Discrete Comput Geom 5:161-196 (1990)

Discrete & Computational
Geometry
© 1990 Springer-Verlag New York Inc.

The Complexity and Construction of Many Faces in Arrangements of Lines and of Segments*

Herbert Edelsbrunner,[1] Leonidas J. Guibas,[2,3] and Micha Sharir[4,5]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Ill 61801, USA

[2] DEC Systems Research Center, 130 Lytton Avenue Palo Alto, Ca 94301, USA

[3] Department of Computer Science, Stanford University, Ca 94305, USA.

[4] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA

[5] School of Mathematical Science, Tel Aviv University, 69978 Tel Aviv, Israel.

Abstract. We show that the total number of edges of m faces of an arrangement of n lines in the plane is $O(m^{2/3-\delta}n^{2/3+2\delta}+n)$ for any $\delta>0$. The proof takes an algorithmic approach, that is, we describe an algorithm for the calculation of these m faces and derive the upper bound from the analysis of the algorithm. The algorithm uses randomization and its expected time complexity is $O(m^{2/3-\delta}n^{2/3+2\delta}\log n + n\log n\log m)$. If instead of lines we have an arrangement of n line segments, then the maximum number of edges of m faces is $O(m^{2/3-\delta}n^{2/3+2\delta}+n\alpha(n)\log m)$ for any $\delta>0$, where $\alpha(n)$ is the functional inverse of Ackermann's function. We give a (randomized) algorithm that produces these faces and takes expected time $O(m^{2/3-\delta}n^{2/3+2\delta}\log + n\alpha(n)\log^2 n\log m)$.

1. Introduction

Let $L=\{l_1,l_2,\ldots,l_n\}$ be a finite set of lines in the plane. Let L induce a partition of the plane, known as the *arrangement* $A(L)$ of L, into $O(n^2)$ faces, edges, and vertices.

* The first author is pleased to acknowledge partial support by the Amoco Fnd. Fac. Dev. Comput. Sci. 1-6-44862 and the National Science Foundation under Grant CCR-8714565. Work on this paper by the third author has been supported by Office of Naval Research Grant N00014-82-K-0381, by National Science Foundation Grant DCR-83-20085, by grants from the Digital Equipment Corporation, and the IBM Corporation, and by a research grant from the NCRD-the Israeli National Council for Research and Development. A preliminary version of this paper has appeared in the *Proceedings of the 4th ACM Symposium on Computational Geometry*, 1988, pp. 44-55.

The *vertices* are the points of intersection of the lines in L, the *edges* are the connected components of the lines after removing the vertices, and the *faces* are the (convex) connected components of the complement of the union of the lines l_i (see [G] or[E] for more details concerning arrangements in the plane and in higher dimensions).

Many combinatorial properties of arrangements of lines have been studied extensively. In this paper we consider the maximum number, $K(m, n)$, of edges bounding m distinct faces in an arrangement of n lines in the plane (where we count an edge twice if it bounds two of these faces). Note that m can vary between 1 and $\kappa(n) = \binom{n}{2} + n + 1$, and that at these extreme values we have $K(1, n) = n$ and $K(\kappa(n), n) = 2n^2$ (there are altogether n^2 edges in the arrangement and each edge bounds two faces). A trivial upper bound for $K(m, n)$ is mn and a trivial lower bound is $2m$. Prior to this and a companion paper [CEG*],the best-known bounds on $K(m, n)$ for general values of m were

(i) $K(m, n) = n + 4\binom{m}{2}$ for $m \geq 2$ and $n \geq 4\binom{m}{2}$ [Ca],
(ii) $K(m, n) = O(mn^{1/2})$ for $n^{1/2} \leq m$ [EW1],
(iii) $K(m, n) = O(m^{1/2}n)$ [EW1], and
(iv) $K(m, n) = \Omega(m^{2/3}n^{2/3})$ [EW1]

(see also Chapter 6 of [E]). Note that each of the upper bounds has a different range of values of m for which it is better than the other (or the trivial) bounds. A graph showing these upper and lower bounds on a logarithmic scale is given in Fig. 1.1.

In this paper we improve the upper bounds by showing that

$$K(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n)$$

for any positive δ (with the constants of proportionality depending on δ).[1]

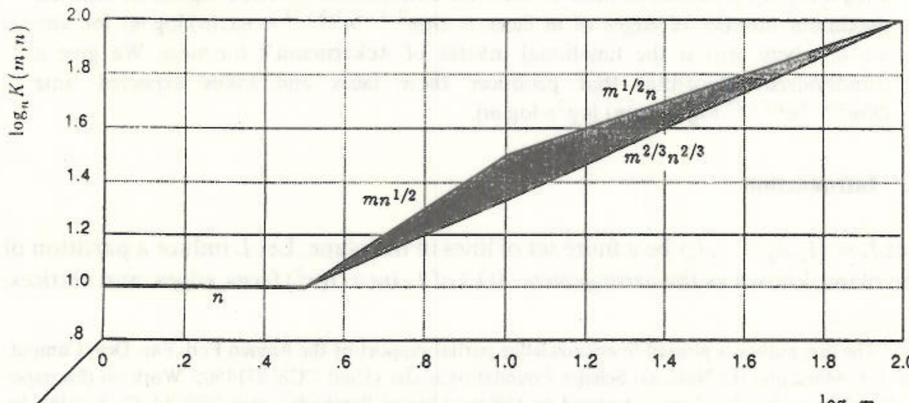

Fig. 1.1. Previous bounds on $K(m, n)$.

[1] With some effort, we can determine for each m and n the optimal choice of δ, and thus obtain a somewhat tighter bound for $K(m, n)$.

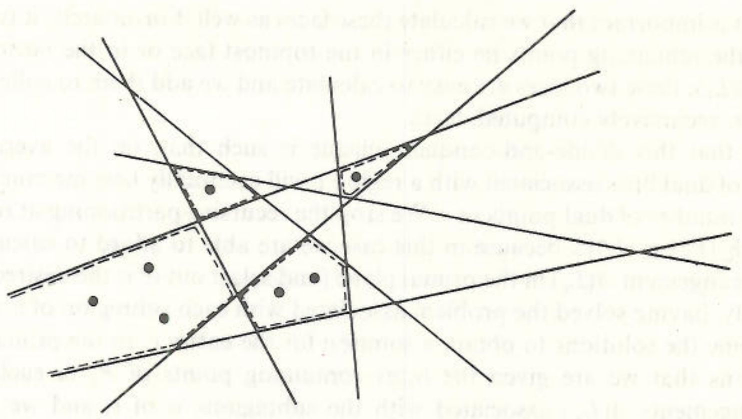

Fig. 1.2. Points designate desired faces.

This bound is slightly weaker than a tight bound $\Theta(m^{2/3}n^{2/3} + n)$, obtained in the companion paper [CEG*]. We nevertheless present this result here because (i) it serves as a simple introduction to the more complex case of segments, (ii) it uses a different proof technique than the one used in [CEG*], and (iii) it leads to an efficient algorithm for computing the desired m faces.

Our approach to the combinatorial probelm is different from previous work on this problem in that it has an algorithmic flavor. We obtain an algorithm for the calculation of m faces in an arrangement of n lines, where each face is designated by specifying an arbitrary interior point in it. In other words, we consider a set L of n lines, $l_1, l_2, \ldots, l_n$, and a set P of m points, $p_1, p_2, \ldots, p_m$, in the plane, and calculate the faces of the arrangement that contain the given points (see Fig. 1.2; for reasons that will become clear later we allow more than one point designating a single face). We construct these faces using the following divide-and-conquer strategy.

It is convenient to describe this strategy in dual space although it is possible to find a fairly natural interpretation of it also in primal space. For this reason, we dualize the points and lines and thus obtain lines p_i^* and points l_j^* in the dual plane. Those lines are referred to as *dual* lines and the points are called *dual* points. The duality transform that we use maps a point p: (a, b) into a line p^*: $y = ax + b$, and a line l: $y = cx + d$ into a point l^*: $(-c, d)$. This duality has the properties that it preserves line-point incidences, and that it maps a point p lying above (resp. below) a line l to a line p^* lying above (resp. below) a point l^*.

Our divide-and-conquer strategy amounts to partitioning the dual plane recursively into convex regions. At each step we have a convex region v in the dual plane, and two sets associated with it: L_v^*—the set of dual points $l_i^* \in v$, and P_v^*—the set of dual lines p_i^* crossing v. We then partition v into convex subregions, w, obtain a corresponding partitioning of L_v^* into subsets L_w^*, and distribute a copy of each line in P_v^* to all sets P_w^* associated with the regions w it crosses.

What is the subproblem, in the primal plane, that a dual region v induces? We have a set of lines $L_v \subseteq L$ and a set of points $P_v \subseteq P$, so the corresponding subproblem is that of calculating the faces of the arrangement $A(L_v)$ that contain points of P_v. But what about the other points? For the success of our recursive scheme it is important that we calculate these faces as well. Fortunately, it turns out that all the remaining points lie either in the topmost face or in the bottommost face of $A(L_v)$; these two faces are easy to calculate and we add them to collection of the other, recursively computed, faces.

Note that this divide-and-conquer scheme is such that, on the average, the number of dual lines associated with a region v will eventually become much larger than the number of dual points in v. We stop the recursive partitioning at regions v for which $|P_v^*| \geq |L_v^*|^2$, because in this case we are able to afford to calculate the entire arrangement $A(L_v)$ in the primal plane (and select out of it the desired faces).

Finally, having solved the problem associated with each subregion of v, we need to combine the solutions to obtain a solution for the entire v. In the primal plane, this means that we are given the faces containing points of P_v in each of the subarrangements $A(L_w)$ associated with the subregions w of v, and we wish to combine (actually, intersect) these faces to obtain the faces containing points of P_v in the full arrangement $A(L_v)$. A major tool that we develop for this purpose is the so-called "combination lemma" which gives a tight upper bound on the maximum combinatorial complexity[2] of the desired faces in terms of the combinatorial complexity of the corresponding faces in the subarrangements (see Lemma 1). We expect this result to have applications to other problems as well.

If we examine the recursion tree that results from this divide-and-conquer strategy, we obtain a structure that is very similar to so-called *partition trees*, originally designed to solve half-plane range searching problems. Indeed, thinking of the dual points as "data" and the dual lines as "queries," we obtain a partition of the set of dual points into disjoint subsets, according to some underlying convex decomposition of the dual plane; this is similar to standard partition trees [EW2], [HW], except that each node v (that is, a region in the dual plane) knows *a priori* all the query lines that require further processing at v (these are the lines in P_v^*—the lines that cross v); we use this information to further partition v, thus making the tree "customized" and easier to "search."

To reiterate, an offshoot of the analysis given in this paper yields a technique for constructing a partition tree for a set of data points and a predetermined set of query lines. Such a tree can then be used to obtain better bounds for batched half-plane range searching when the queries are known in advance (applications include counting the number of intersections between n line segments [GOS], calculating the "signature" of a polygonal curve [O], multiple ray-tracing [SML], etc.).

In our present application, the desired upper bound on $K(m, n)$ is obtained by analyzing the space complexity of the resulting algorithm. The time complexity of the algorithm is roughly a polylogarithmic factor times the upper bound on $K(m, n)$ mentioned above (see Section 3 for a more precise bound). The algorithm is based on a random sampling technique akin to the ε-net method of Haussler and Welzel [HW] and to the random sampling method of Clarkson [Cl]. We obtain a randomized algorithm which almost always terminates, produces the desired output upon termination, and whose expected running time is as stated above.

[2] We use the term "combinatorial complexity" and sometimes just "complexity" for the number of edges bounding some collection of faces.

Next we consider the problem of estimating the maximum number of edges bounding m faces in an arrangement A of n line segments in the plane, and of calculating these faces. This problem is considerably more difficult than the one for lines, because the faces of A are not necessarily convex or simply connected. This makes it harder to process such faces efficiently. Nevertheless, using an intricate extension of our combination lemma (see Lemma 5), we obtain essentially the same bound on the maximum complexity, $R(m, n)$, of m distinct faces in an arrangement of n line segments. More precisely, we prove

$$R(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n) \log m)$$

for any $\delta > 0$, where $\alpha(n)$ is the extremely slowly growing inverse of Ackermann's function. To the best of our knowledge this is the first nontrivial upper bound known for $R(m, n)$. Note that this upper bound almost matches the above-mentioned lower bound on $K(m, n)$. Since trivially $K(m, n) \le R(m, n)$ this implies that our upper bound on $R(m, n)$ is almost tight.

As in the case of lines, our method also yields an algorithm for calculating the designated faces. From a high-level point of view the algorithms for the calculation of the desired faces in arrangements of lines and of line segments are quite similar. Both algorithms employ a key procedure for the following problem:

> given a collection of k points and the faces containing them in each of two subarrangements of the given lines or line segments, calculate the faces containing these points in the arrangement formed by the union (that is, overlay) of the two subarrangements.

In the case of lines this is easy to achieve efficiently because each face is convex. In the case of line segments this is more difficult because of the potentially highly irregular shapes of individual faces. We present an efficient line-sweeping method for merging faces containing k given points in line segment arrangements whose complexity is $O((t + k) \log (t + k))$, where t is the total complexity of input and output faces. Applying this merge recursively, we can calculate the required faces in (randomized expected) time which is within a polylogarithmic factor of the bound on $R(m, n)$. An interesting consequence of our merging procedure is that a single face in an arrangement of n line segments in the plane can be constructed deterministically in time $O(n\alpha(n) \log^2 n)$. This problem arises in certain two-dimensional motion-planning problems in robotics, and has been previously studied in [PSS]. A companion paper, [GSS], extends the line-sweep technique of this paper to the calculation of a single face in arrangements of more general curves.

The technique used in this paper is one of several related approaches that were developed recently, all of which use ε-nets and random sampling as basic tools. This paper uses ε-nets to partition the given lines (or line segments) into a fixed number of (disjoint) subsets so that each subset interacts only with a relatively small number of the given points. These interactions are taken care of recursively. In constrast, we might try to partition the given points into (disjoint) subsets, each interacting with only a small number of the given lines (or line segments). This alternative approach has been studied in a companion paper [CEG*]. It yields tight combinatorial results for the case of lines, and can be used to obtain upper bounds for the complexity of many faces, and for the total number of incidences with many points, in arrangements of other types of curves, and also in arrangements in higher dimensions. While the point-partitioning approach of [CEG*] is mainly combinatorial, the line-partitioning approach used here also yields efficient randomized algorithms. Another advantage of the line-partitioning method over the point-partitioning one is that it extends to line segments (which have not been amenable to the other approach yet). In addition, our "dual" aproach has turned out to be better than the "primal" one in analyzing the complexity of many cells in arrangements of planes or hyperplanes, as is demonstrated in another companion paper [EGSh].

The paper is organized as follows. In Section 2 we analyze the combinatorial complexity of many faces in an arrangement of lines. This analysis is explained in terms of an algorithm that constructs the faces; its implementation is discussed in Section 3. In Sections 4 and 5 we analyze the combinatorial complexity of many faces in an arrangement of line segments, and in Sections 6 and 7 we discuss the implementation of the algorithm implicitly described in the combinatorial analysis. Concluding remarks and open problems are given in Section 8.

2. The Complexity of Many Faces in an Arrangement of Lines

Let $L = \{l_1, l_2, \ldots, l_n\}$ be a set of n lines in the plane, and let $A = A(L)$ denote its arrangement as defined in the introduction. Let $P = \{p_1, p_2, \ldots, p_m\}$ be a set of m given points that do not lie on any of these lines. Consider the problem of calculating all faces of A that contain the points p_i, producing each such face just once, even if it contains several of these points (see Fig. 1.2). We seek an algorithm for solving this problem with a small worst-case space complexity. This space complexity will serve as an upper bound on the maximum number of edges bounding any m faces in any arrangement of n lines in the plane. As it turns out, the expected time complexity of our (randomized) algorithm is within a $\log n$ factor (or a $\log n \log m$ factor if $m = O(\sqrt{n})$) of its worst-case space complexity, so we also get a nearly time-optimal (although randomized) algorithm for the calculation of the faces.

We assume that initially no two of the given points lie in the same face of A. The algorithm that we present below uses a divide-and-conquer approach and each recursive step involves some subset L' of the lines l_i and some subset P' of the points p_j. Since L' is only a subset of L it thus can happen that in the arrangement formed by L' two or more points of P' fall into the same face. In this case we will want the algorithm to maintain this face just once, and have pointers to it from each of the points contained in it. To reflect this potential duplication, we denote by $\bar{K}(m, n)$ the maximum complexity of the faces in an arrangement of n lines that contain m given points (counting each face just once). Unlike $K(m, n)$ which is defined only if $m \le \kappa(n)$, $\bar{K}(m, n)$ is defined for all integers $m \ge 0$, $n > 0$. However, when both functions are defined, we cleary have $\bar{K}(m, n) = K(m, n)$.

We next describe the algorithm for calculating the required faces. The discussion ignores implementation issues (addressed in Section 3) and instead concentrates on combinatorial problems that arise.

First we dualize the line l_i to points l_i^* and the points p_i to lines p_i^*, using the duality transformation defined in the introduction. This gives a set L^* of n points and a set P^* of m lines in the dual plane. To process the dual points and lines, we choose some constant integer $r > 0$, and select a random sample of r of the dual lines p_j^*. When we draw the arrangement of these lines in the dual plane and triangulate each of the faces of this arrangement we obtain a total of $M = O(r^2)$ triangles. The ε-net theory of Haussler and Welzl [HW] or, alternatively, the random sampling lemma of Clarkson [C1] imply that, with high probability, the interior of each of these triangles intersects at most $(cm/r) \log r$ dual lines, for some absolute constant c independent of r and m. The sample is called an ε-net if it has the property that any triangle in the dual plane not meeting in its interior any of the lines p_v^* in the sample intersects at most $(cm/r) \log r$ dual lines in P^*. The ε-net theory implies that (i) such a sample always exists (which suffices for the combinatorial analysis given in this section), and (ii) a random sample of r lines of P^* is an ε-net with high probability (which can be made arbitrarily close to 1 if we choose c large enough). The second property is important for the algorithm that calculates the faces, as given in the following section.

We now divide our problem into subproblems, each associated with one of the M triangles v in the dual plane. We associate with v the subset L_v^* of the dual points in L^* that lie inside v, and the subset P_v^* of the dual lines in P^* that intersect v. In what follows we denote the cardinality of L_v^* by n_v and the cardinality of P_v^* by m_v.[3]

The subproblem associated with v, in the primal plane, is to compute the faces of $A(L_v)$ that contain points of P_v (where L_v is the subset of lines in L whose dual points belong to L_v^*, and P_v is the subset of points in P whose dual lines belong to P_v^*). Note that this subproblem may be "incomplete," in the sense that we ignore faces that contain points in $P - P_v$. We address this problem shortly below.

Each of the M subproblems is solved recursively. That is, we take a random sample of r lines from the corresponding set P_v^*, and construct and triangulate their arrangement. For each triangle w in this arrangement we obtain, as above, a subproblem involving the subset L_w^* of the dual points of L_v^* contained in w and the subset P_w^* of dual lines of P_v^* that intersect the interior of w. (A schematic representation of this process is shown in Fig. 2.1; the arrangement is formed by $r = 2$ lines, thus it is not necessary to triangulate further the four faces that result from this partitioning.)

This process is continued recursively, but not all the way, until just one or no point or line remains. Whenever we reach a subproblem associated with a triangle v for which $m_v \geq \kappa(n_v)$, we stop the process and solve the subproblem directly. That is, we undo the dualization to obtain L_v from L_v^* and P_v from P_v^*. Then we

[3] The reader is advised to note that we consistently use the letters L and n in association with the primal lines (and therefore with the dual points) and that P and m are used in connection with primal points (and thus dual lines).

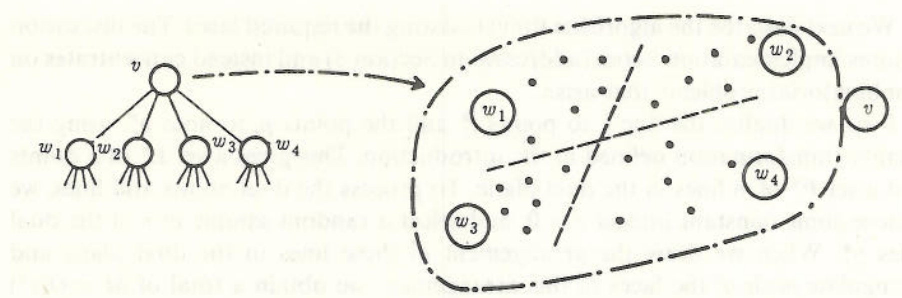

Fig. 2.1. Partition tree and corresponding decomposition.

construct the arrangement $A(L_v)$ (in the primal plane, locate[4] in it each of the points of P_v, and report the faces of $A(L_v)$ that contain them (each face only once). The total number of edges bounding these faces (which is proportional to the space needed to store them) is at most $O(n_v^2) = O(m_v)$. In passing we mention that the time-complexity of this step is at most $O(n_v^2 + m_v \log n_v) = O(m_v \log n_v)$ using the arrangement construction algorithm of [EOS] and the optimal point-location structure of [EGSt]. Another case where we stop the recursion is when $P_v^* = \varnothing$. In this case we do not have to bother constructing $A(L_v)$ since there are no points for which faces need to be calculated.

We obtain the required collection of faces of $A(L)$ using the following "merging" procedure. For each point $p_i \in P$, let $F(p_i)$ denote the face of $A(L)$ that contains p_i, and, for each triangle v in the dual partitioning, let $F_v(p_i)$ denote the face of the arrangement $A(L_v)$ that contains p_i. (Recall that such a face will be shared by all points that lie in it.) For points $p_i \in P_v$, the face $F_v(p_i)$ is available recursively. Let $Q_v = P - P_v$, which is the set of face-designating points that the subproblem at v so far has ignored. Note that Q_v is the set of points whose dual lines miss the interior of v. By duality, each of these points p_i lies either above all the lines in L_v (that is, in the topmost face F_v^+ of $A(L_v)$) or below all of them (in the bottommost face F_v^-). This is illustrated in Fig. 2.2. These two faces together have at most $n_v + 2$ edges, and they can be constructed in time $O(n_v \log n_v)$ (see, e.g., [PS]). As required, we store each of these two faces just once, and maintain a pointer from each point $p_i \in Q_v$ to either F_v^+ or F_v^- whichever contains it. Hence, at this stage all the faces of $A(L_v)$ containing points of P are accounted for. We note that the topmost and bottommost faces in $A(L_v)$ must be constructed even if $m_v = 0$ and we stop the recursion at v.

Now the merging step proceeds as follows. Our goal is to compute $F(p_i)$ for all $p_i \in P$. Let $v_1, v_2, \ldots, v_M$ be the triangles formed by the partitioning (recall that $M = O(r^2)$). For each point $p_i \in P$, and for each triangle v_j, either $p_i \in P_{v_j}$ or $p_i \in Q_{v_j}$. In the former case, $F_{v_j}(p_i)$ is calculated recursively at v_j. In the latter case, $F_{v_j}(p_i)$ is

[4] *Locating* a point in an arrangement means to find the face (or edge or vertex) of the arrangement that contains the point. It is a fairly common term in computational geometry which, among other things, considers data structures that facilitate fast point-location queries (see, e.g., [EGSt]).

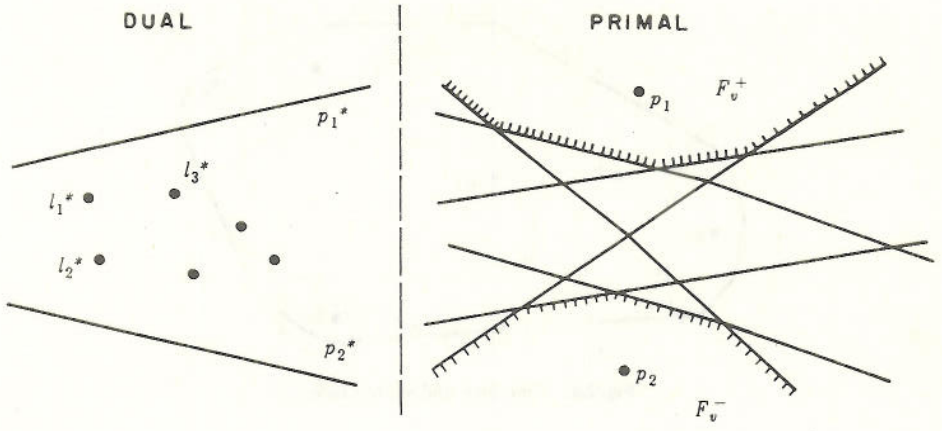

Fig. 2.2. Dual and primal plane.

either $F_{v_j}^+$ or $F_{v_j}^-$, which are also assumed to be available. Since $L = L_{v_1} \cup \cdots \cup L_{v_M}$, it is clear that

$$F(p_i) = \bigcap_{v_j} F_{v_j}(p_i).$$

This intersection is plainly a convex polygon that contains p_i and the number of its edges is at most the sum of the number of edges of the intersected faces.

However, a, major technical problem arises now. Since some of the faces $F_{v_j}(p_i)$ may be shared by other points, we need to avoid duplicate processing and counting of the same face for each point it contains, or else our algorithm might have unacceptably high time-complexity and our upper bound on the total number of edges will be annoyingly loose. A typical case where duplicate processing of this sort can slow down the algorithm is depicted in Fig. 2.3.

A solution of this problem is provided by the following technical lemma, which we refer to as the "combination lemma (for lines)."

Lemma 1. *Let $p_1, p_2, \ldots, p_k$ be points in the plane, and let $\{B_1, B_2, \ldots, B_s\}$ and $\{R_1, R_2, \ldots, R_t\}$ be collections of s "blue" and t "red" (topologically) open convex polygons that satisfy the following three properties:*

(i) *The blue (red) polygons are pairwise disjoint and the total number of blue (red) edges is β (ρ).*

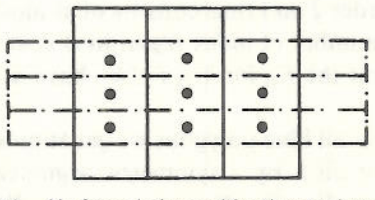

Fig. 2.3. Six faces designated by three points each.

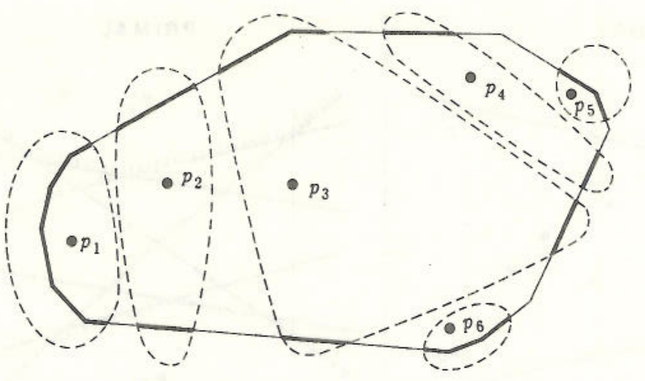

Fig. 2.4. One blue and six red faces.

(ii) *Each point p_i is contained in a blue polygon B_{s_i} and in a red polygon R_{t_i}.*
(iii) *If for each $1 \le i \le k$ we define $E_i = B_{s_i} \cap R_{t_i}$, then $E_i \ne E_j$ if $i \ne j$.*

Then the total number of sides of the E_i is at most $\beta + \rho + 4k - 2s - 2t$.

Proof. Take one of the blue polygons $B = B_j$, and assume that it contains k_j points, say $p_1, p_2, \ldots, p_{k_j}$. Each of these points p_i lies in a different red polygon R_{t_i}. We consider the k_j cells $E_i = B \cap R_{t_i}$ for $1 \le i \le k_j$, which are convex polygons (see Fig. 2.4). To give an upper bound on the number of blue edges of the E_i we define for an edge e of B the intersection of e with R_{t_i} and denote it by e_i. Now write down the cyclic sequence of the nonempty e_i in clockwise order around the boundary, ∂B, of B. We observe the following two properties:

(i) The sequence of indices (red polygons intersecting ∂B) contains no cyclic scattered subsequence of the form $i..j..i..j$.
(ii) If two consective indices (red polygons) are the same, then the edges of B in both elements are different.

To prove (i) just note that if such a case were to arise, then we could connect the first and third edges and the second and fourth edges by two straight segments lying respectively inside the red convex polygons R_{t_i} and R_{t_j}. Both segments have their endpoints on ∂B which implies by the Jordan curve theorem that they intersect. This is a contradiction to $R_{t_i} \cap R_{t_j} = \emptyset$ if $i \ne j$. The claim (ii) follows from the fact that a single edge of B intersects a red polygon in a connected piece—after all both the blue edge and the red polygon are convex.

Ignoring repetitions of indices, (i) implies that the cyclic sequence is a Davenport–Schinzel cycle of order 2 and thus consists of at most $2k_j - 2$ edges (see [ES] for details). By (ii), the number of index repetitions is at most $|B|$, the number of edges of B. It follows that the E_i, for $1 \le i \le k_j$, have at most $|B| + 2k_j - 2$ blue edges.

If we take the sum over all blue polygons we get at most $\beta + 2k - 2s$ blue edges bounding the cells E_i for all i. By a symmetric argument we can show that the number of red edges bounding the E_i is at most $\rho + 2k - 2t$. It follows that the total number of edges of the E_i (each edge either blue or red) is at most $\beta + \rho + 4k - 2s - 2t$. □

By applying Lemma 1 a fixed number of times ($M - 1$ times to be precise), it follows that the overall complexity, K,of all faces $F(p_i)$ can be bounded from above by

$$\sum_{j=1}^{M} (K_{v_j} + (n_{v_j} + 2)) + M \sum_{j=1}^{M} 4m_{v_j} = \sum_{j=1}^{M} (K_{v_j} + O(n_{v_j})) + M \sum_{j=1}^{M} O(m_{v_j}),$$

where K_{v_j} is the number of edges counted at v_j, and the second subterm of the first sum arises from the faces $F_{v_j}^{+}$ and $F_{v_j}^{-}$. But $\sum_j n_{v_j} = n$ and $M \sum_{j=1}^{M} O(m_{v_j}) = O(m)$, since r and thus M is a constant. We can thus rewrite the recurrence relation as

$$K = \sum_{j=1}^{M} K_{v_j} + O(m + n).$$

To solve this recurrence relation, let $\bar{K}(m, n)$ denote as above the maximum complexity of the collection of faces that arise for m points in an arrangement of n lines. Then we have

$$\bar{K}(m, n) \le \begin{cases} 0 & \text{if } m = 0, \\ am & \text{if } m \ge \kappa(n), \\ \sum_{j=1}^{M} \bar{K}(m_i, n_i) + bm + b'n & \text{if } m < \kappa(n), \end{cases}$$

for some constants $a, b, b' > 0$ (note that b, b' depend on r but a does not), where M, m_i, and n_i satisfy the following three conditions (which are immediate from our construction):

$$M = O(r^2), \tag{I}$$

$$\sum_{i=1}^{M} n_i = n, \tag{II}$$

and

$$\text{for each } i \text{ we have} \quad m_i \le \frac{cm}{r} \log r \quad \text{for some constant} \quad c > 0. \tag{III}$$

Under these constraints we have

Lemma 2. $\bar{K}(m, n) \le Dm^{2/3-\delta}n^{2/3+2\delta} + Am + Bn \log m$, *for any* $\delta > 0$, *where the coefficients* A, B, D *depend on* δ.

Proof. We first note that at each level of the recursion m decreases by a factor $\Omega(r \log r)$, for a constant r, and thus the recursion has only $O(\log m)$ levels. The sum of the n_v, over all nodes v at the same recursion level, is clearly n, so that the total contribution of the rightmost term, $b'n$, is at most $O(n \log m)$. We thus ignore this term in the recurrence relation for $\bar{K}$ and prove that the solution to the modified recurrence satisfies $\bar{K}(m, n) \le Dm^{2/3-\delta}n^{2/3+2\delta} + Am$ for any $\delta > 0$.

Fix $0 < \delta < 1/6$ and choose $r = r(\delta) > 0$ sufficiently large (how large will be apparent from the analysis below).

The bound is trivial for $m = 0$. If $m \ge \kappa(n)$, then $\bar{K}(m, n) \le am$ plainly satisfies the required inequality, assuming $A \ge a$. It follows that the bound is trivially true for constant n since $m < \kappa(n)$ only if m is also at most a constant (we need this observation only for $n \le 1$). So assume $m < \kappa(n)$. In this case

$$m = m^{2/3-\delta}m^{1/3+\delta} \le m^{2/3-\delta}n^{2/3+2\delta}, \qquad (*)$$

assuming $n \ge 2$. By induction hypothesis we then have

$$\bar{K}(m, n) \le \sum_{i=1}^{M} (Dm^{2/3-\delta}n_i^{2/3+2\delta} + Am_i) + bm.$$

By properties (III) and (I) we have

$$\sum_{i=1}^{M} m_i \le \frac{cMm \log r}{r} \le (c_1 r \log r)m$$

for some constant c_1 independent of r. Hence

$$\bar{K}(m, n) \le D \cdot \sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3+\delta} + (Ac_1 r \log r + b)m.$$

Thus, using $(*)$ and putting $d = Ac_1 r \log r + b$, we obtain

$$\bar{K}(m, n) \le D \cdot \sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3+2\delta} + dm^{2/3-\delta}\, n^{2/3+2\delta}.$$

But

$$\sum_{i=1}^{M} m_i^{2/3-\delta}n_i^{2/3+2\delta} \le \left(\frac{cm \log r}{r}\right)^{2/3-\delta} \sum_{i=1}^{M} n_i^{2/3+2\delta}$$

which, by the Hölder inequality, does not exceed

$$\left(\frac{cm \log r}{r}\right)^{2/3-\delta} n^{2/3+2\delta}M^{1/3-2\delta} = O\!\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}}\, m^{2/3-\delta}n^{2/3+2\delta}\right).$$

Hence

$$\bar{K}(m, n) \leq \left(D \cdot O\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}} \right) + d \right) m^{2/3-\delta} n^{2/3+2\delta}.$$

But since $\delta > 0$, it is clear that if r is chosen sufficiently large so that

$$O\left(\frac{(\log r)^{2/3-\delta}}{r^{3\delta}} \right) < \tfrac{1}{2},$$

say and if D is taken to be sufficiently large so that $D/2 > d = Ac_1 r \log r + b$, then the expression in the bracket will be less or equal to D, thus establishing the asserted inequality. □

Theorem 3. *The total number of edges, $K(m, n)$, bounding m distinct faces in an arrangement of n lines is at most $O(m^{2/3-\delta} n^{2/3+2\delta} + n)$ for any $\delta > 0$ (where the constants of proportionality depends on δ).*

Proof. Recall that when both functions are defined, we have $K(m, n) = \bar{K}(m, n)$. For $m \leq n^{1/2}$ the asserted bound follows immediately from the results of [Ca] mentioned in the introduction. For $n^{1/2} < m \leq \kappa(n)$ it is easily checked that the term $O(m^{2/3-\delta} n^{2/3+2\delta})$ dominates $O(m)$ and $O(n \log m)$ in the bound of Lemma 2. □

Remarks. (1) The preceding bounds imply that $K(m, n) = O(m^{2/3-\delta} n^{2/3+2\delta})$ for any $\delta > 0$, provided neither m nor n is too small.

(2) Our result leaves a small gap between our upper bound and the lower bound of $\Omega(m^{2/3} n^{2/3} + n)$obtained in [EW1]. An alternative, point-partitioning approach as presented in a companion paper [CEG*] closes this gap and shows that $\Theta(m^{2/3} n^{2/3} + n)$ is the real bound.

(3) A related result is that of Szemerédi and Trotter [ST] who give a tight bound, $\Theta(m^{2/3} n^{2/3} + n)$, on the maximum sum of the degrees of m vertices in an arrangement of n lines. There does not appear to be an easy way to extend the proof technique of [ST] to the case of faces.

3. Calculating Many Faces in an Arrangement of Lines

To complete our analysis of line arrangements, we turn to the implementation of the algorithm outlined in section 2 which constructs the faces in an arrangement of n lines $l_1, l_2, \ldots, l_n$ that contain m given points $p_1, p_2, \ldots, p_m$. Let $T(m, n)$ denote the expected time needed for this task using the approach described in Section 2. We have already noted that at the bottom of the recursion we have $T(m, n) = O(m \log n)$ if $m \geq \kappa(n)$ and $T(m, n) = O(n \log n)$ if $m = 0$ (this is the time needed for the calculation of the two corresponding faces F_v^+ and F_v^-).

As for the general merging step of the algorithm, we need to calculate, for each $p_i \in P$, the face $F(p_i)$ of the arrangement $A(L)$ that contains p_i; this face is the intersection of the M faces $F_{v_j}(p_i)$, $j = 1, 2, \ldots, M$, where $v_1, \ldots, v_M$ are the triangles obtained by the partitioning step, and where $F_{v_j}(p_i)$ is the face of $A(L_{v_j})$ containing p_i. Recall that a major technical difficulty in the analysis of the space complexity of the algorithm, given in the preceding section, was to avoid duplicate access to a face in some $A(L_{v_j})$ that is shared by several of the points. To overcome this difficulty algorithmically, we proceed as follows. For expository reasons we assume $M = 2$, so that we need to intersect only two faces around each p_i.

(i) With each p_i we associate the pair $(F_{v_1}(p_i), F_{v_2}(p_i))$. Regard two points as equivalent if they have the same associated pair of faces. The equivalence classes can be constructed in time $O(m \log m) = O(m \log n)$ by sorting the face-pairs and removing repetitions. This also yields a representative point for each equivalence class; we clearly need to calculate $F(p_i)$ only for these representative points.

(ii) For each representative point p_i, we need to calculate the intersection, E, of the two convex polygons, $B = F_{v_1}(p_i)$ and $R = F_{v_2}(p_i)$, in time that mainly depends on the number of edges of E. This is accomplished using the following "ray-shooting" procedure. First we find a starting point z on ∂E by shooting a horizontal ray from $p_i \in E$ and finding the nearest of its intersections with ∂B and ∂R. We next traverse the boundary of E in counterclockwise direction from z as follows. Suppose we have reached some point x on some edge e of ∂B. We shoot a ray from x along e (so that B lies to the left of the ray) and find its intersection, x', with ∂R. If e ends before x', then we turn at the endpoint of e to the adjacent edge, e', along ∂B and repeat shooting along e' toward ∂R. Otherwise, we turn at x' to ∂R in counterclockwise direction, and shoot along the new edge toward ∂B. Repeating this process, we will eventually return to z, thereby completely tracing the boundary of δE. (Figure 3.1 illustrates this process.) Since both faces, B and R, are convex each ray shooting query can be carried out in time $O(\log n)$ (see [CD]). Thus, the calculation of $F(p_i)$ can be accomplished in time $O(|F(p_i)| \log n)$, $|F(p_i)|$ being the number of edges bounding $F(p_i)$.

In general, that is, if $M > 2$, we apply the merging process $M - 1$ times to take into account all M subproblems generated at v. Since M is a constant depending only on r, the sample size, all faces $F(p_i)$ can be obtained in time $O((K(m, n) + m) \log n)$.

Finally, we consider the overhead of the divide part of our recursion. At each recursive step we take a random sample of size r of the current set of dual lines,

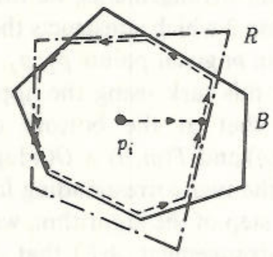

Fig. 3.1. Tracing the boundary of the intersection.

construct and triangulate their arrangement, split the current set of dual points among these triangles, and determine for each triangle which of the current dual lines p_i^* it intersects. Each triangle gives rise to a subproblem which is passed the points that fall into the triangle as well as the lines that intersect it. Each step either takes constant (randomized) time or time linear in n_v or m_v.

At this point, we would like to verify that the random sample is indeed an ε-net. The reason for this is that, although we know that this will be the case with high probability, we have to perform random sampling at many steps during the recursion, and the small probabilities of failure may add up to a nonnegligible amount. To rectify this, we simply check whether the number of dual lines cutting a subtriangle is sufficiently small, for all subtriangles resulting from the partitioning. If not, we discard the sample and try another random selection of r dual lines. In a constant expected number of trials we will obtain an ε-net.

We can therefore obtain the following recurrence formula for $T(m, n)$, the expected time needed for m points and n lines. For $m < n^2$ we have

$$T(m, n) \le \sum_{i=1}^{M} T(m_i, n_i) + O((\bar{K}(m, n) + m + n) \log n),$$

where m_i, n_i, and M satisfy conditions (I), (II), and (III) of the analysis in Section 2. For $m \ge n^2$ we have

$$T(m, n) = O(m \log n),$$

and for $m = 0$ we have

$$T(m, n) = O(n \log n).$$

Using the bounds on $\bar{K}(m, n)$ and $K(m, n)$ obtained above, we can derive the following bound on $T(m, n)$; the proof is a straighforward generalization of the proofs of Lemma 2 and Theorem 3 and is left to the reader.

The following theorem also derives a bound on the maximum space complexity, $S(m, n)$, required by the algorithm. This is easily seen to be proportional to the space used along a single path in the recursion tree, which is easily seen to be $O(m^{2/3-\delta}n^{2/3+2\delta} + n \log m)$. We thus summarize.

Theorem 4. *The expected time complexity of the above randomized algorithm for computing m distinct faces in an arrangement of n lines is*

$$T(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} \log n + n \log n \log m)$$

for any $\delta > 0$. The space required by the algorithm is

$$S(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n \log m)$$

for any $\delta > 0$.

Remarks. (1) If m is much smaller than n^2, the $\log n$ factor in the first term in the time bound given above can be dropped, simply by taking a slightly larger value of δ.

(2) The ray shooting technique used in the above algorithm does not seem to generalize to the more complicated task of constructing m faces in an arrangement of n line segments, which is what we study in Sections 4–7. The alternative merging technique that we use for line segments, described in Section 7, can also be applied to the simpler case at hand. However, we have chosen to present here the ray shooting technique because of its relative simplicity in the case of convex polygons.

4. The Complexity of Many Faces in an Arrangement of Line Segments

This section extends the analysis given in Section 2 to the case of line-segment arrangements, that is, we consider the problem of estimating the maximum combinatorial complexity, $R(m, n)$, of m faces in an arrangement of n line segments in the plane. In contrast to the case of lines where all faces are convex, a face in a line-segment arrangement is not necessarily convex and need not even be simply connected (see Fig. 4.1). Because of the nonconvexity of faces, there is no reason why the maximum number of edges bounding a single face should be at most n. Indeed, the total number of edges bounding a single face can be as large as $\Omega(n\alpha(n))$, where $\alpha(n)$ is the inverse Ackermann's function, and this bound is tight in the worst case, as was shown in [HS], [PSS], and [WS]. Lines are a special case of line segments, which implies $R(m, n) \geq K(m, n)$. Thus, the lower bound of [EW1] for line arrangements extends to line segments, which, combined with the result of [WS], yields $R(m, n) = \Omega(m^{2/3}n^{2/3} + n\alpha(n))$.

In spite of the technical difficulties caused by the boundedness of line segments, we obtain an upper bound on $R(m, n)$ that is roughly the same as the bound on $K(m, n)$ obtained in Section 2. Again, the bound will be derived from an analysis of the space complexity of an algorithm for calculating m such faces. In Sections 6 and 7 we show how to implement the algorithm so that the calculation of m faces in an

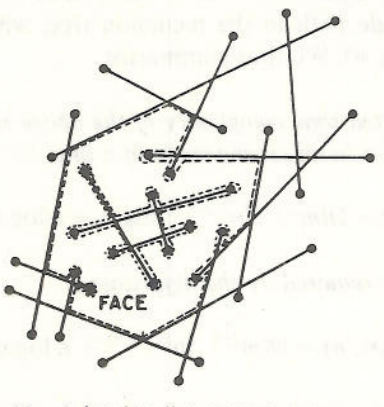

Fig. 4.1. A face in a line-segment arrangement.

arrangement of n line segments takes time that is only slightly more than our upper bound on $R(m, n)$.

Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of n line segments in the plane, and let $A = A(S)$ be their arrangement. Let $P = \{p_1, p_2, \ldots, p_m\}$ be a set of points that we use to designate faces of A, and consider the problem of calculating the faces of A that contain the points p_i. Since we may reach a situation where a face of a subarrangement contains more than one point, we use the auxiliary notation $\bar{R}(m, n)$ to denote the maximum complexity of the faces in an arrangement of n segments that contain m given points (accounting for each face only once). Clearly, $\bar{R}(m, n) = R(m, n)$ whenever both functions are defined.

For each i let l_i denote the line containing s_i, and define $L = \{l_i | 1 \leq i \leq n\}$. Apply the dual construction given in Section 2 to the lines in L and to the points in P. Thus we recursively partition the dual plane into triangles v, and associate with each v the subset L_v of lines in L whose dual points lie in v, and the subset P_v of points in P whose dual lines intersect v. Implicitly through L_v, v also represents the set of line segments $S_v = \{s_i | l_i \in L_v\}$. Consistent with the notation in Section 2, we define $m_v = |P_v|$ and $n_v = |S_v|$. The subproblem associated with v is to compute the faces of $A(S_v)$ that contain points of P_v.

When $m_v \geq \kappa(n_v)$ we stop the recursion at v, pass back to the primal plane, construct there the arrangement $A(S_v)$ of the n_v line segments s_i in S_v, and collect the required faces of $A(S_v)$ that contain the points in P_v. The space complexity of the entire $A(S_v)$, and thus also of the faces in question, is $O(n_v^2) = O(m_v)$.[5] We also stop the recursion if $m_v = 0$. In this case, as is argued shortly, we only need to construct the unbounded face of the associated line-segment arrangement. How to implement this efficiently is described in Sections 6 and 7 below (see Theorem 11). However, in terms of space complexity, we already know that this face has only $O(n_v \alpha(n_v))$ edges [PSS].

(Note that an important feature of the construction is that in the dual plane we use the dual points l_i^* of the (unbounded) lines l_i. We thus ignore the fact that we have to consider only the portions s_i of these lines. However, when we pass back to the primal plane, we always process the line segments s_i rather than the lines that contain them.)

We now proceed to the discussion of how the relevant faces of $A(S)$ are obtained if $0 < m < \kappa(n)$. As in the case of lines, we compute $Q_v = P - P_v$, for each resulting triangle v. Every point in Q_v either lies above all lines in L_v or below all these lines. In any case, those points lie in the unique unbounded face, F_v^∞, of $A(S_v)$. As mentioned above, this face is bounded by $O(n_v \alpha(n))$ edges. (If the given collection S also contains unbounded rays of lines, we may have to consider two unbounded faces of $A(S_v)$, as in the case of lines.)

The main difficulty lies of course in merging (or rather intersecting) the recursively available faces of the subarrangements $A(S_{v_1}), \ldots, A(S_{v_M})$ associated with the triangles $v_1, v_2, \ldots, v_M$, to get the desired faces of $A(S)$. For a particular

[5] Indeed, the number of vertices, edges, and faces of $A(S_v)$ is proportional to the number of intersecting line-segment pairs which is, of course, at most $\binom{n_v}{2}$.

point, $p \in P$, this means that we construct $F(p)$ by intersecting the faces $F_{v_j}(p)$ that contain p in the subarrangements associated with the triangles v_j. Our goal is to obtain an appropriate generalization of the combination lemma for lines (Lemma 1) that will enable us to bound in a similar manner the total complexity of the faces $F(p_i)$. This generalization is quite complicated and is described in detail in the following section. This section continues with explaining the result of this generalization and using it to complete the combinatorial analysis of many faces in line-segment arrangements.

For the remainder of this section (as well as for Sections 5–7) we define a *polygon* as an open region in the plane that can occur as a face in a line-segment arrangement. Thus, a polygon is neither necessarily convex nor simply connected. The boundary of a polygon consists of one or more connected components called *contour cycles*. We think of a contour cycle as a Jordan curve that may touch but cannot cross itself. In particular, if an edge lies in the interior of the closure of the polygon (it bounds the polygon on both sides), then the contour cycle has two portions that touch along the edge (see Fig 4.1). When we count the number of edges of a polygon we count each such edge portion twice. A vertex of the polygon is *reflex* if the inside angle at this vertex exceeds π. With these definitions we have the following generalization of Lemma 1—the proof of this lemma, called the "combination lemma for line segments," is given in Section 5.

Lemma 5. *Let $p_1, p_2, \ldots, p_k$ be k points in the plane, and let $\{B_1, B_2, \ldots, B_s\}$ and $\{R_1, R_2, \ldots, R_t\}$ be collections of "blue" and "red" polygons that satisfy the following three properties*:

(i) *The blue (red) polygons are pairwise disjoint, the total number of blue (red) edges is $\beta(\rho)$, and the total number of reflex vertices is r.*
(ii) *Each point p_i is contained in a blue polygon B_{s_i} and a red polygon R_{t_i}.*
(iii) *If for each $1 \leq i \leq k$ we define E_i to be the connected component of $B_{s_i} \cap R_{t_i}$ that contains p_i (see Fig. 5.3), then $E_i \neq E_j$ if $i \neq j$.*

Then the total number of edges of the E_i is at most $\beta + \rho + O(k) + O(r)$.

Using Lemma 5, we can complete the analysis of the merge step. Applying Lemma 5 a constant number of times we conclude that the overall complexity of the faces $F(p_i)$ is at most the sum of the complexities of all the faces $F_{v_j}(p_i)$, $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, M$, plus $O(m) + O(r)$, where r is the number of reflex vertices in all faces $F_{v_j}(p_i)$. But each such reflex vertex must be an endpoint of one of the n segments in S, so $O(r) = O(n)$. In addition to the complexity of the recursively available faces $F_{v_j}(p_i)$ we need to take into account the total number of edges bounding the unbounded faces $F_{v_j}^{\infty}$, which is at most

$$\sum_{j=1}^{M} O(n_{v_j}\alpha(n_{v_j})) = O(n\alpha(n)).$$

We thus obtain the following recurrence formula for $\bar{R}(m, n)$:

$$\bar{R}(m, n) \leq \begin{cases} 0 & \text{if } m = 0, \\ am & \text{if } m \geq \kappa(n), \\ \sum_{i=1}^{M} \bar{R}(m_i, n_i) + bm + b'n\alpha(n) & \text{if } m < \kappa(n), \end{cases}$$

for some constants $a, b, b' > 0$, where m_i, n_i, and M satisfy conditions (I)–(III) stated in Section 2. A proof that is almost identical to the proof of Lemma 2 (which is therefore omitted) implies the following solution of the recurrence relation.

Lemma 6. $\bar{R}(m, n) \leq Dm^{2/3-\delta}n^{2/3+2\delta} + Am + Bn\alpha(n) \log m$, *for any* $\delta > 0$, *where the coefficients* A, B, D *depend on* δ.

Remark. A major feature of Lemma 5 (and of its simpler variant Lemma 1) is that the terms β and ρ appear with multiplicative constant 1 in the bound for the total combinatorial complexity of the E_i. This ensures that the recurrence formula for $\bar{R}(m, n)$ given above involves the term $\sum_{i=1}^{M} \bar{R}(m_i, n_i)$ with multiplicative constant 1. This is essential for obtaining the bound stated in Lemma 6.

If $m \leq \kappa(n)$ (that is, at most one point per face in the original arrangement is specified), then

$$m \leq m^{2/3-\delta}n^{2/3+2\delta},$$

so that we can drop the second term from the bound in Lemma 6. We thus conclude with the main result of this section.

Theorem 7. *The maximum number of edges of m distinct faces in an arrangement of n line segments is*

$$R(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n) \log m)$$

for any $\delta > 0$.

Remark. It is easy to check that $\bar{R}(m, n) = O(m^{2+\delta} + n\alpha(n) \log m)$, for any $\delta > 0$, also satisifies the recurrence relation derived for $\bar{R}$. This constitutes a weak generalization of Canham's theorem for lines [Ca] to the case of line segments.

5. Proof of the Combination Lemma for Line Segments

In this section we provide a proof of Lemma 5, the combination lemma for line segments. This is the crucial lemma in the analysis of the complexity of many faces in a line-segment arrangement presented in Section 4. We proceed by considering the interaction between a blue and a red polygon, a blue polygon and many red polygons, and finally many blue and many red polygons. The results in this section have a topological and combinatorial flavor and add up to a proof of Lemma 5.

The main concept in this section is that of a *polygon* which is defined general enough so that every face in a line-segment arrangement passes as a polygon. As mentioned in Section 4, a polygon is thus connected but not necessarily simply connected, and its boundary consists of connected components which we call *contour cycles*. We can avoid the technical difficulty caused by the fact that a connected component of the boundary need not be a simple Jordan curve,[6] if we replace each line segment by a rectangle of sufficiently small width. For small enough widths we get the same intersection pattern for the rectangles as for the line segments, and a face (a connected component of the plane minus the union of all rectangles) is now bounded by contour cycles that are simple Jordan curves. For technical reasons we direct each contour cycle so that the polygon it bounds lies to its left. Thus, a contour cycle that delimits a hole of the polygon is directed in clockwise order whereas the outside contour cycle (if it exists) is directed in counterclockwise order.

Our first result is topological and asserts that the traversal of every contour cycle of a connected component E of $B \cap R$, B a "blue" and R a "red" polygon, "agrees" with the teraversal of the contour cycles of B and R. By this we mean that the common points of a contour cycle of E and one of B (or R) are traversed in the same order independent of whether we follow the contour cycle of E or that of B (or R).

Lemma 8. *Let E, B, and R by polygons such that E is a connected component of $B \cap R$, and let a, b, c be three points on $\gamma \cap \xi$, where γ is a contour cycle of E and ξ is a contour cycle of B. The order of points a, b, c along ξ is the same as along γ.*

Proof. Note first that the directions of γ and ξ along common boundary pieces agree since E and B lie on the same side of these pieces. Take ξ, the contour cycle of B that contains points a, b, c, and let ab_ξ, bc_ξ and ca_ξ be the pieces (Jordan arcs) of ξ from a to b, from b to c, and from c to a. By assumption, points a, b, c belong also to γ. If ab_ξ is contained in γ, then the assertion is trivially true since the traversal from a to b on γ only passes points of ab_ξ, and c does not belong to ab_ξ. Otherwise, ab_γ, the portion of γ leading from a to b, contains pieces that do not belong to ξ—these pieces are necessarily contained in the union of ∂R and $\partial B - \xi$ (see Fig. 5.1). Let δ be such a piece, that is, δ is a maximal connected component of $\gamma - \xi$, whose starting point, z, lies on ab_ξ. We prove below that the endpoint, w, of δ also lies on

[6] A (*simple*) *Jordan curve* has the property that every sufficiently small disk whose center lies on the curve is cut into two connected components if we remove from it all points of the curve. This implies that a Jordan curve is either unbounded at both ends or it is bounded in which case it is said to be *closed*. A connected piece of a Jardan curve is called a *Jordan arc*; it satisfies the same condition as the Jordan curve expect at its two endpoints at which removing the points of the arc leaves every small enough disk connected.

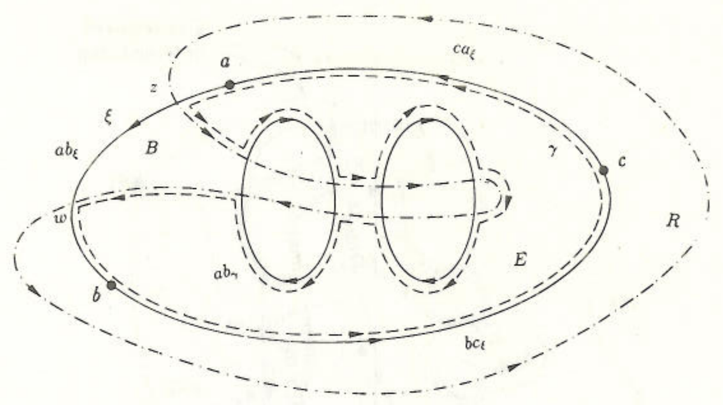

Fig. 5.1. Traversing the boundary of the intersection.

ab_ξ which implies the lemma since we cannot reach any point of $\xi - ab_\xi$ before passing through b.

Assume that w does not lie on ab_ξ. Let δ' be the portion of ξ leading from z to w, including the endpoints (see Fig. 5.2; note that δ' must contain b). By construction, δ and δ' are disjoint and their union is a closed Jordan curve σ. Let a' and b' be two interior points of E that lie sufficiently close to a and b. It is thus possible to connect a' and b' by a Jordan arc, α, that lies sufficiently close to $ab\xi$ such that $\alpha \cap \delta' = \emptyset$ and α and δ cross in a single point. Thus, a' and b' lie in different components of the complement of σ which is impossible since σ is disjoint from E and E is connected (see Fig. 5.2). □

Remark. Lemma 8 expresses a consistency property of intersections of polygons. Among other things it implies that if an edge e of B (or R, for that matter) contains several edges of a contour cycle of E, then these edges appear in the same order along this contour cycle as along e.

Consider next a blue polygon B with ℓ contour cycles $\xi_1, \xi_2, \ldots, \xi_\ell$ and let $p_1, p_2, \ldots, p_m$ be the points designating m desired regions contained in B. We let R_i be the red polygon that contains p_i, for $1 \leq i \leq m$, and we write E_i for the connected

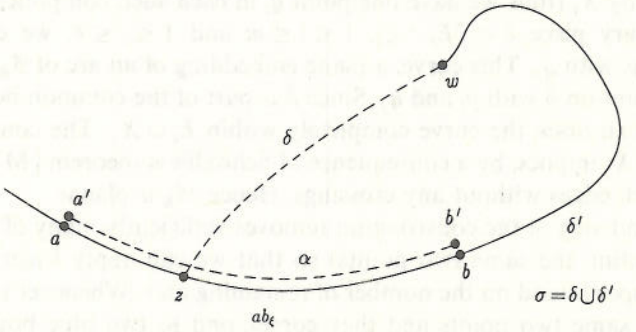

Fig. 5.2. An impossible configuration.

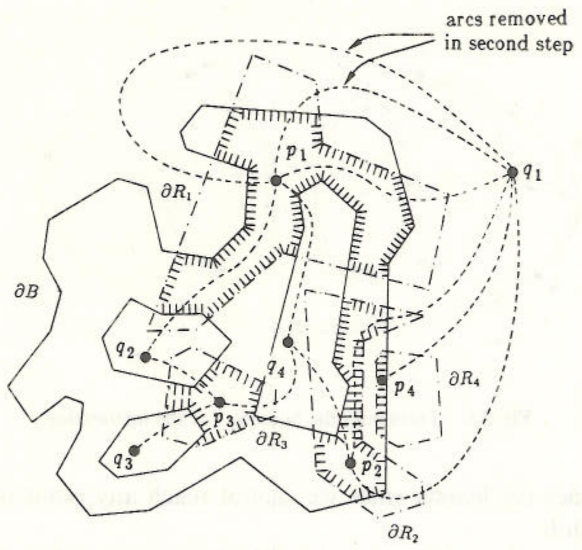

Fig. 5.3. The graph $\mathscr{G}_B$.

component of the polygon $B \cap R_i$ that contains p_i (see Fig. 5.3). To be consistent with the assumptions for Lemma 5 we allow $R_i = R_j$ but we assume that $E_i \neq E_j$ if $i \neq j$. We analyze the blue boundary pieces of the E_i by constructing a graph, $\mathscr{G}_B$, and proving certain properties about $\mathscr{G}_B$. This graph is instrumental in proving an upper bound on the total number of blue edges bounding the E_i.

We need a few definitions. A *blue boundary piece* of E_i is a connected component of $\partial E_i \cap \partial B$. Every blue boundary piece, δ, belongs to a contour cycle γ of some E_i and to a blue contour cycle ξ_j. Since contour cycles are directed, we can define a *predecessor* and a *successor* of δ in both cycles, which are the blue boundary pieces immediately before and after δ in γ and in ξ_j. Note that it is possible that the predecessor (successor) of δ in γ is the same as in ξ_j, but this is not necessarily the case. Graph $\mathscr{G}_B$ is a graph whose nodes are the points $p_1, p_2, \ldots, p_m$ and ℓ additional points $q_1, q_2, \ldots, q_\ell$ so that q_i lies in the interior of the connected component of the complement $\mathbb{R}^2 - B$ of B bounded by ξ_i; we denote this component by X_i (thus we have one point q_i in each such component). For every blue boundary piece $\delta \subset \partial E_i \cap \xi_j$, $1 \leq i \leq m$ and $1 \leq j \leq \ell$, we draw a curve connecting p_i with q_j. This curve, a plane embedding of an arc of $\mathscr{G}_B$, connects an arbitrary point on δ with p_i and q_j. Since δ is part of the common boundary of E_i and X_j we can draw the curve completely within $E_i \cup X_j$. The connectedness of each E_i and X_j implies, by a consequence of Schönfliess theorem [M], that we can draw all such edges without any crossings. Hence, $\mathscr{G}_B$ is planar.

The second step of the construction removes sufficiently many of the duplicate arcs (connecting the same two points) so that we can apply Euler's relation to derive an upper bound on the number of remaining arcs. Whenever two arcs of $\mathscr{G}_B$ connect the same two points and they correspond to two blue boundary pieces such that one is the successor of the other in both contour cycles, then we delete the successor arc. (In the case that both blue boundary pieces are successors of each other, we make an arbitrary decision to break the tie.) As a result of this deletion operation, every blue boundary piece $\delta \subset E_i \cap \xi_j$ intersects an arc of $\mathscr{G}_B$, unless there is another such piece $\delta' \subset E_i \cap \xi_j$ that precedes δ in both contour cycles. Note that the removal of arcs as described does not eliminate all multiarcs but it guarantees that, barring one extreme situation mentioned below, every region of the embedding of $\mathscr{G}_B$ is bounded by at least four arcs, counting an arc twice if it lies in the closure of the region. The one case where this does not hold is when $\mathscr{G}_B$ contains only two vertices and one connecting (doubly counted) arc. This arises when B contains just one point p_i, and the corresponding E_i uses only one contour cycle of B. In this case we can delete this arc too, because it will not be used in the argument to follow. The factor "four" rather than "three" which is typical for planar graph arguments can be used because $\mathscr{G}_B$ is bipartite by definition and has therefore no odd cycles. Using Euler's relation we derive that the number of remaining arcs is at most $2(m + \ell) - 4$ (also in the special case mentioned above).

We are now ready to prove a strong variant of Lemma 5 stating that the k connected components of the $B_i \cap R_j$, $1 \leq i \leq s$ and $1 \leq j \leq t$, containing the k points $p_1, p_2, \ldots, p_k$ are bounded by a total of at most $\beta + \rho + r + 12k + 6\ell - 24$ edges, where

β is the total number of (blue) edges of the B_i,
ρ is the total number of (red) edges of the R_j,
r is the total number of reflex vertices of the B_i and R_j, and
ℓ is the total number of contour cycles of the B_i and R_j.

Lemma 5 is implied because $\ell \leq r + 2k$ (each B_i or R_j has at most one exterior contour cycle, there are at most $2k$ blue and red polygons, and each interior contour cycle must contain a reflex vertex), and thus $\beta + \rho + r + 12k + 6\ell - 24 = \beta + \rho + O(r) + O(k)$. In the argument to come we traverse all blue (and symmetrically all red) contour cycles and count the blue (red) edges of the E_i, $1 \leq i \leq k$, as we encounter them, by charging them to various "acounts." Note that a blue (red) edge can contain several such edges. We define $\mathscr{G}$ as the union of all graphs $\mathscr{G}_{B_i}$ and $\mathscr{H}$ as the union of all graphs $\mathscr{G}_{R_j}$ defined symmetrically for all red polygons.

The easy case is if a blue (red) edge, e, contains at most one edge of all the E_i—the appearance of this edge is accounted for by the term β (ρ) in the upper bound. Otherwise, let e_1 and e_2 be two components of $e \cap (\bigcup_{1 \leq i \leq k} \partial E_i)$, consecutive along e, and assume that e_1 is already accounted for. (We always charge the first such component to e itself, so these charges are absorbed in the term β (ρ).) We assume e is blue.

(i) If e_1 and e_2 do not lie on a common contour cycle of an E_i (and thus belong to the boundaries of two different polygons E), then we charge e_2 to the arc of $\mathscr{G}$ that is induced by the blue boundary piece that contains e_2 (note that this arc cannot have been deleted from $\mathscr{G}$; see Fig. 5.4 (a)).

On the other hand, if e_1 and e_2 belong to a common contour cycle γ of some E_i, then we distinguish two cases. Let γ_0 be the piece of γ connecting the last point of e_1

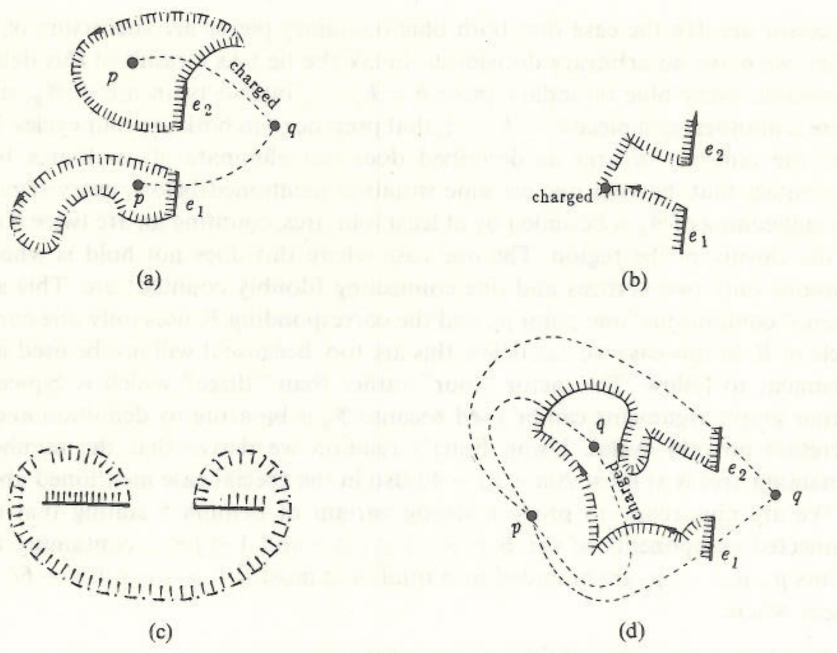

Fig. 5.4. (a) Charging rule (i). (b) Charging rule (ii)(a). (c) Charging rule (ii)(b). (d) Charging rule (iii).

with the first point of e_2. Note that, by Lemma 8, the relative interior of γ is disjoint from e. Moreover, it is disjoint from the blue contour cycle containing e.

(ii) If γ_0 contains no blue boundary piece, then there are two subcases:
   (a) γ_0 contains a red reflex vertex (see Fig. 5.4 (b)). Then we charge e_2 to the first such reflex vertex.
   (b) γ_0 contains no red reflex vertices. In this case γ_0 must "go around" e, as shown in Fig. 5.4 (c); that is, γ_0 keeps turning to the left after leaving e_1, crosses the line containing e before e, crosses it again after e, and eventually reaching e again at e_2 (making only left turns all along). In this case we charge e_2 to the arc of $\mathscr{H}$ induced by γ_0, or, if this arc has been removed from $\mathscr{H}$, to the preceding arc of $\mathscr{H}$ that connects the same two nodes of that graph.

(iii) Otherwise, we charge e_2 to the arc of $\mathscr{G}$ that is induced by the first blue boundary piece on γ_0 (again, by Lemma 8, this arc cannot have been deleted from $\mathscr{G}$; see Fig. 5.4 (d)).

It is easy to see that each reflex vertex is charged in (ii)(a) at most once. The definition of $\mathscr{G}$ guarantees that in each case where we charge $\mathscr{G}$ there is in fact an arc that takes the charge. We now argue that the mechanism we use can charge an arc of $\mathscr{G}$ at most twice. In case (i) the arc takes the charge for an edge, e_2, that is the first edge of the corresponding blue boundary piece. Since every blue boundary piece has only one first edge, this case can occur only once. In case (iii), the arc that takes the charge for a pair of edges, e_1 and e_2, corresponds to a different blue contour cycle. However, e_1, the blue boundary piece δ that corresponds to the charged arc, and e_2 all belong to the same contour cycle of some E_i and δ is the first blue boundary piece that occurs between e_1 and e_2 in this cycle. Clearly, it cannot be the first blue boundary piece after a blue boundary piece other than that containing e_1, which implies that this case also puts at most one charge onto any one arc.

Finally we claim that in case (ii)(b) no arc of $\mathscr{H}$ is charged more than once. Assume to the contrary that such an arc is charged twice, because it is induced by two red boundary pieces, γ_0, γ'_0, that bound the same region E_i, and each of them is being charged by rule (ii)(b). It is easily checked that these charges must be induced by two *distinct* blue edges e, e' (recall that by our convention e represents just one side of a blue segment, and similarly for e'). Let e_1, e_2 be the two consecutive portions of e, separated along ∂E_i by γ_0, and similarly let e_1', e_2' be the two consecutive portions of e', separated along ∂E_i by γ'_0.

Take a point $p \in e$ lying immediately after e_1 and a point $p' \in e'$ lying immediately after e'_1. If we move from p slightly away from e to its left, and similarly for p', we will be in the same connected component of the complement of the red polygon containing E_i—this is the component bounded by the red contour cycle containing both γ_0 and γ'_0. We can therefore connect p to p' by a path π that is disjoint from E_i. Let us form a "dummy" blue contour cycle β composed of e, e', and π and of the other sides of e, e', and π (this argument shows incidentally that e' cannot be the other side of e).

Our assumptions imply that the four subsegments e_1, e_2, e'_1, and e'_2 appear in this (circular) order along ∂E_i. By Lemma 8 this must also be the order in which they appear along β. However, it is easily verified that this is impossible. This contradiction shows that no arc of $\mathscr{H}$ is charged more than once.

Thus, summing over all blue and red edge duplications, we see that the total number of edges bounding the E_i, $1 \le i \le k$, is at most

$$\beta + \rho + r \qquad \text{plus three times the total number of arcs of } \mathscr{G} \text{ and } \mathscr{H}.$$

The number of arcs of $\mathscr{G}$ and $\mathscr{H}$ is at most $4k + 2\ell - 4s - 4t$ since every point p_i is counted twice (once for the blue and once for the red polygon that contains it). This implies the claim and completes the proof of Lemma 5. □

6. Calculating Many Faces in an Arrangement of Line Segments

We next turn to the task of calculating the faces in an arrangement of n line segments, $s_1, s_2, \ldots, s_n$, that contain m given points, $p_1, p_2, \ldots, p_m$. Generalizing the ray shooting method used in Section 3 for line arrangements is problematic because it would call for performing such ray shooting queries inside polygonal regions that are not simply connected. No efficient technique for doing so is currently known. This section presents an alternative approach based on the line-sweeping technique (see, e.g., [PS]).

Consider the algorithmic issues that arise in efficiently implementing the algorithm implicitly described in Sections 4 and 5. At the bottom of the recursion, when $m_v \geq \kappa(n_v)$, we need to calculate the entire arrangement of the n_v line segments in S_v, and extract from it the faces containing the m_v points reaching v. Using the sweep algorithm in [BO] this arrangement can be constructed in $O(n_n^2 \log n_v)$ time, and the faces that contain the m_v points can be identified in time $O(m_v \log n_v)$ using any of a number of efficient point-location methods. Hence, such a node v can be processed in time $O(m_v \log n_v)$.

In the general merging step, we first need to calculate, for each subregion v, the exterior face of $A(S_v)$, which contains the points in Q_v by definition. This step of the computation is described at the end of Section 7. It falls out as a special case of the general merge procedure described there. The resulting algorithm, stated in Theorem 11 of Section 7, has time complexity $O(n_v\alpha(n_v) \log^2 n_v)$, if the unbounded face has to be constructed from scratch (which is the case if $m_v = 0$), but this can be improved to $O(n_v\alpha(n_v) \log n_v)$, if we already have available the unbounded faces in the subarrangements whose overlay gives $A(S_v)$ (which is the case if $0 < m_v < \kappa$ (n_v); if $m_v > \kappa(n_v)$ we compute the entire arrangement $A(S_v)$, so no special attention is required for the unbounded face).

Next, in the merging itself, for each $p_i \in P$ we need to calculate the face $F(p_i)$ of the arrangement $A(S)$ that contains p_i. This face is the connected component containing p_i of the intersection of the M faces $F_{v_j}(p_i)$, $j = 1, 2, \ldots, M$, where $v_1, \ldots, v_M$ are the triangles formed in the partitioning (note that each of these faces is either computed recursively or is the corresponding unbounded face F_v^∞ whose computation has just been discussed). As in Section 3, our goal is to construct these faces in time that only depends on their total combinatorial complexity. Section 7 presents a method, called the *blue-red merge*, that can be used to construct the faces $F(p_i)$ in such an efficient manner.

Let us be more specific about the blue-red merge. The input to this procedure is a set of pairwise disjoint blue (and red) polygons bounded by a total number of β (ρ) edges, and a set of k points each one inside some blue and some red polygon. The output is the set of k polygons that are the connected components of the blue-red intersections that contain the k points. The blue-red merge constructs the output polygons in time $O((\beta + \rho + k) \log (\beta + \rho + k))$ (see Theorem 10). To construct the faces $F(p_i)$ we apply the blue-red merge $M - 1$ times to the faces $F_{v_j}(p_i)$, $j = 1, 2, \ldots, M$. The amount of time required for the merging is thus

$$\sum_{j=1}^{M} O((\bar{R}(m_{v_j}, n_{v_j}) + m_{v_j}) \log n_{v_j} + n_{v_j}\alpha(n_{v_j}) \log n_{v_j})$$
$$= O((\bar{R}(m, n) + m + n\alpha(n)) \log n).$$

Here we use the fact that the number of reflex vertices in the relevant faces is $O(n)$—they must be endpoints of line segments in S_v. In addition, we use Lemma 5 for each of the $M - 1$ blue-red merging steps to deduce that the output size of each merge is linear in its input size. From this the above bound follows readily. We now put everything together, also taking into account the cost of the recursive partitioning of the dual plane (as in Section 3).

Let $T(m, n)$ denote the time needed by our algorithm to calculate the faces in an arrangement of n line segments that contain m given points, under the assumption that the random samples that we draw at each recursive step are indeed ε-nets. (As in Section 3, we verify this property after each sampling, discard the sampling if it is not an ε-net, and try another one. This makes the expected running time of the algorithm within a constant factor of $T(m, n)$.) We get the following recurrence relation for $T(m, n)$:

$$T(m, n) = O(n\alpha(n)\log^2 n) \qquad \text{if} \quad m = 0,$$

$$T(m, n) = O(m \log n) \qquad \text{if} \quad m \geq \kappa(n),$$

and

$$T(m, n) \leq \sum_{i=1}^{M} T(m_i, n_i) + O((R(m, n) + m + n\alpha(n)) \log n) \qquad \text{if} \quad m < \kappa(n),$$

where m_i, n_i, and M satisfy conditions (I)–(III) stated in the analysis given in Section 2. Using the bounds on $\bar{R}(m, n)$ and $R(m, n)$ obtained above, we can easily obtain the following bounds, in much the same way as in the proofs of Lemma 2 and Theorem 3.

Theorem 9. *The m faces designated by m points in an arrangement of n line segments in the plane can be constructed in randomized expected time*

$$T(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} \log n + n\alpha(n) \log^2 n \log m),$$

where δ is any positive real number. The space required by the algorithm is

$$S(m, n) = O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n) \log m)$$

for any $\delta > 0$.

7. The Blue–Red Merge

This section presents the details of the blue–red merge which computes the relevant faces in an arrangement $A(S_v)$ assuming that the faces at the M subarrangements $A(S_{w_j})$ have already been constructed recursively. As in Section 3 we assume $M = 2$, so that we need to intersect only two faces around each point p_i. We are thus given two collections of (not necessarily simply connected) open polygons in the plane, $\{B_1, B_2, \ldots, B_s\}$ and $\{R_1, R_2, \ldots, R_t\}$; the B_i are called the *blue* polygons and the R_j are the *red* polygons. We can assume that any two blue (red) polygons are disjoint. In addition we are given a set P of k points, $p_1, \ldots, p_k$, where each p_i is contained in some blue polygon and in some red polygon. See Fig. 7.1 (a) and (b) for an illustration of this set-up. Let β and ρ denote the total number of edges of the blue and red polygons, respectively.

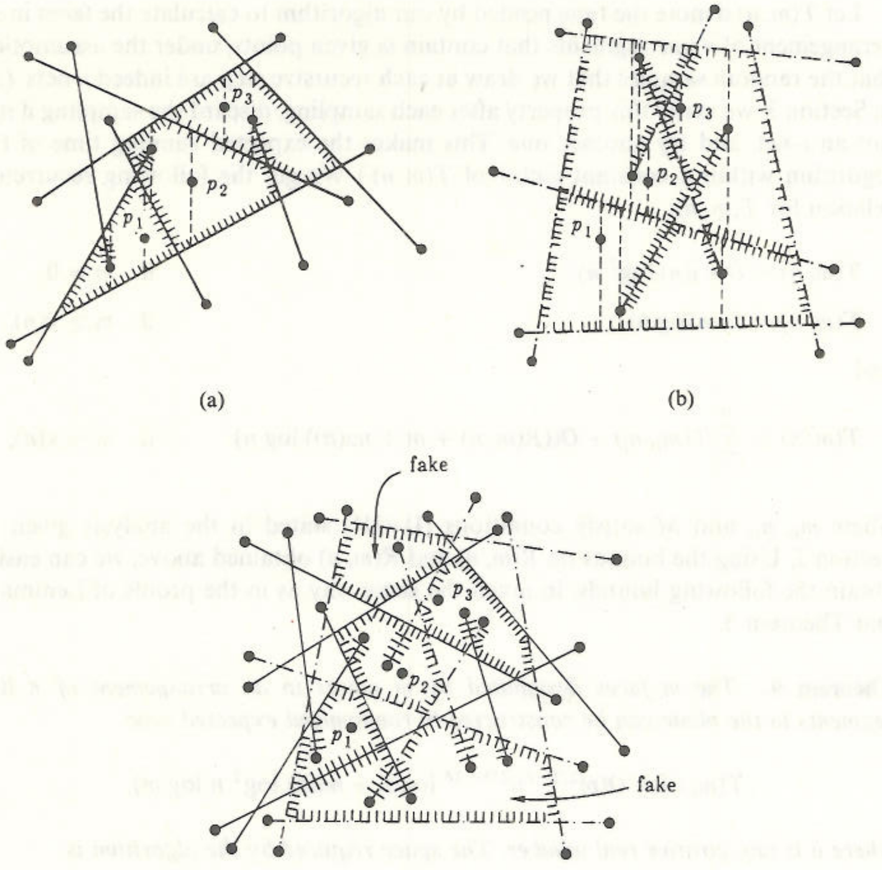

Fig. 7.1. Intersecting blue and red polygons. (a) The blue polygons. (b) The red polygons. (c) The purple polygons.

Our goal is to calculate all polygons E_i, where, for each $p_i \in P$, E_i is the connected component containing p_i of the intersection of the red polygon and the blue polygon that contain p_i. The resulting E_i are called the *purple* polygons, as each is covered by a red and a blue polygon. Fig. 7.1 (c) shows the purple polygons that arise from the blue and red polygons and surround the points shown in Fig. 7.1 (a) and (b). As in Section 3, we do not exclude the possibility that $E_i = E_j$ for $i \neq j$. If π is the total number of purple edges, then $\pi = \beta + \rho + O(k) + O(r)$, by Lemma 5, where r is the total number of reflex blue and red vertices. Since $r \leq \beta + \rho$ we have $\pi = O(\beta + \rho + k)$.

To facilitate the merge, we require certain information to be precomputed and available for each collection of polygons. Specifically, let P_B and P_R be the set of reflex vertices of the blue and red polygons, respectively. We require that each blue polygon B_i be subdivided into convex regions by drawing vertical rays from each point $p \in (P \cup P_B) \cap B_i$ and stopping them as they encounter an edge of B_i; we call the resulting vertical line segment through p the *blue vertical divider* of p, and denote it by *blue*(p) (see Fig. 7.1 (a)). Symmetrically, we require that each red polygon R_j is decomposed into convex regions by *red vertical dividers red*(p), $p \in (P \cup P_R) \cap R_j$ (see Fig. 7.1 (b)). These convex decompositions of the blue and red polygons will be available recursively. These decompositions do not increase the complexity of the procedure by more than a constant factor.

A particular convex blue (or red) region terminates on the left or the right either

(i) because of a point of P,
(ii) because of a blue (or red) reflex vertex, or
(iii) because of a locally x-extremal (nonreflex) vertex of the corresponding polygon.

The blue–red merge produces a similar decomposition of the purple polygons into convex regions, which we call the *purple regions*.

We construct the purple regions by sweeping a vertical line across the plane. A purple region starts (and ends) at

(i) a point of P,
(ii) a blue or red reflex vertex,
(iii) an x-extremal vertex of a blue or red polygon, or
(iv) the intersection of a blue and red edge.

In a left-to-right sweep we discover the portion of each purple polygon that is to the right of the leftmost point in $P \cup P_B \cup P_R$ that lies in it. Afterward, in a right-to-left sweep, we get the portion of each purple polygon to the left of the rightmost point in $P \cup P_B \cup P_R$ that lies in the polygon. Together, the two sweeps discover all edges of the purple polygons. In fact, the sweeps construct slightly more than the purple regions (the convex decomposition of the purple polygons) and we have to remove superfluous regions after the two sweeps (see Fig. 7.1 (c)). More about this later.

We are now ready to describe the left-to-right sweep—the right-to-left sweep is symmetric. We start by constructing a priority queue that stores $P \cup P_B \cup P_R$ ordered by x-coordinates. This priority queue is referred to as the *event schedule*. During the sweep we maintain separate data structures for the blue, red and purple regions. Thus, it is convenient to think of blue, red, and purple planes swept simultaneously and independently. The main purpose of keeping the blue and red regions separate is to avoid processing "uninteresting" blue–red intersections that do not contribute to the boundary of a purple region. The only data structure that represents blue, red, and purple data mixed together is the event schedule. The main part of the blue–red merge is to detect intersections of blue and red edges that occur on the boundary of a purple region. When such an intersection is detected it is added to the event schedule and appropriate actions are taken to adjust the data structures supporting the sweep.

Introduction of Scouts. Each time a point $p \in P \cup P_B \cup P_R$ is encountered, we possibly start one or two new purple regions in the purple plane. If $p \in P$, then it belongs to a blue and a red region and we start one new purple region. If p is a reflex vertex of a blue (red) polygon we check whether it lies inside a red (blue) polygon. If it does not we simply discard it, and if it does we start two or one purple region(s) depending on whether or not both incident edges of p lie to the right of the vertical line through p.[7] Whenever a new purple region is started we create two *scouts*, an *upper* and a *lower* scout for the region. The job of the two scouts is to walk along the upper and lower boundary of the new purple region and to look out for events to come which might influence the shape of their region. The scouts always stay with the sweep line and never stroll ahead or stay behind. We describe the way the upper scout of the purple region does its job—the lower scout behaves symmetrically.

The upper scout, u, starts on the lowest blue or red edge above p, the point that gives rise to the purple region at hand. If $p \in P$, then the edge is either the blue edge that contains the upper endpoint of $blue(p)$ or the red edge that contains the upper endpoint of $red(p)$. If p is a blue (red) reflex vertex, then we consult the data structure that represents the red (blue) sweep to find the lowest red edge vertically above p[8] and we compare this edge with the blue (red) edge that contains the upper endpoint of $blue(p)$ ($red(p)$). The scout moves right along that edge following the sweep line and it watches out for certain events that might influence the construction of the purple boundary.

Without loss of generality, assume that u currently lies on a blue edge. Most importantly, u looks up to the lowest red edge above—call it e_r. Since u is a point of a purple region, all points between u and e_r including u lie in the red polygon bounded by e_r. The reason for u's concern is that, at some future point in time, the red boundary above u might drop below the blue boundary u is currently following. If this indeed happens u will switch to the red boundary which will then delimit the purple region. However, there might be another scout, v, already watching the same red edge from below. In that case only the higher of u and v needs to watch e_r—the other scout can rest, since it is protected by the other scout, who will have to warn it in case the red boundary drops unexpectedly.

In more technical detail, "watching e_r" means that u determines whether e_r and e_b, the edge it walks on, intersect to the right of the sweep line. If so it adds the intersection point to the event schedule of the sweep. If such an intersection does not exist and u reaches the right endpoint of e_b (or e_r) it continues walking on (watching) the next blue (red) edge, if it exists. On the other hand, if e_b and e_r intersect, then u switches over to e_r when the sweep line passes the intersection. In this case, u starts watching e_b which is now the lowest blue edge above u. Of course, there is also the case that the red boundary watched by u discontinuously changes

[7] Assuming that we deal with polygons bounded by simple Jordan curves there are exactly two such edges. In our application, however, each reflex vertex is an endpoint of a line segment and thus, assuming general position of the segments, is incident to only one edge. We treat the two sides of this edge as two edges. Thus, either both incident edges are to the left or both are to the right of the vertical line through the reflex vertex. However, it is also easy to handle the case of endpoints being shared by more than one segment.

[8] This data structure is a balanced tree that stores the red (blue) edges that currently intersect the sweep line. Thus, logarithmic time (in the number of red (blue) edges) is sufficient to find the lowest red (blue) edge above a given point. New edges can be added and old edges can be removed in logarithmic time.

at some point—in this case u must look for a new assignment. Below, this case is treated in detail.

Another job of u is to watch its partner, the lower scout of the sample purple region. This is because when the two scouts meet then the purple region ends. Of course, the region ends earlier if another point of P or a reflex vertex is encountered between the two scouts.

Scout Invariants. There are two key properties that are satisfied at any given time. The first is that at any point in time each blue and each red edge is watched by at most one upper or lower scout. Of course, the assignment of the scouts may change and, over time, a single edge can be watched by many scouts. The second property is that a blue edge is watched only by scouts that walk along red edges, and a red edge is watched only by scouts walking along blue edges.

Changing Assignments. Reassignments for scouts are necessary when new purple regions start and old ones end. A purple region might end, for example, when its two scouts meet. This occurs, for instance, when the rightmost vertex of a blue region lies inside a red region. In this event, the two purple scouts are dismissed. However, some transfer of watching responsibility is indicated before the two scouts leave the scene. If the dismissed upper scout, u, was watching a red edge, e_r, then we must check v, the next upper scout below u. If v is idle (this can only be because u protected v from e_r), then v takes over the responsibility to watch e_r. If v is already watching another red edge, then we leave it undisturbed as its red edge must lie below e_r.

The two scouts of a purple region also come together when a purple region ends at the intersection of a red and a blue edge. Any reassignment of watching responsibilities are handled as before.

If a purple region ends because of a point $p \in P \cup P_B \cup P_R$ that appears between the scouts, then again the two scouts are dimissed. In this case, however, they will generally be replaced by new scouts employed to guard the purple region(s) started at the vertical divider of p.

The reassignment of watching responsibility necessary when we sweep through such a point p is done as follows. At the time p is encountered, one or two new purple regions are created. Assume for simplicity that there is only one new region, let u be its upper scout and assume that u starts out on a blue edge. First, u finds the lowest red edge above it using the data structure that represents the sweep in the red plane. Second, u consults the upper scout, w, immediately above and the upper scout, v, immediately below. If w is watching the same red edge as u, then u forgets about its assignment and becomes idle. Otherwise, u takes up its assignment and checks whether v watches the same red edge. If yes, then v becomes idle. Similar actions are taken when two new purple regions are spawned at p.

Finally, some transfer of watching responsibility may be required at a blue-red crossing lying, say, on the upper boundary of some purple region. Suppose the corresponding upper scout, u, walks along a blue edge, e_b, just before the intersection occurs. To the right of the crossing u follows a red edge, e_r. As noted above, u now starts watching e_b, but we also need to check the upper scout v

immediately below u, who might want to start watching e_r. Details are similar as in the cases considered above.

Analyzing the Blue-Red Sweep. The scouts simply trace the boundaries of the purple regions. Each scout adds the intersections between blue and red edges that it predicts to the event schedule. New events are added only when we sweep through a point in P, through a blue vertex, through a red vertex, or through the intersection between a blue and a red edge that is also a vertex of a purple region. Moreover, at each such point only a constant number of additions of new events are made. Thus the total number of events ever scheduled is proportional to the total input and output size, which, by Lemma 5, is $O(\beta + \rho + k)$. The time to add or remove an entry to or from the event schedule is logarithmic in its size which is thus $O(\log(\beta + \rho + k))$. The additional operations involve updating the balanced trees that represent the blue, red, and purple cross-sections along the sweep line, creating and dismissing scouts, and reassigning watching responsibilities. It is plain that we need to perform only $O(\beta + \rho + k)$ such operations, and that each one can be carried out in time $O(\log(\beta + \rho + k))$.

There is a minor point to be clarified here: the algorithm constructs more than only the "true" purple regions (those that are connected by sequences of adjacencies across vertical dividers to regions that contain points of P). Indeed, it constructs all connected components of the blue-red intersections that contain a point of P or a reflex blue or red vertex (see, for example, Fig. 7.1 (c) which shows two purple faces that contain no point of P). The construction of these additional faces seems necessary since a genuine purple face can go back and forth in a serpentine-like fashion, which makes it difficult for a sweep to capture its entire boundary unless it collects various portions of the face in advance. Since the number of reflex vertices is at most $\beta + \rho$, this does not affect the asymptotic time-complexity of the above algorithm. The final step now removes fake purple regions. This can be done by a simple graph search in time proportional to the number of purple regions produced by the algorithm, hence in time $O(\beta + \rho + k)$. Thus the blue-red merge takes time $O((\beta + \rho + k) \log(\beta + \rho + k))$.

Theorem 10. *Let the B_i (R_j) form a collection of pairwise disjoint blue (red) polygons in the plane bounded by a total number of β (ρ) edges, and let P be a set of k points each contained in a blue and a red polygon. The connected components of the intersections between the blue and the red polygons that contain the given points can be constructed in time $O((\beta + \rho + k) \log(\beta + \rho + k))$.*

We next show how to apply the blue-red merge to the calculation of a single face, F, in an arrangement of a collection of n line segments, $S = \{s_1, s_2, \ldots, s_n\}$. As usual, F is assumed to be represented by a single point p contained in F. We now employ a straightforward divide-and-conquer procedure (nothing like our present intricate partition tree scheme):

Step 1. Partition S into subsets S_1 and S_2 of about half the size of S each.
Step 2. Calculate the faces F_1 and F_2 in the arrangements $A(S_1)$ and $A(S_2)$ that contain p.
Step 3. Apply the blue-red merge described above to F_1, F_2, and $\{p\}$.

By the results of [PSS], we know that F_1 and F_2 together have $O(n\alpha(n))$ edges. Thus, by Theorem 10, Step 3 constructs F from F_1 and F_2 in time $O(n\alpha(n) \log n)$. We therefore get

$$T(n) = 2T(n/2) + O(n\alpha(n) \log n) = O(n\alpha(n) \log^2 n)$$

for the total amount of time required to construct F. We state this result as a theorem.

Theorem 11. *A single face in an arrangement of n line segments in the plane can be calculated in time $O(n\alpha(n) \log^2 n)$.*

Remarks. (1) Theorem 11 extends and simplifies previous results on constructing a single face in a line-segment arrangement obtained in [PSS].

(2) In the construction of m faces in a line-segment arrangement, we need Theorem 11 for calculating the unbounded face, F_v^∞, in $A(S_v)$, at each subregion v. If v represents an inner recursive step, then we already have a constant number of unbounded faces available such that F_v^∞ is the unbounded connected component of their intersection. Hence, we do not have to pay for the recursive overhead (as in Theorem 11), which gives us an $O(n_v\alpha(n_v) \log n_v)$-time algorithm for constructing F_v^∞. No $\log n_v$ factor can be saved, however, if v is at the bottom of recursion.

(3) Note that the algorithm given in Theorem 11 is deterministic.

8. Discussion and Open Problems

In this paper we have obtained almost tight upper bounds for the maximum number of edges bounding m faces in an arrangement of n lines or line segments. We also presented efficient randomized algorithms for the calculation of these faces. The expected randomized time-complexity of these algorithms is only slightly higher than the upper bounds on the nuber of edges that are to be reported. The main technical tools that we have introduced and used in our analysis are

(i) efficient planar partitioning schemes for a set of n points and m lines, using a random sampling technique similar to those of [HW] and [C1], and
(ii) the combination lemmas for faces in arrangements of lines and of line segments.

This final section concludes with some comments on our techniques, reports on further progress, and states related open problems.

The problem of calculating m faces in an arrangement of n lines or line segments generalizes two simpler variants. One of them, originally posed by Hopcroft, is the following:

Given m points and n lines in the plane, determine whether any of the points lies on any of the lines.

Another variant is

Given m points and n lines in the plane, find for each point the nearest line that lies vertically below it.

Since both problems are restricted cases of the problem studied in this paper, our algorithm yields efficient solutions to these simpler problems as well, improving earlier solutions in [CSY]. Note, however, that in these problems the output size is not a significant issue—the first problem is just a decision problem and the output in the second problem has only linear size. In constrast, for the problem studied in this paper the output size, and thus the space- and time-complexity of the algorithm, can be forced to be superlinear, that is, $\Omega(m^{2/3}n^{2/3})$. An obvious open problem that arises is whether the two simpler problems can be solved in $o(m^{2/3}n^{2/3})$ time. As is shown elsewhere [EGSh], the straightforward generalizations of the two problems to three dimensions can indeed be solved faster than the best-known solution for the problem of calculating the entire cells of the arrangement containing the given points.

Our approach to calculating faces in arrangements is based on a dualization of the problem which puts limits on its generality. Nevertheless, we could give an equivalent description of our technique using only the primal plane. We thus draw a random sample of r of the given points and then partition the lines into $O(r^2)$ so-called 3-*corridors* each being the primal equivalent of a triangle in the dual plane (see [HW]). Each point is passed to all 3-corridors that contain it. It is an interesting open problem whether or not this primal view of our technique can be generalized to apply to arrangements of other curves such as circles and alike.

Recently, after the original submission of this paper, considerable progress was made on the problems studied here and on many related problems. The new results improve, extend, or apply the results and techniques developed here. Many of these latter works, listed below, are based on the tools and methods of this paper:

(1) We have already mentioned the companion paper [CEG*], in which an improved and tight combinatorial bound has been obtained for the case of lines.

(2) Later, Edelsbrunner *et al.* [EGH*] have obtained efficient algorithms to preprocess an arrangement of n lines so that, given a query point, the face of the arrangement containing the point can be computed efficiently.

(3) Our complexity bound for the case of segments has been recently improved by Aronov *et al.* [AEGS] to $O(m^{2/3}n^{2/3} \log^{1/3}(n^2/m) + n\alpha(n) + n \log m)$. This is still not known to be tight, but the gap between the upper and lower bounds has been reduced considerably.

(4) The blue-red merge, and a weaker version of the combination lemma, have been extended in Guibas *et al.* [GSS] to the case of arrangements of curved arcs, provided no pair of the arcs intersect in more than a fixed constant number s of points. They have applied these tools to obtain an $O(\lambda_{s+2}(n) \log^2 n)$ algorithm for the calculation of a single face in such an arrangement, where $\lambda_{s+2}(n)$ is the maximum length of $(n,s+2)$-Davenport-Schinzel sequences.

(5) The results obtained in this paper have been applied in [AS] to bound the complexity of all nonconvex cells in an arrangement of n triangles in 3-space. A by-product of the research in [AS] is a simpler proof of the combination lemmas, Lemmas 1 and 5.

(6) Finally, there has been recently considerable progress in obtaining *deterministic* algorithms for partitioning arrangements of lines in the plane. The best result in this direction is due to Agarwal [A], and it leads, among other applications, to a deterministic algorithm for computing m faces in an arrangement of n line segments in time $O(m^{2/3}n^{2/3}\log^{\sigma} n + (m+n)\log n)$ for some constant $\sigma < 3$.

Acknowledgments

We wish to thank Ken Clarkson for providing valuable insights into the random sampling technique which have helped us in substantial improvements of our technique. Thanks are also extended to John Hershberger, Janos Pach, Raimund Seidel, Jack Snoeyink, Jorge Stolfi, and Emo Welzl for useful discussions on the problems studied in this paper. Part of the work on this paper has been carried out at DEC Systems Research Center, and the authors would like to express their gratitude for its generous hospitality.

References

[A] Agarwal, P. K. An efficient algorithm for partitioning arrangements of lines and its applications. In *Proc. 5th ACM Symp. Comput. Geom.*, 1989, pp. 11–22.

[AEGS] Aronov, B., Edelsbrunner, H., Guibas, L., and Sharir, M. Improved bounds on the number of edges of many faces in arrangements of line segments. Report UIUCDCS-R-89-1527, Department of Computer Science, University of Illinois, Urbana, Illinois, 1989.

[AS] Aronov, B., and Sharir, M. Triangles in space, or: Building (and analyzing) castles in the air. In *Proc. 4th ACM Symp. Comput. Geom.*, 1988, pp. 381–391.

[BO] Bentley, J. L., and Ottmann, T. A. Algorithms for reporting and counting geometric intersections. *IEEE Trans. Comput.* **28** (1979), 643–647.

[Ca] Canham, R. J. A theorem on arrangements of lines in the plane. *Isreal J. Math.* **7** (1969), 393–397.

[CD] Chazelle, B., and Dobkin, D. P. Intersection of convex objects in two and three dimensions. *J. Assoc. Comput. Mach.* **34** (1987), 1–27.

[Cl] Clarkson, K. New applications of random sampling in computational geometry. *Discrete Comput. Geom.* **2** (1987), 195–222.

[CEG*] Clarkson, K., Edelsbrunner, H., Guibas, L. J., Sharir, M., and Welzl, E. Combinatorial complexity bounds for arrangements of curves and spheres. *Discrete Comput. Geom.*, this issue, 99–160.

[CSY] Cole, R., Sharir M., and Yap, C. K. On k-hulls and related problems. *SIAM J. Comput.* **16** (1987), 61–77.

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

[EGH*] Edelsbrunner, H., Guibas, L. J., Hershberger, J., Seidel, R., Sharir, M., Snoeyink, J., and Welzl, E. Implicitly representing arrangements of lines or segments. *Discrete Comput. Geom.* **4** (1989), 433–466.

[EGSh] Edelsbrunner, H., Guibas, L. J., and Sharir, M. The complexity of many cells in arrangements of planes and related problems. *Discrete Comput. Geom.*, this issue, 197–216.

[EGSt] Edelsbrunner, H., Guibas, L. J., and Stolfi, J. Optimal point location in a monotone subdivision. *SIAM J. Comput.* **15** (1986), 317–340.

[EOS] Edelsbrunner, H., O'Rourke, J., and Seidel, R. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.* **15** (1986), 341–363.

[ES] Edelsbrunner, H., and Sharir, M. The maximum number of ways to stab n convex nonintersecting sets in the plane is $2n - 2$. *Discrete Comput. Geom.* **5** (1990), 35–42.

[EW1] Edelsbrunner, H., and Welzl, E. On the maximal number of edges of many faces in an arrangement. *J. Combin. Theory Ser. A* **41** (1986), 159–166.

[EW2] Edelsbrunner, H., and Welzl, E. Halfplanar range search in linear space and $O(n^{0.695})$ query time. *Inform. Process. Lett.* **23** (1986), 289–293.

[G] Grünbaum, B. *Convex Polytopes.* Wiley, London, 1967.

[GOS] Guibas, L. J., Overmars, M. H., and Sharir, M. Counting and reporting intersections in arrangements of line segments. Tech. Report 434, Computer Science Department, NYU, 1989.

[GSS] Guibas, L. J., Sharir, M., and Sifrony, S. On the general motion planning problem with two degrees of freedom. In *Proc. 4th ACM Symp. Comput. Geom.*, 1988, pp. 289–298.

[HS] Hart, S., and Sharir, M. Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes. *Combinatorica* **6** (1986), 151–177.

[HW] Haussler, D., and Welzl, E. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.* **2** (1987), 127–151.

[M] Moise, E. E. *Geometric Topology in Dimension 2 and 3.* Springer-Verlag, New York, 1977.

[O] O'Rourke, J. The signature of a plane curve. *SIAM J. Comput.* **15** (1986), 34–51.

[PSS] Pollack, R., Sharir, M., and Sifrony, S. Separating two simple polygons by a sequence of translations. *Discrete Comput. Geom.* **3** (1988), 123–136.

[PS] Preparata, F. P., and Shamos, M. I. *Computational Geometry—An Introduction.* Springer-Verlag, New York, 1985.

[SML] Schmitt, A., Müller, H., and Leister, W. Ray tracing algorithms—theory and practice. In *Theoretical Foundations of Computer Graphics and CAD* (R. A. Earnshaw, Ed.), NATO ASI Series, Vol. F40, Springer-Verlag, Berlin, 1988, pp. 997–1030.

[ST] Szemerédi, E., and Trotter, W. T. Extremal problems in discrete geometry. *Combinatorica* **3** (1983), 381–392.

[WS] Wiernik, A., and Sharir, M. Planar realization of nonlinear Davenport–Schinzel sequences by segments. *Discrete Comput. Geom.* **3** (1988), 15–47.

Received March 28, 1988, and in revised form July 10, 1989.