$\sum_{1 \leq k \leq V} d_k(d_k - 1) \leq 8n^2.$$

Since, as we already noticed, $\sum_{1 \leq k \leq V} d_k = O(V)$, we have $\sum_{1 \leq k \leq V} d_k^2 = O(n^2) = O(V)$. ■

We can use Lemma 2 to derive an interesting combinatorial result on planar point sets. Consider S, our set of n points in the plane, $p_1, \ldots, p_n$, and let $L_{i,j}$ be the line passing through p_i and p_j. Any pair of such lines defines two *double wedges*, and it is easy to see that the n points define on the order of n^4 double wedges. We say that a double wedge is *empty* if it does not strictly contain any point of S (Fig. 4). The next result will show in particular that there are only $O(n^2)$ empty double wedges.

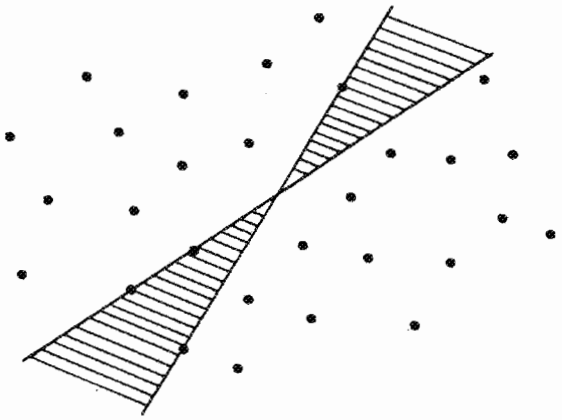

Fig. 4. An empty double wedge.

LEMMA 3. *It is possible to enumerate all the empty double wedges formed by n points in* $O(n^2)$ *time.*

PROOF. Place a line L in arbitrary position in the original plane, and move it in a continuous fashion anywhere, as long as it does not intersect any point of S. It is easy to see that the locus of the points D_L thus defined coincides exactly with one of the faces of G. It follows that the dual transform D puts the empty double wedges in one-to-one correspondence with the segments that have vertices of G as endpoints and do not intersect any edge of G except at their endpoints. Since there are no more than $\sum_{1 \leq k \leq V} \binom{d_k}{2}$ such segments, Lemma 2 shows that we have at most $O(V) = O(n^2)$ empty double wedges. To compute them, we can use the representation of G and examine each face in turn. Note that in order to obtain the empty wedges containing a vertical line, we can reset the problem with the roles of x and y now exchanged. ■

4. The half-plane range query problem.

We now turn to the main result of this paper. Given a set S of n points

in the plane, we consider the problem of reporting all the points in S that lie on a given side of a query line. From now on, k will refer to the number of points to be reported. Previous work on this problem includes an $O(n^3)$ space data structure that allows us to answer a query in $O(k+\log n)$ time [6]. Other solutions have been proposed for solving the half-plane range query problem, using only linear space: one is the polygon tree of Willard [17], and more recently the conjugation tree of Edelsbrunner and Welzl [9]. These structures yield $O(n^{0.77})$ and $O(n^{0.695})$ query time, respectively. Yao [18] provided an $O(n^{0.98}+k)$ time and $O(n)$ space algorithm for the same problem in three dimensions, as well as for circular queries in two dimensions. We show here that in two dimensions it is possible to achieve optimal $O(k+\log n)$ query time, using only $O(n)$ space. We should stress the fact that blithely comparing these complexity results would be unfair. Although our algorithm is significantly superior to the methods of [6, 9, 17] when the points are to be explicitly reported, it is inadequate for, say, counting points or dealing with more complex test figures such as polygons. For these situations, the polygon tree of [17] and the conjugation tree of [9] are still the most efficient tools known to date.

As a starter, recall the basic method of the algorithm in [6]. We compute G, the dual set of S, via the transform D, and for each face of the graph, we keep a list of all the lines of G lying respectively above and below the face. This allows us to solve our problem in time $O(k+\log n)$, since a point p of S is above (resp. below) a query line L, if and only if its dual line, D_p, is above (resp. below) the dual point D_L. Indeed, with this preprocessing in hand, we can answer a query by first determining which face of G contains the point D_L, and then outputting the contents of the associated list containing the lines *above* (resp. *below*) D_L if the query halfplane lies above (resp. below) L. Kirkpatrick's optimal planar point location algorithm can be used for this purpose [13], so as to locate T_L in G in $O(\log n)$ operations. This scheme clearly entails the use of cubic space, which we can then proceed to reduce to $O(n^2)$ by providing G with a set of $O(n^2)$ additional pointers.

We say that an edge of a face f of G is an *upper* (resp. *lower*) edge with respect to f if its inward-directed normal points downward (resp. upward). Let e_1 and e_2 be two arbitrarily chosen edges of f, respectively of the upper and lower type with respect to f. We set up a pointer from each lower (resp. upper) edge of f to e_1 (resp. e_2). With this construction, we can simply follow the pointers from D_L upward or downward, depending on the orientation of the query halfplane. This will lead us to cross each of the desired lines exactly once. Proving the validity of this fact is straightforward, so we leave out the details. Note that we stop when reaching a face with no upper (lower) pointer, if going upward (resp. downward). The choice of e_1 and e_2 will determine the order in which the lines are reported, which is immaterial for our purposes. This addition of special pointers allows us to save a factor of n in storage, thus reducing it to $O(n^2)$.

To produce a more radical breakthrough in the complexity of the algorithm, while retaining the same basic idea, let us momentarily step back to the original point set S. The first phase of our algorithm consists of partitioning S into its set of convex layers (sometimes referred to as "onion"). These layers are convex polygons, defined iteratively as follows (Fig. 5): intially i is 1.

1. Define S_i as the convex hull of all the points currently in S.
2. Remove the vertices of S_i from S.
3. If S is not empty, increment i and go back to 1. Otherwise stop.

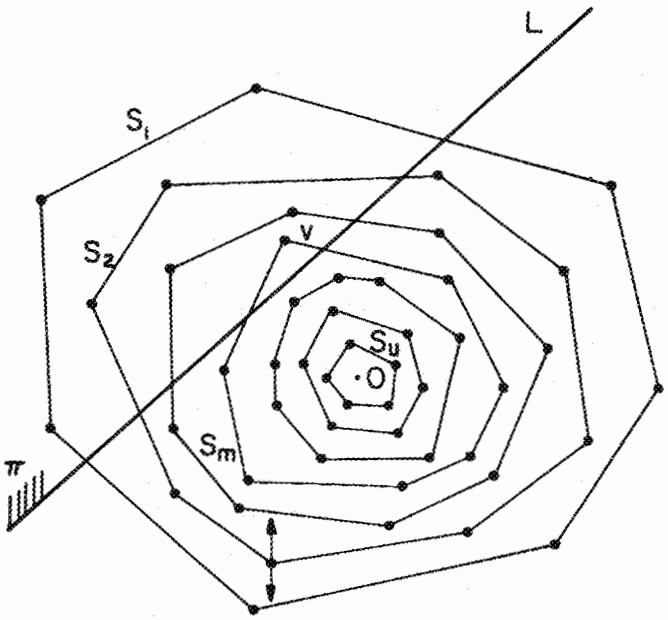

Fig. 5. The set of convex layers.

Let u be the total number of layers; from now on, any use of duality will refer to the transform T, with the origin placed anywhere in the interior of S_u. Overmars and van Leeuwen have shown how to compute the layers of n points in $O(n \log^2 n)$ time using a general technique for computing convex hulls in a dynamic environment [16]. This result was recently improved in [2], where an optimal $O(n)$ space, $O(n \log n)$ time algorithm was described. With this preprocessing in hand, we may now proceed to present the main part of the algorithm. Let L be the line delimiting the query half-plane π. It is easy to determine whether L intersects S_i, for a given i, in $O(\log n)$ time, using the Fibonacci-search technique of [4]. If the answer is negative, either π contains no point of S_i or it contains them all, and this can be decided in constant time. It is easy to prove that the set of hulls S_i intersected by L forms an initial segment of the sequence $S_1, S_2, \ldots, S_u$. Let us assume also that L does intersect S_1, which implies in turn that π contains at least one point of S;

otherwise the problem is trivial. Let $S_1, \ldots, S_m$ be the layers that L intersects. We say that S_m is the *neighboring* layer of L (Fig. 5). We will prove later on that the index m can be computed in $O(\log n)$ operations, so we assume the validity of this result for the time being. With this information, we can compute the two (possibly coinciding) intersection points of L and S_m in $O(\log n)$ time. This immediately gives us a vertex v of S_m that lies in the half-plane π.

The goal at this point is to find one vertex lying in π on each layer $S_1, \ldots, S_{m-1}$. To do so, we augment our set of layers with *vertical connections*, by proceeding as follows: for each vertex w of S_i $(1 \leqq i \leqq u)$ keep a pointer to the two edges immediately above and below w, i.e. the edges hit by a vertical ray emanating from w upward or downward (Fig. 5). The first type of pointers is called *upper*, the second *lower*. Once the convex layers have been computed, we can easily set up these pointers in linear time, proceeding in a merge-like fashion for each pair of consecutive layers. We omit the details of this very simple procedure. To achieve our goal, we now iterate on the following process: starting from v, follow the upper (resp. lower) pointer, if L lies below (resp. above) π. The edge e pointed to has at least one endpoint in π (for example its endpoint in π further from L), so we can simply continue the iteration from this point.

Since we proceed from the layer S_m outwards, we will effectively report one vertex on each of the layers $S_1, \ldots, S_m$ (Fig. 6-A). Depending on the position of L with respect to the origin, we may even end up reporting more vertices, but all these extra vertices will be in π, however (Fig. 6-B). The next step is to start a traversal from each of the vertices reported, on their corresponding layer, sweeping across the part of the layer inside π. Note that if the origin lies in π —which can be determined in constant time— we must also report all the vertices in $S_{m+1}, S_{m+2}, \ldots, S_u$ (Fig. 6-B). This may cause vertices to be duplicated in the report. It is easy, however, to eliminate duplicates in $O(k)$ time, using an array of indices, so this will not affect the overall asymptotic complexity of the algorithm.

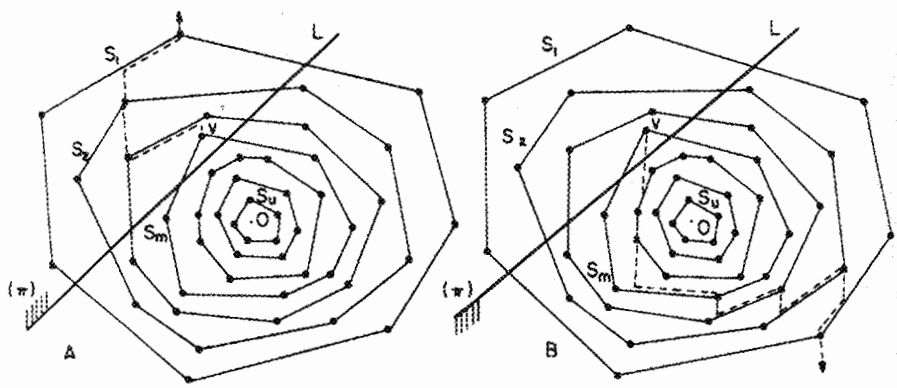

Fig. 6. Walking across the layers.

The method above is very simple to describe, but it may lead to certain implementation difficulties if L is vertical. We describe below a different method for the same "walk outwards" that requires slightly more storage but is completely isotropic. This alternative method rests crucially on an interesting geometric observation. If Q is a convex polygon that contains another convex polygon P in its interior so that each diagonal of Q intersects P, then we call Q an *immediate cover* of P (Fig. 7).

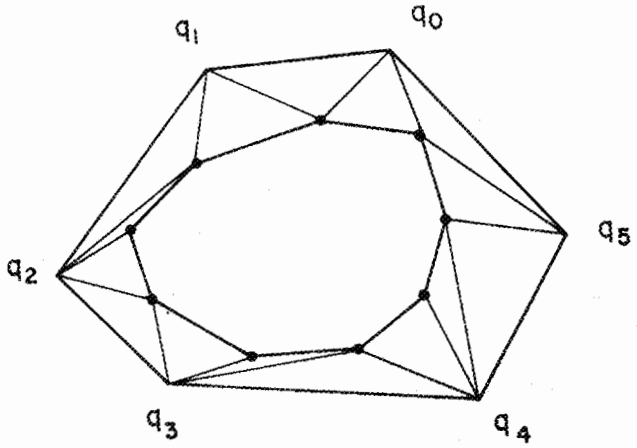

Fig. 7. The triangulation approach.

LEMMA 4. *Suppose that convex polygon Q is an immediate cover of convex polygon P and that the region between P and Q has been triangulated. Then each vertex of P will be connected to at least one vertex and at most four vertices of Q in the triangulation.*

PROOF. Let p be a vertex of P. Consider the two edges of P incident at p. Each of them defines a half-plane not containing P. No more than two vertices of Q may lie in such a half-plane, as otherwise there would be a diagonal of Q avoiding P. From this the lemma follows. ■

Given two successive convex hulls S_i and S_{i+1}, the outer hull can be triangulated in any greedy fashion, as long as none of the diagonals drawn intersects the inner hull. At that point the polygon enclosing the inner hull forms an immediate cover of it. The triangulation of the ring between the two layers can now be completed in any manner whatsoever. This guarantees that at most four of the triangulation edges reach every inner vertex. It is easy to see how to implement such a triangulation in linear time by keeping two roving pointers, one on S_i

and one on S_{i+1}. The algorithm of Garey et al. [10] can also be adapted to this task. Once obtained, the doubly-connected-edge-list or quad-edge representation of this planar graph can be used to carry out the above walk.

Implementing and testing both methods on concrete examples will decide which one should be retained in practice. We now turn our attention to the computation of S_m, the neighboring layer of L.

LEMMA 5. *Once the convex layers have been computed, the value of m can be found in $O(\log n)$ time, after $O(n)$ preprocessing.*

PROOF. The problem is that of "locating" a line among a set of convex polygons. We use duality to recast the problem as a planar point location problem. Applying the transform T to each point p_i we can show that, if as mentioned earlier, the origin is chosen inside S_u, each layer is mapped into another convex polygon, and furthermore, the u layers of S are mapped into another set of u layers with the nested order reversed, i.e., the innermost layer becomes the outermost one and vice-versa. Also, it can be easily shown that the point T_L lies between the dual versions of the two layers S_m and S_{m+1}. We can then preprocess the set of dual layers in $O(n)$ time, so as to apply a planar point location algorithm to T_L, and thus determine m in $O(\log n)$ time. Possible choices are Kirkpatrick's method [13] based on a hierarchy of coarser and coarser subdivisions, or Lipton and Tarjan's remarkable, yet impractical algorithm [14]. ■

The proof of the lemma is complete; yet we wish to strengthen this result from a practical standpoint. One drawback of the method which we have just presented is its reliance on planar point location. Even for Kirkpatrick's algorithm, which is conceptually quite simple, it is not clear at all that this simplicity will carry over in practice. To circumvent this difficulty, we will show that it is possible to dispense with planar point location altogether, using the concept of *filtering search* introduced in [3]. Since, in our case, $O(\log n)$ seems to be an inherent search cost, we will try to partition the set of layers $S_1, \ldots, S_u$ into groups of roughly $\log n$ consecutive layers. Let $C_i = S_{i\lfloor \log n \rfloor}$. The algorithm will consist of searching for the value of the index m by computing the intersections of L with $C_1, C_2, C_3, \ldots$, iterating as long as there are, indeed, intersections to report. Let $C_1, \ldots, C_h$ be the intersecting layers thus discovered. Using a logarithmic intersection algorithm, as usual, this step will take $O(h \log n)$ time. Certainly S_m lies between C_h and C_{h+1}. To refine this information and actually determine the value of m, we will step into the dual space, proceeding as described above. One major difference, however, is that the dual structure will now consist of only $\lceil \log n \rceil$ convex boundaries. This allows us to do planar point location in a radically different (and simpler) way.

Let D be the dual structure of the layers S_i between C_h and C_{h+1}. To locate the point T_L among the boundaries of D, we first compute the α edges of D that intersect the vertical line V passing through T_L. To do so, we use the *hive-graph*

structure described in [3]. This is a simple structure which allows us to report all the intersections in time $O(\log n+\alpha) = O(\log n)$. From this information, we easily derive the value of m. Once m has been computed, we can proceed as described earlier. Since for any i, if L intersects both C_i and C_{i+1}, at least roughly $\log n$ points will be later reported between these two layers, the $O(h \log n)$ time spent in the first phase of the algorithm is also $O(k)$, so the total run time of the algorithm is $O(k+\log n)$.

An alternative for computing m, suggested to us by H. Edelsbrunner [5], involves using the fact that the hive-graph structure of [3] will also allow us to compute all β intersections between the segment OT_L and the dual structure of S in $O(\beta+\log n)$ time. It is easy to see that the farthest intersection from O lies on the dual of S_m, which gives us the value of m immediately. Since $\beta \leqq k$, this method will also require $O(k+\log n)$.

THEOREM 3. *It is possible to solve the half-plane range query problem on n points in $O(k+\log n)$ time, using $O(n)$ storage and $O(n \log n)$ time in preprocessing (k being the number of points reported). The method is modular and practical; it involves a straightforward planar point location and does not require the use of dynamic tree structures.*

The contribution of this work is to show the power of *geometric duality* for solving several geometric problems. We have described an optimal algorithm for computing the planar graph G formed by n lines in the plane. Using a duality argument, we have then shown how this result can be used for computing, in $O(n^2)$ time, the smallest-area triangle formed by n points. We also pointed out a property of G that is fairly unusual for a planar graph, and we used this property to derive an algorithm for computing the empty wedges of a set of n points. We then looked at the half-plane range query problem and described an optimal $O(n)$ space, $O(k+\log n)$ time algorithm for solving this problem, using again a duality argument (k is the size of the output).

An interesting open question is to investigate the possibility of implicitly representing the dual graph of a set of points so as to avoid a quadratic complexity. We have shown how to compute this graph optimally in two dimensions. But perhaps, the most interesting open problem is to try to study the existence of $O(n)$ or $O(n \log n)$ space algorithms for reporting or counting the points inside a polygon in $O(k+f(n))$ time, where $f(n)$ is a polylogarithmic function of n. Note that we have exploited the fact that a convex polygon entering a half-plane must have a vertex in the half-plane, a property which is false for more general query regions. Also our method for the half-plane range query problem entails the explicit enumeration of the points and thus seems inadequate to handle counting problems. This discrepancy is not specific to this problem, however: it is recurrent in most of the range search problems studied in the past.

REFERENCES

1. K. Q. Brown, *Geometric transforms for fast geometric algorithms*, PhD thesis, Carnegie-Mellon Univ., 1979.
2. B. Chazelle, *Optimal algorithms for computing depths and layers*, Brown University, Technical Report, CS-83-13, March 1983.
3. B. Chazelle, *Filtering search: A new approach to query-answering*, Proc. 24th Annual FOCS Symp., pp. 122-132, November 1983.
4. B. Chazelle and D. Dobkin, *Detection is easier than computation*, Proc. 12th Annual SIGACT Symp., Los Angeles, California, pp. 146-153, May 1980.
5. H. Edelsbrunner, *Private Communication*, June 1983.
6. H. Edelsbrunner, D. G. Kirkpatrick and H. A. Maurer, *Polygonal intersection searching*, Inf. Proc. Lett., 14, pp. 74-79, 1982.
7. H. Edelsbrunner, J. O'Rourke and R. Seidel, *Constructing arrangements of lines and hyperplanes with applications*, Proc. 24th Annual FOCS Symp., pp. 83-91, November 1983.
8. H. Edelsbrunner and E. Welzl, *Halfplanar range estimation*, Tech. Univ. of Graz, Tech Rep. F98, 1982.
9. H. Edelsbrunner and E. Welzl, *Halfplanar range search in linear space and $O(n^{0.695})$ query time*, Tech. Univ. of Graz, Tech Rep. F111, 1983.
10. M. R. Garey, D. S. Johnson, F. P. Preparata and R. E. Tarjan, *Triangulating a simple polygon*, Inf. Proc. Lett. 7(4), pp. 175-179, 1978.
11. L. J. Guibas and J. Stolfi, *Primitives for the manipulation of general subdivisions and the computations of Voronoi diagrams*, Proc. 15th Annual SIGACT Symp., pp. 221-234, April 1983.
12. L. Guibas, L. Ramshaw and J. Stolfi, *A kinetic framework for computational geometry*, Proc. 24th Annual FOCS Symp., pp. 100-111, November 1983.
13. D. G. Kirkpatrick, *Optimal search in planar subdivisions*, SIAM J. on Comp., Vol. 12, No. 1, pp. 28-35, February 1983.
14. R. J. Lipton and R. E. Tarjan, *Applications of a planar separator theorem*, SIAM J. Comp., 9(3), pp. 615-627, 1980.
15. D. E. Muller and F. P. Preparata, *Finding the intersection of two convex polyhedra*, Theoret. Comput. Sci. 7(1978), pp. 217-236.
16. M. H. Overmars and J. van Leeuwen, *Maintenance of configurations in the plane*, Journal of Computer and System Sciences, 23, p. 166-204, 1981.
17. D. E. Willard, *Polygon retrieval*, SIAM J. Comp., 11, pp. 149-165, 1982.
18. F. F. Yao, *A 3-space partition and its applications*, Proc. 15th Annual SIGACT Symp., pp. 258-263, April 1983.