Discrete Comput Geom 5:533-573 (1990)

Discrete & Computational
Geometry

Partitioning Arrangements of Lines II: Applications*

Pankaj K. Agarwal†

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

Abstract. In this paper we present efficient deterministic algorithms for various problems involving lines or segments in the plane, using the partitioning algorithm described in a companion paper [A3]. These applications include: (i) an $O(m^{2/3}n^{2/3} \cdot \log^{2/3} n \cdot \log^{\omega/3}(m/\sqrt{n}) + (m+n)\log n)$ algorithm to compute all incidences between m points and n lines, where ω is a constant <3.33; (ii) an $O(m^{2/3}n^{2/3} \cdot \log^{5/3} n \cdot \log^{\omega/3}(m/\sqrt{n}) + (m+n)\log n)$ algorithm to compute m faces in an arrangement of n lines; (iii) an $O(n^{4/3}\log^{(\omega+2)/3} n)$ algorithm to count the number of intersections in a set of n segments; (iv) an $O(n^{4/3}\log^{(\omega+2)/3} n)$ algorithm to count "red-blue" intersections between two sets of segments, and (v) an $O(n^{3/2}\log^{\omega+1} n)$ algorithm to compute spanning trees with low stabbing number for a set of n points. We also present an algorithm that, given set of n points in the plane, preprocesses it, in time $O(n\sqrt{m}\log^{\omega+1/2} n)$, into a data structure of size $O(m)$ for $n\log n \le m \le n^2$, so that the number of points of S lying inside a query triangle can be computed in $O((n/\sqrt{m})\log^{3/2} n)$ time.

1. Introduction

In the first part of this paper [A3], we showed that

Theorem 1.1 [A3]. *Given a collection $\mathcal{L}$ of n lines in the plane and a parameter $1 \le r \le n$, the plane can be partitioned into $O(r^2)$ triangles, in time $O(nr\log n\log^{\omega} r)$,*

* Work on this paper has been supported by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grant DCR-83-20085, and by grants from the Digital Equipment Corporation and the IBM Corporation. A preliminary version of this paper appears in the *Proceedings of the 5th ACM Symposium on Computational Geometry*, 1989, pp. 11-22.

† Current address: Department of Computer Science, Duke University, Durham, NC 27706, USA.

so that no triangle meets more than $O(n/r)$ lines of $\mathcal{L}$ in its interior, where ω is some constant < 3.33.

This partitioning is useful to obtain divide-and-conquer algorithms for a variety of problems involving lines (or line segments) in the plane. Typically, an original problem involving the lines of $\mathcal{L}$ is split into $O(r^2)$ subproblems, one per triangle in the resulting partitioning, each involving only $O(n/r)$ lines of $\mathcal{L}$ meeting the corresponding triangle. (As mentioned in [A3], once we have such a partitioning of the plane, the lines intersecting the interior of each triangle, i.e., the set of lines involving each subproblem, can be easily computed in $O(nr)$ time.) These subproblems are then solved either by recursive application of the partitioning technique, or, if the size of the subproblems is sufficiently small, by some different and direct method.

In this second part of the paper we apply our partitioning algorithm to obtain fast algorithms for a variety of problems involving lines or segments in the plane. The problems that benefit from our algorithm have the common property that they can be solved efficiently using the random-sampling technique. Our algorithms for most of these problems have the same flavor. We divide the original problem into $O(r^2)$ subproblems, as explained above, then solve each subproblem directly by a simpler algorithm, and finally merge the results of these problems. A considerable part of this paper is devoted to the discussion of these simpler algorithms, and to details of the merging. In several applications the merging is trivial (e.g., in problems (i), (iv), and (v) below), but in other applications it may require some extra nontrivial techniques. The following list summarizes the results obtained in this paper:

(i) *Computing incidences between lines and points* (Section 2). Given a set of n lines and a set of m points in the plane, compute how many lines pass through each given point. (Alternatively, compute the lines passing through each point, or just determine whether any line passes through any point.) Edelsbrunner *et al.* [EGSh] have given a randomized algorithm for this problem whose expected running time is $O(m^{2/3-\delta}n^{2/3+2\delta} + (m+n)\log n)$, for any $\delta > 0$. A slightly improved, but still randomized, algorithm has been given in [EGH*]. We present a deterministic algorithm with $O(m^{2/3}n^{2/3}\log^{2/3} n \log^{\omega/3}(m/\sqrt{n}) + (m+n)\log n)$ time complexity. Since the maximum number of incidences between m points and n lines is $\Theta(m^{2/3}n^{2/3} + m + n)$, our algorithm is close to optimal in the worst case.

(ii) *Computing many faces in an arrangement of lines* (Section 3). Given a set of n lines and a set of m points in the plane, compute the faces in the arrangement of the lines containing the given points. Edelsbrunner *et al.* [EGSh] have given a randomized algorithm for this problem with expected running time $O(m^{2/3-\delta}n^{2/3+2\delta} + n\log n \log m)$, for any $\delta > 0$. As in the case of the incidence problem, a slightly better randomized algorithm has been given in [EGH*]. We present a deterministic $O(m^{2/3}n^{2/3}\log^{5/3} n \log^{\omega/3}(m/\sqrt{n}) + (m+n)\log n)$ algorithm, again coming close to optimal in the worst case (see [CEG*] for combinatorial bounds).

(iii) *Computing many faces in an arrangement of segments* (Section 4). This is the same problem as the previous one except that now we have a collection of segments instead of lines. The previous best solution is by Edelsbrunner *et al.* [EGSh], which is randomized and has expected running time $O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n)\log^2 n \log m)$, for any $\delta > 0$, where $\alpha(n)$ is a functional inverse of Ackermann's function. We present a deterministic algorithm with improved time complexity $O(m^{2/3}n^{2/3}\log n \log^{\omega/3+1}(n/\sqrt{m}) + n\log^3 n + m\log n)$.

(iv) *Counting segment intersections* (Section 5). We give a deterministic $O(n^{4/3}\log^{(\omega+2)/3} n)$ algorithm to count the number of intersections in a given collection of n segments; this is an improvement over Guibas *et al.*'s algorithm [GOS1], which counts the intersections in $O(n^{4/3+\delta})$ randomized expected time, for any $\delta > 0$.

(v) *Counting and reporting red-blue intersections* (Section 6). Given a set Γ_r of n_r "red" segments and another set Γ_b of n_b "blue" segments in the plane, count the number of intersections between Γ_r and Γ_b, or report all of them. (In this problem, we need to ignore the potentially large number of intersections within Γ_r or within Γ_b.) The previous best solution is by Agarwal and Sharir [AS], which reports all K red-blue intersections deterministically in $O((n_r\sqrt{n_b} + n_b\sqrt{n_r} + K)\log n)$ time, where $n = n_r + n_b$. We give a deterministic $O(n^{4/3}\log^{(\omega+2)/3} n)$ algorithm to count all red-blue intersections. It can also report all K red-blue intersections in time $O(n^{4/3}\log^{(\omega+2)/3} n + K)$.

(vi) *Implicit point-location problem* (Section 7). Given a collection of m points and a collection of (possibly intersecting) n triangles in the plane, find which points lie in the union of the triagles. This turns out to be a special case of a general problem of implicit point location in planar maps formed by overlapping figures. We present a deterministic algorithm with $O(m^{2/3}n^{2/3}\log^{2/3} n \log^{\omega/3}(n/\sqrt{m}) + (m+n)\log n)$ time complexity.

(vii) *Approximate half-plane range searching* (Section 8). Given a set S of n points in the plane and a parameter (not necessarily constant) $\varepsilon > 0$, preprocess them so that, for any query line ℓ, we can approximately count the number of points lying above ℓ with an error of at most $\pm \varepsilon n$. We give an algorithm that preprocesses S, in time $O((n/\varepsilon)\log n \log^{\omega}(1/\varepsilon))$, into a data structure of size $O(1/\varepsilon^2)$ so that a query can be answered in $O(\log n)$ time.

(viii) *Constructing spanning trees with low stabbing number* (Section 9). Given a set S of n points in the plane, we present an $O(n^{3/2}\log^{\omega+1} n)$ algorithm to construct a family of $k = O(\log n)$ spanning trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ of S with the property that, for any line ℓ, there is tree $\mathcal{T}_i$, such that ℓ intersects at most $O(\sqrt{n})$ edges of $\mathcal{T}_i$. Moreover, with additional preprocessing of $O(n\log n)$ time and $O(n)$ space, the tree $\mathcal{T}_i$ corresponding to a query line ℓ can be determined in $O(\log n)$ time. The previously best-known algorithm is by Matoušek [Ma1], which runs in $O(n^{7/4}\log^2 n)$ time, and, moreover, produces a stabbing number $O(\sqrt{n}\log^2 n)$ instead of $O(\sqrt{n})$.

(ix) *Space-query-time tradeoff in triangle range searching* (Section 10). Given a set S of n points in the plane, preprocess it so that, for any query triangle, we can quickly compute the number of points contained in that triangle. We give an

algorithm with $O((n/\sqrt{m}) \log^{3/2} n)$ query time, using $O(m)$ space. The preprocessing time is bounded by $O(n\sqrt{m} \log^{\omega+1/2} n)$. Similar bounds have been obtained independently by Chazelle [Ch3].

(x) *Overlapping planar maps.* Given two planar maps P, Q, and a bivariate function $F_P(x, y)$, $F_Q(x, y)$ associated with each of them, such that over each face of P the function F_P has some simple structure (e.g., it is constant, linear, or convex over each face), and similarly for Q, determine a point that minimizes $F_P(x, y) - F_Q(x, y)$. We show that if the maps satisfy certain conditions, then an optimal point can be computed in $O(n^{4/3} \log^{(\omega+2)/3} n)$ time, where n is the total complexity of the two maps. The details of this application can be found in [A2].

2. Computing or Detecting Incidences Between Points and Lines

Consider the following problem (see Fig. 1):

> Given a set $\mathscr{L} = \{\ell_1, \ldots, \ell_n\}$ of n lines and a set $P = \{p_1, \ldots, p_m\}$ of m points in the plane, for each point p_i compute the lines in $\mathscr{L}$ passing through it. This is an extension of Hopcroft's problem which asks whether there is a point in P lying on a line in $\mathscr{L}$.

Szemerédi and Trotter [STr] showed that the maximum number of incidences between n lines and m points is $\Theta(m^{2/3}n^{2/3} + m + n)$ (a much simpler proof, with a substantially smaller constant of proportionality, appears in [CEG*]). Edelsbrunner *et al.* [EGSh] have given a randomized algorithm for computing all incidences; its expected running time is $O(m^{2/3-\delta}n^{2/3+2\delta} + (m + n) \log n)$, for any $\delta > 0$ (see also [CSY]). Like many other randomized algorithms of this kind, this algorithm can be made deterministic without any additional overhead, using Matoušek's algorithm [Ma2]. A slightly faster randomized algorithm is given in [EGH*] with $O(m^{2/3}n^{2/3} \log^4 n + (m + n^{3/2}) \log^2 n)$ expected running time, which however is not known as yet to admit such "cheap" determinization. In this section we first present a very simple algorithm whose running time is roughly $m\sqrt{n} \log^{1/2} n$; this, combined with our partitioning algorithm, will yield a deterministic algorithm that is faster than the preceding ones.

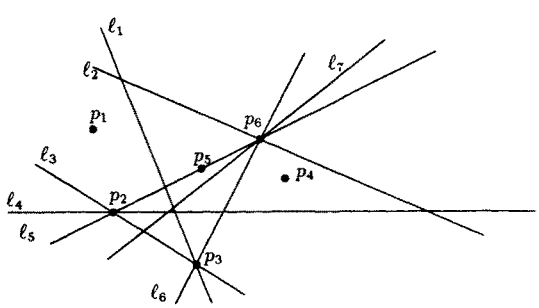

Fig. 1. An instance of the incidence problem.

We can assume that $m < n^2$, because otherwise we can compute all incidences in time $O(n^2 + m \log n) = O(m \log n)$ by constructing the arrangement of $\mathscr{L}$ and locating in it each of the points.

Divide the set P into t disjoint subsets $P_1, \ldots, P_t$, each of size at most $\lceil m/t \rceil$. For each P_i, we compute the incidences between P_i and $\mathscr{L}$ as follows. Dualize the lines ℓ_j to points ℓ_j^*, and the points p_j to lines p_j^*, so we have a set p_i^* of $\lceil m/t \rceil$ lines and a set $\mathscr{L}^*$ of n points in the plane. Since duality preserves incidences, it suffices to determine the points of $\mathscr{L}^*$ lying on each line p_j^*; this can be done by constructing the arrangement $\mathscr{A}(P_i^*)$, processing it for fast point location as in [EGSt], and locating in it each of the points of $\mathscr{L}^*$. The cost of all this is $O(m^2/t^2 + n \log n)$ (see [EOS] and [EGSt]). Summing over all P_i's, the overall running time becomes

$$T(m, n) = O\left(t\left(\frac{m^2}{t^2} + n \log n\right)\right) = O\left(\frac{m^2}{t} + nt \log n\right).$$

For $t = \lceil m/\sqrt{n \log n} \rceil$, the total running time is

$$T(m, n) = O(m\sqrt{n} \log^{1/2} n + n \log n). \tag{2.1}$$

Next, we describe the main algorithm. First, partition the plane into $M = O(r^2)$ triangles $\Delta_1, \ldots, \Delta_M$ so that the interior of each triangle meets $O(n/r)$ lines of $\mathscr{L}$, for some r to be specified later. Let P_i (resp. $\mathscr{L}_i$) denote the set of points (resp. lines) lying inside (resp. meeting the interior of) the triangle Δ_i; let n_i (resp. m_i) be the size of $\mathscr{L}_i$ (resp. P_i). The sets $\mathscr{L}_i$ are computed by determining the triangles intersected by each line of $\mathscr{L}$, as described in [A3], and the sets P_i, are obtained, in time $O((r^2 + m) \log r)$, by locating each point of P in the planar subdivision formed by the triangles Δ_i. The incidences between the lines and the points lying on the triangle boundaries can be easily computed in time $O((m + nr) \log n)$, once we have distributed the lines over the triangles. We then apply, for each triangle Δ_i, the above algorithm to determine the incidences between P_i and $\mathscr{L}_i$ within Δ_i. Since partitioning the plane takes $O(nr \log n \log^{\omega} r)$ time (see Theorem 1.1), the total time $T(m, n)$ spent in computing the incidences between n lines and m points is therefore at most

$$\begin{aligned} T(m, n) &\leq \sum_{i=1}^{M} T(m_i, n_i) + O(r^2 \log r + m \log n + nr \log n \log^{\omega} r) \\ &= \sum_{i=1}^{M} O(m_i \sqrt{n_i} \log^{1/2} n_i + n_i \log n_i) + O((m + nr \log^{\omega} r) \log n). \end{aligned} \tag{2.2}$$

Since $n_i = O(n/r)$, (2.2) becomes

$$\begin{aligned} T(m, n) &= O\left(\sqrt{\frac{n}{r}} \log^{1/2} n \cdot \sum_{i=1}^{M} m_i\right) + O((m + nr \log^{\omega} r) \log n) \\ &= O\left(\frac{m\sqrt{n}}{\sqrt{r}} \log^{1/2} n + m \log n + nr \log^{\omega} r \log n\right) \end{aligned} \tag{2.3}$$

because $\sum_{i=1}^{m} m_i = m$. Now choose

$$r = \max\left\{\frac{m^{2/3}}{n^{1/3}\log^{1/3} n \log^{2\omega/3}(m/\sqrt{n})}, 2\right\};$$

since $m < n^2$, we have $r \leq n$ as required. Therefore (2.3) gives

$$T(m, n) = O\left(m^{2/3}n^{2/3}\log^{2/3} n \cdot \log^{\omega/3}\frac{m}{\sqrt{n}} + (m+n)\log n\right).$$

Hence, combining this with the case $m \geq n^2$, we have

Theorem 2.1. *Given a set of n lines and a set of m points in the plane, we can compute the lines passing through each point in time $O(m^{2/3}n^{2/3}\log^{2/3} n \cdot \log^{\omega/3}(m/\sqrt{n}) + (m+n)\log n)$. (In particular, we can determine whether any line passes through any point within the same amount of time.)*

3. Computing Many Faces in Arrangements of Lines

Next we consider the following problem:

> Given a set $\mathscr{L} = \{\ell_1, \ldots, \ell_n\}$ of n lines and a set $P = \{p_1, \ldots, p_m\}$ of m points, compute the faces of $\mathscr{A}(\mathscr{L})$ containing one or more points of P.

Clarkson *et al.* [CEG*] have proved that the combinatorial complexity of m distinct faces in any arrangement of n lines in the plane is $O(m^{2/3}n^{2/3} + n)$ (see also [Ca]), and Edelsbrunner *et al.* [EGSh] have given a randomized algorithm to compute m distinct faces, whose expected running time is $O(m^{2/3-\delta}n^{2/3+2\delta} + n\log n \log m)$, for any $\delta > 0$. This algorithm can be made deterministic, without substantially changing its time complexity, using the original technique of Matoušek [Ma2]. As in the case of the incidence problem, a slightly faster randomized algorithm, for large values of m, is presented in [EGH*] and has $O(n^{3/2}\log^2 n + m^{2/3}n^{2/3}\log^4 n)$ expected running time, but we do not know of any way to make it deterministic without substantially increasing its running time. We present a deterministic algorithm that computes these faces in time $O(m^{2/3}n^{2/3}\log^{5/3} n \log^{\omega/3}(m/\sqrt{n}) + n\log n)$.

Similar to the previous section, we first give a slower $O(m\sqrt{n}\log^2 n + n\log n)$ algorithm for this problem and then, using the same divide-and-conquer technique, we obtain an algorithm with the asserted time bound. Without loss of generality we can assume that $m \leq n^2$, for otherwise the faces can be computed in time $O(m\log n)$ by constructing the entire arrangement $\mathscr{A}(\mathscr{L})$. Our slower algorithm works as follows.

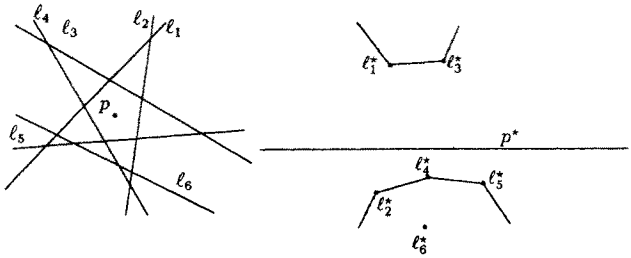

Fig. 2. A face in an arrangement of lines, and its dual.

Partition the set P into t disjoint sets $P_1, \ldots, P_t$ so that P_i contains $m_i \leq \lceil m/t \rceil$ points. We show how to compute the faces of $\mathscr{A}(\mathscr{L})$ containing the points of P_i, and repeat this procedure for all $i \leq t$. Let $\mathscr{L}^*$ denote the set of points dual to the lines $\mathscr{L}$, and let P_i^* denote the set of lines dual to the points in P_i. Let f be a face of $\mathscr{A}(\mathscr{L})$ containing some point p. For each line $\ell \in \mathscr{L}$ bounding f, its dual point ℓ^* is such that the dual line p^* can be moved (actually rotated around some point) to touch ℓ^*, without crossing any other point of $\mathscr{L}^*$ while rotating. In other words, the dual of the face f containing a point p corresponds to the portions of the convex hulls $\mathrm{CH}(\mathscr{L}^* \cap (p^*)^+)$ and $\mathrm{CH}(\mathscr{L}^* \cap (p^*)^-)$ between their common tangents, where $(p^*)^+$, $(p^*)^-$ denote the half-planes lying respectively above and below p^*, as shown in Fig. 2. Therefore, it suffices to describe how to compute the convex hull of the points in $\mathscr{L}^*$ lying above or below the line p^* for each line $p^* \in P_i^*$.

First, compute the arrangement $\mathscr{A}(P_i^*)$. Let $\mathscr{D}$ denote the dual of the planar graph formed by $\mathscr{A}(P_i^*)$, i.e., the vertices of $\mathscr{D}$ correspond to the faces of $\mathscr{A}(P_i^*)$, and there is an edge φ_{jk} between two vertices v_j, v_k of $\mathscr{D}$ if the corresponding faces f_j, f_k of $\mathscr{A}(P_i^*)$ share an edge e_{jk} in $\mathscr{A}(P_i^*)$ (see Fig. 3). Let $\mathscr{L}_j^* \subseteq \mathscr{L}^*$ denote the set of points lying in the face $f_j \in \mathscr{A}(P_i^*)$. For each $\mathscr{L}_j^*$, compute its convex hull $\mathrm{CH}(\mathscr{L}_j^*)$. We associate $\mathscr{L}_j^*$ and its hull with node v_j of $\mathscr{D}$.

Let $\mathscr{T}$ denote any spanning tree of $\mathscr{D}$; it can be easily computed in time $O(m_i^2)$. If $\mathscr{T}$ contains a subtree of $\mathscr{T}$, all of whose nodes are associated with empty subsets of $\mathscr{L}^*$, we remove that subtree from $\mathscr{T}$. It is easily seen that a line $p^* \in P_i^*$ intersects at most m_i edges of $\mathscr{T}$ (in the sense that the two faces of $\mathscr{A}(P_i^*)$ connected by such an edge lie on different sides of p^*). Perform a depth-first search on $\mathscr{T}$ and connect the

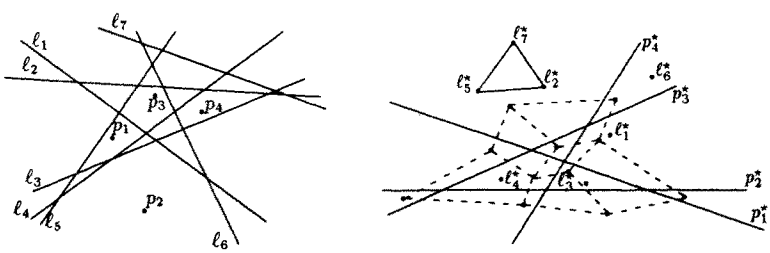

Fig. 3. Arrangements $\mathscr{A}(\mathscr{L})$, $\mathscr{A}(P_i^*)$, and the dual graph $\mathscr{D}$.

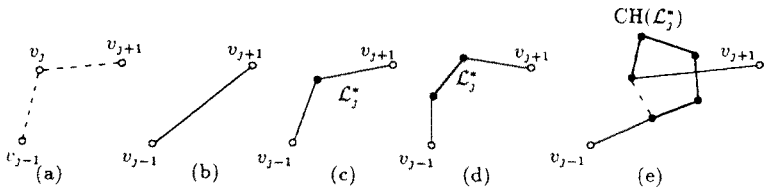

Fig. 4. Transforming a vertex v_j of Π: (a) vertex v_j of Π, (b) v_j is deleted from Π, (c)–(e) v_j is replaced by $\mathrm{CH}(\mathcal{L}_j^*)$.

vertices of $\mathcal{T}$ in the order they are first visited by the depth-first traversal; this gives a spanning path Π with the property that a line $p^* \in P_i^*$ intersects at most $2m_i$ edges of Π (in the same sense as above, see [CW]), and that each edge of Π is intersected by exactly one line of P_i^*. Next we construct a spanning path $\mathscr{C}$ of $\mathcal{L}^*$ from Π by modifying each vertex v_j of Π, depending on the cardinality of $\mathrm{CH}(\mathcal{L}_j^*)$. There are three cases to consider:

(i) $|\mathrm{CH}(\mathcal{L}_j^*)| = 0$: remove the vertex v_j and the edges $\varphi_{j-1,j}$, $\varphi_{j,j+1}$ from Π, and add the edge $\varphi_{j-1,j+1}$ to Π (Fig. 4(b)); this shortcuting may be repeated several times if needed, producing at the end a shortcut edge $\varphi_{kk'}$.

(ii) $|\mathrm{CH}(\mathcal{L}_j^*)| \leq 1$: replace the vertex v_j by $\mathrm{CH}(\mathcal{L}_j^*)$ (Fig. 4(c)).

(iii) $|\mathrm{CH}(\mathcal{L}_j^*)| \geq 2$: let ℓ_x^*, ℓ_y^* be two adjacent vertices of $\mathrm{CH}(\mathcal{L}_j^*)$. Replace v_j by $\mathrm{CH}(\mathcal{L}_j^*)$, make the edge $\varphi_{j-1,j}$ (resp. $\varphi_{j,j+1}$) incident to ℓ_x^* (resp. ℓ_y^*) (Fig. 4(d), (e)), and if $|\mathrm{CH}(\mathcal{L}_j^*)| > 2$, then remove the edge $\overline{\ell_x^* \ell_y^*}$ from $\mathrm{CH}(\mathcal{L}_j^*)$ (Fig. 4(e)).

It is easily seen that the resulting structure is a spanning path $\mathscr{C}$ of $\mathcal{L}^*$ (see Fig. 5).

Lemma 3.1. *A line $p^* \in P_i^*$ intersects at most $2m_i$ edges of $\mathscr{C}$.*

Proof. Let $p^* \in P_i^*$ be a line intersecting s edges of Π. We prove that p^* intersects at most s edges of $\mathscr{C}$, by showing that each intersection between p^* and an edge of $\mathscr{C}$ can be charged to an edge φ of Π intersecting p^*, in such a way that no edge of Π is charged more than once. There are three types of edges in $\mathscr{C}$:

(i) edges that were already present in Π (e.g., $\overline{\ell_1^* \ell_6^*}$ in Fig. 5),

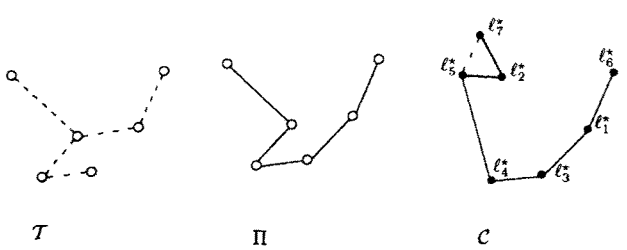

Fig. 5. Spanning tree $\mathcal{T}$ and spanning paths Π and $\mathscr{C}$ for points of $\mathcal{L}^*$ shown in Fig. 3.

(ii) edges of $\mathrm{CH}(\mathscr{L}_j^*)$ for some $v_j \in \mathscr{T}$ (e.g., $\overline{\ell_2^*\ell_5^*}$ in Fig. 5), and
(iii) edges that were introduced while removing a vertex of Π (e.g., $\overline{\ell_4^*\ell_5^*}$ in Fig. 5).

We charge an intersection of P^* with an edge of type (i) to the edge itself. Edges of type (ii) do not intersect p^*, because $\mathrm{CH}(\mathscr{L}_j^*)$ lies inside a face of $\mathscr{A}(P_i^*)$. Finally, if p^* intersects an edge $\varphi_{k,k'}$ of type (iii) (i.e., a shortcut edge introduced while deleting vertices from Π), then p^* must intersect at least one edge $\varphi_{j,j+1}$ of Π for $j = k, k+1, \ldots, k'-1$. We can therefore charge this intersection to $\varphi_{j,j+1}$. It is easily seen that we charge only those edges of Π that intersect p^* and no edge is charged twice. Hence p^* intersects at most $s \le 2m_i$ edges of $\mathscr{T}$. □

Edelsbrunner *et al.* [EGH*] have shown that if T is a spanning path of a set S of k points in the plane, then T can be preprocessed in $O(k \log k)$ time so that, for any line ℓ intersecting at most s edges of T, $\mathrm{CH}(S \cap \ell^+)$ can be computed in $O(s \log^3 k)$ time. Since in our case $k = n$ and $s \le 2m_i$, $\mathrm{CH}(\mathscr{L}^* \cap p^*)$, for $p^* \in P_i^*$, can be computed in time $O(m_i \log^3 n)$, which implies that the total time spent in computing the faces in $\mathscr{A}(\mathscr{L})$ containing the points of P_i is bounded by $O((m^2/t^2)\log^3 n + n \log n)$.

However, Edelsbrunner *et al.*'s procedure returns only an implicit representation, which they referred to as the "necklace representation," of the desired faces. That is, the output of their algorithm is a list of pointers, each pointing to some node storing a disjoint portion of the convex hull, intermixed with "bridging edges" that connect these portions in the overall hull. If we want to compute each desired face explicitly, we have to traverse all the hull portions that the algorithm points to, and the time to compute a single face f_j becomes $O(m_i \log^3 n + k_j)$, where k_j is the number of edges in f_j. Therefore, the total time spent in computing the faces containing the points of P_i is $O(m_i^2 \log^3 n + n \log n + \sum_{p_j \in P_i} k_j)$. But in the worst case $\sum_{p_j \in P_i} k_j$ could be as large as $\Theta(m_i n_i)$, e.g., when all of the points lie in the same face, which happens to be bounded by all the lines of $\mathscr{L}$. This bound is too large for our purposes, which means that we cannot afford to output the same face too many times. We circumvent this problem by modifying the above algorithm as follows. Suppose we have already computed the faces containing $p_1, \ldots, p_j$ of P_i, and we are about to compute the face f_{j+1} containing p_{j+1}. Before computing this face we first check whether p_{j+1} lies in any of the faces computed so far; we compute f_{j+1}, as described above, only if it is indeed a new face. Since each face of $\mathscr{A}(\mathscr{L})$ is a convex polygon, we can easily test p_{j+1} for containment in each of the already-computed faces of $\mathscr{A}(\mathscr{L})$ in $O(\log n)$ time, so the total time needed to decide whether f_{j+1} should be computed is $O(j \log n)$. Thus, the total time required to compute the collection S of the desired faces is

$$T(m_i, n) = \sum_{j=1}^{m_i} O(m_i \log^3 n + j \log n) + O\left(\sum_{f_j \in S} |f_j|\right) + O(n \log n)$$

$$= O(m_i^2 \log^3 n + n \log n) + O\left(\sum_{f_j \in S} |f_j|\right).$$

Edelsbrunner and Welzl [EW2] (see also [Ca]) have proved that the complexity of m distinct faces in an arrangement of n lines is $O(m\sqrt{n})$. Therefore

$$T(m_i, n) = O(m_i^2 \log^3 n + n \log n) + O(m_i\sqrt{n}).$$

Since $m_i \leq \lceil m/t \rceil$, summing over all P_i's we obtain

$$T(m, n) = \sum_{i=1}^{t} O\left(\frac{m^2}{t^2} \log^3 n + n \log n + \frac{m}{t}\sqrt{n}\right)$$

$$= O\left(\frac{m^2}{t} \log^3 n + nt \log n + m\sqrt{n}\right).$$

Choosing $t = \lceil (m \log n)/\sqrt{n} \rceil$, we obtain $T(m, n) = O(m\sqrt{n} \log^2 n + n \log n)$.

Remark 3.2. We believe that using, in the above procedure, the algorithm of [EGH*] of merging the convex hulls to obtain the explicit face representation is an overkill, and a simpler, more naive solution should exist. But at present we do not know how to simplify the algorithm.

Now we describe the main algorithm. As in the previous section, we partition the plane into $M = O(r^2)$ triangles $\Delta_1, \ldots, \Delta_M$ each of which meets $O(n/r)$ lines of $\mathscr{L}$. Let P_i (resp. $\mathscr{L}_i$) denote the set of points of P (resp. lines of $\mathscr{L}$) contained in (resp. meeting) Δ_i, and let $f_i(p)$ denote the face of $\mathscr{A}(\mathscr{L}_i)$ containing a point p. The *zone* of Δ_i in $\mathscr{A}(\mathscr{L}_i)$ is defined as the collection of the face portions $f \cap \Delta_i$, for all faces $f \in \mathscr{A}(\mathscr{L}_i)$, that intersect the boundary of Δ_i (see Fig. 6). Clarkson *et al.* [CEG*] have observed that the total number of edges in the zone of Δ_i is $O(n/r)$ (see also [CGL] and [EOS], where a zone is defined with respect to a half-plane). If a face $f_i(p)$ is fully contained in the interior of Δ_i, then $f_i(p) = f(p)$. Otherwise if $f_i(p)$ intersects the boundary of Δ_i, then it is a face of the zone of Δ_i. Moreover, if a face $f \in \mathscr{A}(\mathscr{L})$ does not lie in the interior of a triangle Δ_i, it is split into two or more pieces, each being a face in the zone of some triangle. Also, such a face f intersects a triangle Δ_i if and only if f is a face in the zone of Δ_i. Thus, all the faces in $\mathscr{A}(\mathscr{L})$ containing the points of P can be obtained by computing, for every Δ_i, (i) the faces of $\mathscr{A}(\mathscr{L}_i)$ that contain the points of P_i, and (ii) the zone of Δ_i. The faces of $\mathscr{A}(\mathscr{L})$ (containing points of P) that are split among the zones, can be easily glued together by matching their edges that lie on triangle edges.

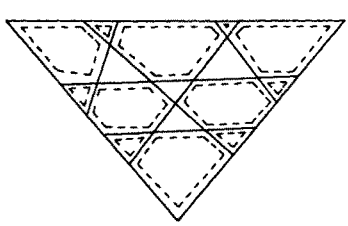

Fig. 6. Zone of a triangle Δ_i.

Edelsbrunner and Guibas [EG] have given an $O(n \log n)$ algorithm to compute a zone with respect to a half-plane in an arrangement of n lines. The same algorithm can be applied to calculate the zone of each Δ_i. As for computing the faces that lies in the interior of Δ_i, we use the simplified algorithm described above. Thus, the total time spent in processing Δ_i is $O(m_i\sqrt{n_i}\log^2 n_i + n_i \log n_i)$. Finally, the total time spent in merging the zones is $O(nr \log n)$ because zones of two different triangles do not intersect, and each zone has $O(n/r)$ edges. Hence the total time $T(m, n)$ spent in computing m distinct faces in an arrangement of n lines in the plane is (provided $m \leq n^2$)

$$\begin{aligned} T(m, n) &= \sum_{i=1}^{M} O(m_i\sqrt{n_i}\log^2 n_i + n_i \log n_i) + O(nr \log n) + O(nr \log n \log^\omega r) \\ &= O\left(\sqrt{\frac{n}{r}}\log^2 n \sum_{i=1}^{M} m_i + nr \log n\right) + O(nr \log n \log^\omega r) \\ &\quad \left(\text{because } n_i \leq \frac{n}{r} \text{ and } M = O(r^2)\right) \\ &= O\left(\frac{m\sqrt{n}}{\sqrt{r}}\log^2 n + nr \log n \log^\omega r\right), \end{aligned}$$

because $\sum_{i=1}^{m} m_i = m$. For

$$r = \max\left\{\frac{m^{2/3}\log^{2/3} n}{n^{1/3}\log^{2\omega/3}(m/\sqrt{n})}, 2\right\},$$

the above bound becomes

$$T(m, n) = O\left(m^{2/3}n^{2/3}\log^{5/3} n \log^{\omega/3}\frac{m}{\sqrt{n}} + n \log n\right).$$

Combining this with the trivial bound $O(m \log n)$, for $m > n^2$, we obtain.

Theorem 3.3. *Given a set of n lines in the plane, we can compute the faces of its arrangement that contain m given points in time*

$$O(m^{2/3}n^{2/3}\log^{5/3} n \log^{\omega/3}(m/\sqrt{n}) + (m + n)\log n).$$

4. Computing Many Faces in Arrangements of Segments

Consider the following problem:

Given a set $\mathcal{G} = \{e_1, \ldots, e_n\}$ of n segments and a set $P = \{p_1, \ldots, p_m\}$ of m points, compute the faces of $\mathcal{A}(\mathcal{G})$ containing the points of P.

Aronov *et al.* [AEGS] have shown that the combinatorial complexity of m distinct faces in an arrangement of n segments is bounded by

$$O(m^{2/3}n^{2/3} + n\alpha(n) + n \log m).$$

Edelsbrunner *et al.* [EGSh] have given a randomized algorithm to compute m distinct faces in an arrangement of n segments whose expected running time is $O(m^{2/3-\delta}n^{2/3+2\delta} + n\alpha(n) \log m \log^2 n)$, for any $\delta > 0$. Our algorithm for computing many faces in an arrangement of lines cannot be easily extended to the case of segments, so we present an alternative technique that proceeds by applying the partitioning algorithm in the dual plane rather than in the primal. Our algorithm is closely related to the proof of the combinatorial bound given in [AEGS]. Again we assume that $m \leq n^2$ for otherwise we can compute the faces in $O(m \log n)$ time by constructing the entire arrangement $\mathcal{A}(\mathcal{G})$.

Let ℓ denote the line containing the segment e of $\mathcal{G}$. Dualize each line ℓ to a point ℓ^*, and each point p of P to a line p^*; this yields a set P^* of m lines, and a set $\mathcal{L}^*$ of n points in the dual plane. Partition the dual plane into $t = O(r^2)$ triangles $\Delta_1', \ldots, \Delta_t'$ so that no triangle meets more than $O(m/r)$ lines of P^*. By Theorem 1.1, this can be done in $O(mr \log m \log^\omega r)$ time. If a triangle Δ_i' contains $n_i > n/r^2$ points of $\mathcal{L}^*$, split it further into $\lceil n_i r^2/n \rceil$ triangles, none of which contains more than n/r^2 points. Clearly, the distribution of the points of $\mathcal{L}^*$ among the triangles and the further partitioning of the triangles can be done in $O(n \log n)$ time. Let $\Delta_1, \ldots, \Delta_M$ denote the set of resulting triangles; we still have $M = O(r^2)$. Let $\mathcal{L}_i^*$ denote the set of points contained in Δ_i, and let P_i^* denote the set of lines meeting Δ_i. Let $\mathcal{G}_i$ denote the set of segments corresponding to the points $\mathcal{L}_i^*$. If a line p_j^* does not meet Δ_i, then p_j lies either above all lines containing the segments of $\mathcal{G}_i$ or below all such lines, which implies that p_j lies in the unbounded face of $\mathcal{A}(\mathcal{G}_i)$. Hence, for each subcollection $\mathcal{G}_i$, it suffices to compute the unbounded face of $\mathcal{A}(\mathcal{G}_i)$ and the faces that contain the points of P_i. As a matter of fact, we compute the entire arrangement $\mathcal{A}(\mathcal{G}_i)$ in time $O(n^2/r^4)$, and select the desired faces from it. Let $f_i(p)$ denote the face of $\mathcal{A}(\mathcal{G}_i)$ containing the point p. Note that the face $f(p)$ of $\mathcal{A}(\mathcal{G})$ containing p is the connected component of $\bigcap_{i=1}^{M} f_i(p)$ containing p. Therefore, for each $p \in P_i$, we have to "merge", i.e., compute the connected component containing p of the intersection of, all M corresponding faces.

Recall that our algorithm [A3] first computes r approximate levels, which are disjoint polygonal chains with a total of $O(r^2)$ vertices, and then triangulates each "corridor" lying between two adjacent polygonal chains. We construct a binary tree $\mathcal{T}$ of height $H = O(\log r)$ whose leaves correspond to these triangles and whose root corresponding to the enclosing rectangle $\mathbf{R}$ (see [AEGS]). We first construct a binary tree $\mathcal{T}_C$, as described in [AEGS], for each corridor C on the set of triangles lying in C so that the preorder traversal of $\mathcal{T}_C$ visits the leaves (i.e., the triangles in C) in the order in which they appear along C from left to right (see Fig. 7). Binary tree $\mathcal{T}$ is then constructed with the trees $\mathcal{T}_C$ as its leaves, in a similar manner.

Each node v of $\mathcal{T}$ is associated with a simply connected region $\mathcal{P}_v$, which is the union of the regions associated with the leaves of the subtree $\mathcal{T}_v$ of $\mathcal{T}$ rooted at v (the construction of $\mathcal{T}$ implies that each $\mathcal{P}_v$ is simply connected). For each node v

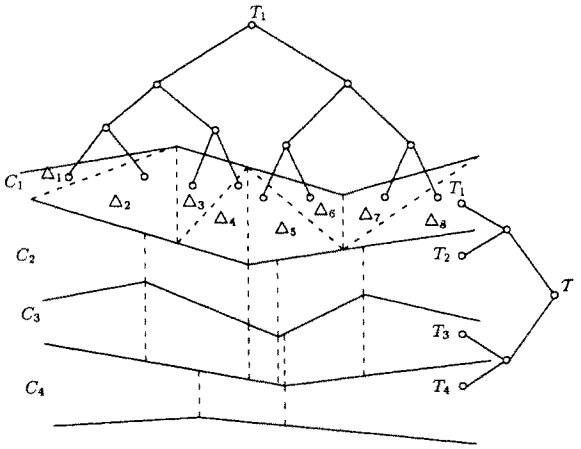

Fig. 7. $\mathscr{T}_C = T_1$ and $\mathscr{T}$.

of $\mathscr{T}$, let $\mathscr{G}_v = \bigcup_{\Delta_i \subseteq \mathscr{P}_v} \mathscr{G}_i$ and $\mathscr{P}_v = \bigcup_{\Delta_i \subseteq \mathscr{P}_v} P_i$. Let $n_v = |\mathscr{G}_v|$ and $m_v = |P_v|$. Observe that any point $p \in P - P_v$ lies either above all the lines containing the segments of $\mathscr{G}_v$ or below all these lines, and therefore all these points lie in the unbounded face of $\mathscr{A}(\mathscr{G}_v)$. Let w and z denote the children of the interior node v. It is easily seen that $P_v = P_w \cup P_z$. For every node v of $\mathscr{T}$, we compute the unbounded face of $\mathscr{A}(\mathscr{G}_v)$ and the faces containing the points of P_v. Let F_v denote the set of these faces and let R_v denote the total number of edges in the faces of F_v. Note that the face $f_v(p)$ of $\mathscr{A}(\mathscr{G}_v)$ is the connected component of $f_w(p) \cap f_z(p)$ that contains the point p, where $f_w(p)$ (resp. $f_z(p)$) is the face of $\mathscr{A}(\mathscr{G}_w)$ (resp. $\mathscr{A}(\mathscr{G}_z)$) containing the point p. Thus if we have already computed F_w and F_z, then F_v can be computed by applying the "red-blue merge" described in [EGSh]. Let $\mathscr{M}_v$ denote the time spent in merging F_w and F_z. It follows from the analysis of [EGSh] that

$$\mathscr{M}_v = O((R_v + m_v + n_v \alpha(n_v)) \log n_v). \tag{4.1}$$

Therefore, the total time $\mathscr{M}(m, n)$ spent in merging the faces is

$$\begin{aligned} \mathscr{M}(m, n) &= \sum_{v \in \mathscr{T}} O((R_v + m_v + n_v \alpha(n_v)) \log n_v) \\ &= \sum_{i=1}^{H} \sum_{h(v)=i} O((R_v + m_v + n_v \alpha(n_v)) \log n_v) \end{aligned} \tag{4.2}$$

where $h(v)$ is the height of v. But it has been proved in [EGSh] that

$$R_v \leq R_w + R_z + 4m_v + 6n_v. \tag{4.3}$$

Let $\mathscr{U}_v$ (resp. $\mathscr{I}_v$) denote the set of leaves (resp. interior nodes) in the subtree $\mathscr{T}_v$. If $h(v) = i$, then by (4.3)

$$R_v \leq \sum_{u \in \mathscr{U}_v} R_u + 4 \sum_{z \in \mathscr{I}_v} m_z + O(n_v \cdot i),$$

where the last term follows from the fact that $\sum_x n_x$, over all nodes at the same level of $\mathscr{T}_v$, is n_v, and the height of v is i. Let $k_v = |\mathscr{U}_v|$ denote the number of leaves of $\mathscr{T}_v$. As shown in [AEGS], the special way in which $\mathscr{T}$ was constructed guarantees that

$$m_v \leq \frac{ck_v m}{r} + 1, \tag{4.4}$$

where c is some constant > 0. Moreover, for each leaf u of $\mathscr{T}$, $|n_u| = O(n/r^2)$. Therefore $R_u = O(n^2/r^4)$, and

$$R_v = O\left(O\left(k_v \frac{n^2}{r^4}\right) + \sum_{z \in \mathscr{Z}_v} \left(\frac{k_z m}{r} + 1\right) + n_v \cdot i\right).$$

which implies that

$$\mathscr{M}(m, n) = \sum_{i=1}^{H} \sum_{h(v)=i} O\left(\left(\frac{k_v n^2}{r^4} + \sum_{z \in \mathscr{Z}_v} \frac{k_z m}{r} + n_v(\alpha(n) + i)\right) \log n\right).$$

It can be easily proved that

$$\sum_{h(v)=i} k_v = O(r^2), \qquad \sum_{h(v)=i} n_v = n, \quad \text{and} \quad \sum_{h(v)=i} \sum_{z \in \mathscr{Z}_v} k_z = O(ir^2).$$

Therefore

$$\begin{aligned}\mathscr{M}(m, n) &= \sum_{i=1}^{H} O\left(\left(\frac{n^2}{r^2} + imr + n(\alpha(n) + i)\right) \log n\right) \\ &= O\left(\left(\frac{n^2}{r^2} \log r + n\alpha(n) \log r + (n + mr) \log^2 r\right) \log n\right),\end{aligned}$$

because $H = O(\log r)$.

Now going back to the original problem, we spent $O(mr \log m \log^{\omega} r)$ time in partitioning the plane into M triangles, and $O(n^2/r^4)$ time in constructing $\mathscr{A}(\mathscr{G}_i)$ for each Δ_i (see [EOS]). Thus, the total time $T(m, n)$ spent in computing m distinct faces of an arrangement of n segments in the plane is at most

$$\begin{aligned}T(m, n) &= \sum_{i=1}^{M} O\left(\frac{n^2}{r^4}\right) + O(mr \log m \log^{\omega} r) \\ &\quad + O\left(\left(\frac{n^2}{r^2} \log r + n\alpha(n) \log r + (n + mr) \log^2 r\right) \log n\right) \\ &= O\left(\left(\frac{n^2}{r^2} + mr \log^{\omega - 1} r + n\alpha(n) + n \log r\right) \log n \log r\right).\end{aligned}$$

Hence, by choosing

$$r = \max\left\{\frac{n^{2/3}}{m^{1/3}\log^{(\omega-1)/3}(n/\sqrt{m})}, 2\right\},$$

we obtain

$$T(m,n) = O\left(m^{2/3}n^{2/3}\log n \log^{(2\omega+1)/3}\frac{n}{\sqrt{m}} + n\log n\log^2\frac{n}{\sqrt{m}} + m\log n\right)$$
$$= O\left(m^{2/3}n^{2/3}\log n\log^{(2\omega+1)/3}\frac{n}{\sqrt{m}} + n\log^3 n + m\log n\right).$$

Theorem 4.1. *The faces of an arrangement of n line segments, which contain m given points, can be computed in time*

$$O\left(m^{2/3}n^{2/3}\log n\log^{(2\omega+1)/3}\frac{n}{\sqrt{m}} + n\log^3 n + m\log n\right).$$

Remark 4.2. If we partition Δ_i' into $\lceil n_i/(\sqrt{m/r}\log^{1/2} r)\rceil$ triangles (instead of $\lceil n_i r^2/n\rceil$), each containing at most $\sqrt{m/r}\log^{1/2} r$ points of $\mathscr{L}^*$, and choose $r = \max\{(n^{2/3}/m^{1/3})\log^{2\omega/3-1}(n/\sqrt{m}), 2\}$, then the running time of the algorithm can be improved slightly to

$$O\left(m^{2/3}n^{2/3}\log n\log^{\omega/3+1}\frac{n}{\sqrt{m}} + n\log^3 n + m\log n\right).$$

5. Counting Segment Intersections

In this section we consider the following problem:

> Given a set $\mathscr{G} = \{e_1, \ldots, e_n\}$ of n line segments in the plane, we wish to count the number of intersection points between them.

This is a variant of one of the most widely studied problems in computational geometry, namely that of reporting all intersections (see [BO], [B], [Ch1], and [CE]). The recent algorithm of Chazelle and Edelsbrunner [CE] reports all k intersection points in time $O(n\log n + k)$ using $O(n + k)$ space. Although it has optimal running time, it requires quadratic working storage in the worst case. Guibas *et al.* [GOS2] gave an $O(n^{4/3+\delta} + k)$ randomized algorithm, for any $\delta > 0$, using only $O(n)$ working storage (see also [Cl2] and [Mu]). The only algorithms known for counting the intersection points in time that does not depend on k are by Chazelle [Ch1] and by Guibas *et al.* [GOS2]. The latter algorithm is faster but randomized, and has expected running time $O(n^{4/3+\delta})$, for any $\delta > 0$. We modify

Guibas *et al.*'s algorithm to give a slightly faster and deterministic algorithm, although the space requirement goes up roughly to $n^{4/3}$. Their algorithm relies on a procedure that, for a given triangle Δ, counts the number of intersection points contained in Δ in $O((m^2 + n)\log n)$ time, where n is the number of segments meeting Δ, and $m \leq n$ is the number of segments having at least one of their endpoints inside Δ. For the sake of completeness, we briefly overview this procedure because we also make use of it.

Partition the segments of $\mathcal{G}$ meeting Δ into two subsets:

(i) $\mathcal{G}_1$: "long" segments of $\mathcal{G}$ whose endpoints do not lie inside Δ.
(ii) $\mathcal{G}_s$: "short" segments of $\mathcal{G}$ having at least one endpoint inside Δ.

There are three types of intersections to be counted:

- "Short-short" intersection: intersections between the segments of $\mathcal{G}_s$.
- "Long-long" intersections: intersections between the segments of $\mathcal{G}_1$.
- "Long-short" intersections: intersections between a segment of $\mathcal{G}_1$ and another segment of $\mathcal{G}_s$.

Counting Short-Short Intersections. The Short-short intersections can be counted in $O(m^2)$ time by testing all pairs of segments of $\mathcal{G}_s$.

Counting Long-Long Intersections. Let $\bar{\mathcal{G}}_1$ be the set of lines containing the segments of $\mathcal{G}_1$. Since the segments in $\mathcal{G}_1$ do not have their endpoints inside Δ, the number of long-long intersections is the same as the number of intersections of $\bar{\mathcal{G}}_1$ lying in the interior of Δ. By Lemma 3.1 of the first part of this paper (see [A3]), the latter quantity can be computed in time $O(n \log n)$.

Counting Long-Short Intersections. For every segment $e \in \mathcal{G}_s$, let $\tilde{e}$ denote $e \cap \Delta$; and $\tilde{\mathcal{G}}_s = \{\tilde{e} | e \in \mathcal{G}_s\}$. Let $\mathcal{G}_1$ denote the set of lines containing the segments of $\mathcal{G}_1$. It clearly suffices to count, for each $\ell \in \bar{\mathcal{G}}_1$, the number of intersections between ℓ and $\tilde{\mathcal{G}}_s$.

Dualize each segment $\tilde{e} \in \tilde{\mathcal{G}}_s$ to a double wedge e^*, and construct the arrangement $\mathcal{H}$ of these double wedges. For any double wedge e^*, each face f of $\mathcal{H}$ is either contained in e^* or does not intersect e^*. The *weight* of a face f is the number of double wedges containing f; the weights of all faces of $\mathcal{H}$ can be determined while constructing $\mathcal{H}$.

A line $\ell \in \bar{\mathcal{G}}_1$ intersects a segment $\tilde{e} \in \tilde{\mathcal{G}}_s$ if and only if the point ℓ^* lies in the double wedge e^*. Thus, for every segment e in $\mathcal{G}_1$, the number of segments in $\tilde{\mathcal{G}}_s$ intersecting e is equal to the weight of the face in $\mathcal{H}$ containing the point ℓ^*. Therefore, we determine the number of segments intersecting ℓ by locating ℓ^* in $\mathcal{H}$. Computing $\mathcal{H}$ and preprocessing it for fast point-location queries can be done in time $O(m^2)$ [EOS], [EGSt], so all long-short intersections can be computed in time $O(m^2 + n \log n)$.

The above discussion implies that all intersection points of $\mathcal{G}$ contained in Δ can be counted in $O(m^2 + n \log n)$ time. The time complexity of the above procedure can be improved to $O(m\sqrt{n \log n} + n \log n)$ by partitioning $\mathcal{G}_s$ into $\lceil m/\sqrt{n \log n} \rceil$

subsets of size at most $\sqrt{n \log n}$ each, and counting the number of intersection points between each of the subsets and $\mathscr{G}_1$.

Next we describe the main algorithm. Partition the plane into $M = O(r^2)$ triangles $\Delta_1, \ldots, \Delta_M$, each meeting at most $O(n/r)$ lines containing the segments of $\mathscr{G}$. Using the algorithm described above, we count the number of intersections contained in each Δ_i, for $i \leq M$, and add up the results. If m_i denotes the number of endpoints lying inside Δ_i, the time spent in counting intersections within Δ_i is $O(m_i\sqrt{n/r} \log^{1/2} n + (n/r) \log n)$. Using the same analysis as in previous sections, the total time of the algorithm is

$$\sum_{i=1}^{M} O\left(m_i \sqrt{\frac{n}{r}} \cdot \log^{1/2} n\right) + O(nr \log n \log^{\omega} r)$$

$$= O\left(\frac{n^{3/2}}{r^{1/2}} \log^{1/2} n + nr \log n \log^{\omega} r\right)$$

because $\sum_{i=1}^{M} m_i \leq 2n$. Hence, by choosing $r = \lceil n^{1/3}/\log^{(2\omega+1)/3} n \rceil$, we obtain

Theorem 5.1. *Given a set of n line segments, their intersection points can be counted in time $O(n^{4/3} \log^{(\omega+2)/3} n)$ and $O(n^{4/3}/\log^{(2\omega+1)/3} n)$ space.*

Remark 5.2. We can combine this algorithm with the algorithm of [CE] that computes the number of intersections k in time $O(n \log n + k)$. That is, we first run the algorithm of [CE] and stop it as soon as the number of intersections exceeds $n^{4/3} \log^{(\omega+2)/3} n$. Then we use our algorithm. We thus have

Corollary 5.3. *The number of k intersections between n line segments can be counted in time $O(\min\{n \log n + k, n^{4/3} \log^{(\omega+2)/3} n\})$ and space $O(\min\{n + k, n^{4/3}/\log^{(2\omega+1)/3} n\})$.*

6. Counting and Reporting Red-Blue Intersections

Next, we consider a variant of the segment intersection problem:

> Given a set Γ_r of n_r "red" line segments and another set Γ_b of n_b "blue" line segments, count or report all intersections between Γ_r and Γ_b.

Let $n = n_r + n_b$. Mairson and Stolfi [MS] gave an $O(n \log n + K)$ algorithm to report all K red-blue intersections, when red-red and blue-blue intersections are not present. The algorithm of Chazelle and Edelsbrunner [CE] for reporting segment intersections can also be applied to report all red-blue intersections in this special case. However, in the general case these algorithms cannot avoid encountering red-red and blue-blue intersections. For the general case, Agarwal and Sharir [AS] presented an $O((n_r\sqrt{n_b} + n_b\sqrt{n_r} + K) \log n)$ algorithm to report all K

red-blue intersections. They showed that a restricted version of this problem, in which it is only required to detect a red-blue intersection, can be solved in $O(n^{4/3+\delta})$ (randomized expected) time, for any $\delta > 0$, by reducing it to the problem of computing at most $2n$ faces in $\mathscr{A}(\Gamma_r)$ and in $\mathscr{A}(\Gamma_b)$. As for the counting problem, in the absence of monochromatic intersections, Chazelle *et al.* [CEGS1] have developed an $O(n \log n)$ algorithm to count all red-blue intersections (see also [CEGS2]). In this section we present an $O(n^{4/3} \log^{(\omega+2)/3} n)$ algorithm to count all red-blue intersections in the general case, using roughly $n^{4/3}$ space. Our algorithm actually computes, for every red segment e, the number of blue segments intersecting e. The algorithm can be modified to report all K red-blue intersections in time $O(n^{4/3} \log^{(\omega+2)/3} n + K)$.

As in the previous section, we first consider a restricted version of the problem. Let Γ_r and Γ_b be two sets of segments as defined above, all meeting the interior of a triangle Δ, such that m of these segments contain at least one endpoint inside Δ; we wish to count the number of red-blue intersections lying inside Δ. We describe an $O((m^2 + n) \log n)$ algorithm that, for every red segment e, counts the number of blue segments intersecting e, and can be modified to report all red-blue intersections with $O(1)$ overhead per intersection. The algorithm proceeds as follows:

Partition the segments of Γ_r and Γ_b into four subsets:

(i) A: "long" segments in Γ_r whose endpoints do not lie inside Δ; let $|A| = a$.
(ii) B: "short" segments in Γ_r having at least one endpoint inside Δ; let $|B| = b$.
(iii) C: "long" segments in Γ_b whose endpoints do not lie inside Δ; let $|C| = c$.
(iv) D: "short" segments in Γ_b having at least one endpoint inside Δ; let $|D| = d$.

Note that $a + c = n - m$ and $b + d = m$. We have to count (or report) four types of red-blue intersections

- Intersections between A and C,
- intersections between A and D,
- intersections between B and C, and
- intersections between B and D.

Our approach is similar to the one used by Guibas *et al.* [GOS2] for counting segment intersections, as described in the previous section.

Intersections Between A and C. For a segment $e \in A \cup C$, its intersection points with $\partial\Delta$ are called the *endpoints* of e. Let S denote the set of endpoints of segments in $A \cup C$ sorted along $\partial\Delta$ in clockwise direction, starting from one of its vertices v. Let $a, b \in S$ be the endpoints of a red segment e with a appearing before b in S. Similarly, let a', b' be the endpoints of a blue segment e'. It is easily seen that e intersects e' if a, b, a', and b' appear in one of the following two orders:

(i) a, a', b, and b' (see Fig. 8(a)), or
(ii) a', a, b', and b (see Fig. 8(b)).

For each red segment e, we show how to count red-blue intersections along e. Scan the boundary of Δ in clockwise direction. When we encounter a blue segment for the first time, we insert it on top of a stack, maintained as a binary tree $\mathscr{B}$, and when

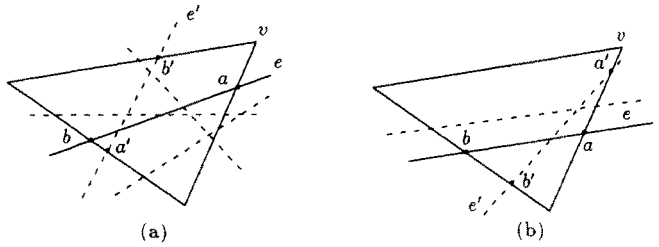

Fig. 8. Intersections between A and C.

it is encountered for the second time, we delete it from $\mathscr{B}$. On the other hand, when we encounter a red segment e for the first time, we do nothing, but when we encounter it for the second time, we count the number of (blue) segments in the stack that were inserted after encountering the first endpoint of e. This gives the number of type (i) intersections between e and C. Type (ii) red-blue intersections can be counted in a symmetric way by scanning $\partial\Delta$ in counterclockwise direction.

We leave it for the reader to verify that this algorithm can be easily modified to report all red-blue intersections between A and C.

For each segment $e \in A$, we spend $O(\log n)$ time, therefore the total time spent in counting (resp. reporting all K_{AC}) such red-blue intersections is $O((a + c)\log(a + c)) = O(n \log n)$ (resp. $O(n \log n + K_{AC})$).

Intersections Between A and D. For every $e \in D$, let $\tilde{e}$ denote $e \cap \Delta$; and let $\tilde{D} = \{\tilde{e} | e \in D\}$. Let $\bar{A}$ denote the set of lines containing the segments of A, and let A^* denote the set of points dual to the lines of $\bar{A}$. Let $\mathscr{L}$ denote the set of lines dual to the endpoints of the segments in $\tilde{D}$. Construct the arrangement $\mathscr{A}(\mathscr{L})$. For each $\ell \in \bar{A}$, we count the number of intersections between ℓ and $\tilde{D}$ by locating the point ℓ^* in $\mathscr{A}(\mathscr{L})$, as in Section 5. The total time spent in counting these intersections is easily seen to be $O(m^2 + n \log n)$.

As for reporting the intersections between A and D contained in Δ, let $\psi(f)$ denote the set of double wedges dual to the segments of $\tilde{D}$ containing a face f of $\mathscr{A}(\mathscr{L})$. If two faces f_1 and f_2 share an edge γ, contained in a line $\ell \in \mathscr{L}$, then $\delta_\gamma = \psi(f_1) \oplus \psi(f_2)$ is the set of segments having dual of ℓ as an endpoint.[1] Therefore by first constructing the arrangement $\mathscr{A}(\mathscr{L})$, and then locating each point of A^* in $\mathscr{A}(\mathscr{L})$, we can report all K_{AD} intersections between A and D contained in Δ, in time $(m^2 + n \log n + \sum_\gamma |\delta_\gamma| + K_{AD})$. Thus, it suffices to bound $\sum_{e \in \mathscr{A}(\mathscr{L})} |\delta_e|$. Suppose the segments of $\tilde{D}$ have $t \leq 2d$ distinct endpoints and v_i segments are incident to the ith endpoint. Obviously, $\sum_{i=1}^{t} v_i = 2d$ and, for each line $\ell \in \mathscr{L}$, there are t edges of $\mathscr{A}(\mathscr{L})$ contained in ℓ, therefore

$$\sum_{e \in \mathscr{A}(\mathscr{L})} |\delta_e| = \sum_{i=1}^{t} t v_i \leq 2d^2. \tag{6.1}$$

Hence, the total time spent is $O(n \log n + m^2 + K_{AD})$.

[1] We use $A \oplus B$ to denote the symmetric difference of sets A and B.

Intersections Between B and C. If we just want to report or count the total number of intersections between B and C contained in Δ, we can use the same procedure as in the previous case. But if we want to count the number of red-blue intersections for each red segment separately, we need a different technique.

Let $\tilde{B} = \{e \cap \Delta | e \in B\}$, and let B^* denote the set of double wedges dual to the segments in $\tilde{B}$. Let $\bar{C}$ denote the set of lines containing the segments of C, and let C^* denote the set of points dual to the lines in $\bar{C}$. The number of intersections between a segment $e \in \tilde{B}$ and C is equal to the number of points of C^* in the double wedge e^*. Therefore, for every double wedge, we want to find the number of points of C^* lying in it. This can be done in time $O(b^2 + c \log(b + c)) = O(m^2 + n \log n)$, using the algorithm described in Edelsbrunner *et al.* [EGH*].

Intersections Between B and D. For every segment $e \in B$, we can determine the segments of D intersecting it by testing all such pairs of segments. This takes $O(m^2)$ time.

Thus, for every segment in Γ_r, we can count the number of blue segments intersecting it inside Δ in time $O(m^2 + n \log n)$, and we can report all red-blue intersections inside Δ within the same time plus $O(1)$ overhead per intersection. The running time can be improved to $O(m\sqrt{n} \log^{1/2} n + n \log n)$ by partitioning the collection of short segments (that is $B \cup D$) into $\lceil m/\sqrt{n \log n} \rceil$ subsets of size $\sqrt{n \log n}$ each, and then repeating the above procedure for each subset and the entire $A \cup C$.

Going back to the original problem, we partition the plane into $O(r^2)$ triangles, each meeting at most n/r lines containing the segments of $\Gamma_r \cup \Gamma_b$. Using the algorithm described above, count (resp. or more generally report all K_i) red-blue intersections within the *ith* triangle in $O(m_i\sqrt{n/r} \log^{1/2} (n/r) + (n/r) \log (n/r))$ (resp. $O(m_i\sqrt{n/r} \log^{1/2} (n/r) + (n/r) \log (n/r) + K_i)$) time, where m_i is the number of segment endpoints falling inside the *ith* triangle. Following the same analysis as in Section 5, we obtain

Theorem 6.1. *Given a set of n_r "red" line segments and another set of n_b "blue" line segments, we can count, for each red segment, the number of blue segments intersecting it in overall time $O(n^{4/3} \log^{(\omega+2)/3} n)$ using $O(n^{4/3}/\log^{(2\omega+1)/3})$ space, where $n = n_r + n_b$. Moreover, we can report all K red-blue intersections in time $O(n^{4/3} \log^{(\omega+2)/3} n + K)$.*

Remark 6.2. (i) Our algorithm uses roughly $n^{4/3}$ space only for partitioning the plane into $O(r^2)$ triangles; all other stages of the algorithm require $O(n)$ space. If we choose $r = O(1)$ and solve the problem recursively as in [GOS2], we can reduce the space complexity to $O(n)$, but the running time increases to $O(n^{4/3+\delta})$, for any $\delta > 0$ (which can be made as small as we wish by choosing r sufficiently large).

(ii) If we allow randomization, then using the random-sampling technique of [Cl1] or of [HW], we can count all red-blue intersections in $O(n^{4/3} \log n)$ expected time, and can report all K red-blue intersections in expected time $O(n^{4/3} \log n + K)$. We leave it for the reader to fill in the missing details.

(iii) Note that if Γ_r is a set of lines, then we have to consider only the first two cases, because $B = \emptyset$.

7. Batched Implicit Point Location

The planar point-location problem is a well-studied problem in computational geometry [K], [EGSt], [STa]. In this problem it is required to preprocess a given planar subdivision so that, for a query point, the face containing p can be computed quickly. Guibas *et al.* [GOS1] have considered a generalization of this problem, in which the map is defined as the arrangement (i.e., overlay) of n polygonal objects of some simple shape, and we want to compute certain information for the query points related to their arrangement (for example, to determine which query points lie in the union of these polygons). For simplicity we break the given polygonal objects into a collection of line segments, and consider the following formal statement of the problem:

> We are given a collection $\mathscr{G} = \{e_1, \ldots, e_n\}$ of n segments, and with each segment e we associate a function φ_e defined on the entire plane, which assumes values in some associative and commutative semigroup S (denote its operation by $+$), and let $\Phi(x) = \sum_{e \in \mathscr{G}} \varphi_e(x)$. Given a set $P = \{p_1, \ldots, p_m\}$ of m points, compute $\Phi(p_1), \ldots, \Phi(p_m)$ efficiently.

We assume that φ_e and Φ satisfy the following conditions:

(i) The function φ_e has constant complexity, that is, we can partition the plane into $O(1)$ convex regions so that within each region φ_e is constant. This also implies that, for any given point x, $\varphi_e(x)$ can be computed in $O(1)$ time.
(ii) Any two values in S can be added in $O(1)$ time.
(iii) Given a set $\mathscr{G}$ of n segments in the plane, we can preprocess $\mathscr{G}$ in time $O(n \log^k n)$, for some $k \geq 0$, into a linear-size data structure so that, for a query point x lying either above all the lines containing the segments of $\mathscr{G}$, or below all these lines, $\Phi(x)$ can be calculated in $O(\log n)$ time.

We will see that several natural problems, including the containment problem mentioned above, can be cast into this abstract framework. Note that we consider here the *batched* version of the problem, in which all query points are known in advance. In another paper [A3] we consider the preprocessing-and-query version of the problem and solve it using different techniques based on spanning trees with low stabbing number.

A naive approach to solving this problem is to construct the arrangment $\mathscr{A}(\mathscr{G})$ (more precisely, the arrangement obtained by overlapping all the convex subdivisions associated with each of the functions φ_e), so that the value of Φ is constant within each resulting face. Now $\Phi(p_1), \ldots, \Phi(p_m)$ can be easily computed in $O(m \log n)$ time by locating the points of P in the above planar map. If $m \geq n^2$, then this is the method of choice, and it runs in overall $O(m \log n)$ time, but if $m < n^2$ this

procedure takes $\Omega(n^2)$ time in the worst case, so the goal is to come up with a subquadratic algorithm. Guibas *et al.* [GOS1] have indeed given a randomized algorithm whose expected running time is $O(m^{2/3-\delta}n^{2/3+\delta} + m \log n + n \log^{k+1} n)$, for any $\delta > 0$. Our (deterministic) algorithm improves their result and works as follows.

Let $\mathscr{L}$ denote the set of lines containing the segments of $\mathscr{G}$. Let $\mathscr{L}^*$ (resp. P^*) denote the set of points (resp. lines) dual to the lines (resp. points) of $\mathscr{L}$ (resp. P). Partition the dual plane, in time $O(mr \log m \log^{\omega} r)$, into $t = O(r^2)$ triangles $\Delta_1', \ldots, \Delta_t'$, each meeting $O(m/r)$ lines of P^*. If a triangle contains $n_i > n/r^2$ points of $\mathscr{L}^*$, then partition it further, in time $O(n \log n)$, into $\lceil n_i r^2/n \rceil$ triangles, none of which contains more than n/r^2 points. Let $\Delta_1, \ldots, \Delta_M$ denote the resulting triangles; we have $M = O(r^2)$. Let P_i^* denote the set of lines passing through Δ_i, and let $\mathscr{L}_i^*$ denote the set of points contained in Δ_i; thus $|P_i^*| = O(m/r)$, $|\mathscr{L}_i^*| \leq n/r^2$. Let $\Phi_i = \sum_{e \in \mathscr{G}_i} \varphi_e$. For each $p \in P_i$, compute $\Phi_i(p)$ by constructing the entire arrangement $\mathscr{A}(\mathscr{G}_i)$, as discussed above (see also [GOS1]). The total time spent in computing $\Phi_i(p)$ for all $p \in P_i$ is $O(n^2/r^4 + (m/r) \log n)$ [EOS], [EGSt].

Next, we show how to add the values computed within each triangle to calculate $\Phi(p) = \sum_i \Phi_i(p)$. We use a procedure similar to the one used in Section 4 for computing many faces in an arrangement of segments. In particular we construct a binary tree $\mathscr{T}$ with the properties defined in Section 4. For each node v of $\mathscr{T}$, let $\mathscr{G}_v$, P_v be as defined in Section 4, let $m_v = |P_v|$, $n_v = |\mathscr{G}_v|$, and $\Phi_v = \sum_{e \in \mathscr{G}_v} \varphi_e$. At each node v of $\mathscr{T}$, the goal is to compute Φ_v for all $p \in P_v$. At the end of this process we will have obtained, at the root u of $\mathscr{T}$, the value of $\Phi_u = \Phi$ for all $p \in P_u = P$. We calculate Φ_v in a bottom-up fashion, starting at the leaves of $\mathscr{T}$, as described above.

Let v be an internal node of $\mathscr{T}$ having children w and z. We preprocess $\mathscr{G}_w$, $\mathscr{G}_z$ to obtain data-structures $\mathscr{D}_w$, $\mathscr{D}_z$ of linear size so that, for any point lying either above all the lines containing the segments of $\mathscr{G}_w$ (resp. $\mathscr{G}_z$), or below all these lines, Φ_w (resp. Φ_z) can be computed in logarithmic time. Now, for each $p \in P_v$, $\Phi_v(p) = \Phi_w(p) + \Phi_z(p)$. If $p \in P_w$, we already have computed $\Phi_w(p)$ at w. Otherwise p lies either above all lines containing the segments of $\mathscr{G}_w$, or below all these lines, so we can use $\mathscr{D}_w$ to compute $\Phi_w(p)$ in $O(\log n_w)$ time. Similar actions are taken to compute $\Phi_z(p)$. Thus we can obtain Φ_v for all points in P_v in time $O(m_v \log n_v)$. By the third property of φ_e, $\mathscr{D}_w$ can be constructed in $O(n_w \log^k n_w)$ time, and similarly for $\mathscr{D}_z$. Hence, the total time spent in computing Φ_v over all nodes v of $\mathscr{T}$, including the initial partitioning of the dual plane, is

$$
\begin{aligned}
T(m, n) &= \sum_{v \in \mathscr{T}} O(m_v \log n_v + n_v \log^k n_v) + O(mr \log m \log^{\omega} r) \\
&\quad + O\left(r^2\left(\frac{n^2}{r^4} + \frac{m}{r} \log n\right)\right) \\
&= \sum_{i=1}^{H} \sum_{h(v)=i} O(m_v \log n_v + n_v \log^k n_v) + O\left(\frac{n^2}{r^2} + mr \log n \log^{\omega} r\right),
\end{aligned}
$$

where $H = O(\log r)$ is the height of $\mathscr{T}$ and $h(v)$ is the height of a node v of $\mathscr{T}$. As

mentioned in Section 4, it was shown in [AEGS] that

$$m_v \le \frac{ck_v m}{r} + 1,$$

where c is some constant > 0 and k_v is the number of leaves in the subtree of $\mathcal{T}$ rooted at v. Moreover, we argued in Section 4 that

$$\sum_{h(v)=i} k_v = O(r^2) \quad \text{and} \quad \sum_{h(v)=i} n_v = n.$$

Therefore

$$T(m,n) = \sum_{i=1}^{H} O\left(r^2 \cdot \frac{m}{r} \log n + n \log^k n\right) + O\left(\frac{n^2}{r^2} + mr \log n \log^\omega r\right)$$
$$= O\left(mr \log n \log^\omega r + n \log^k n \log r + \frac{n^2}{r^2}\right).$$

By choosing

$$r = \max\left\{\frac{n^{2/3}}{m^{1/3} \log^{1/3} m \log^{\omega/3} (n/\sqrt{m})}, 2\right\},$$

we obtain

Theorem 7.1. *Given a collection $\mathcal{G}$ of n segments, a function φ_e associated with each $e \in \mathcal{G}$ with the properties listed above, and a set P of m points, we can compute $\sum_{e \in \mathcal{G}} \varphi_e(p)$, for each $p \in P$, in time*

$$O\left(m^{2/3} n^{2/3} \log^{2/3} m \log^{2\omega/3} \frac{n}{\sqrt{m}} + n \log^k n \log \frac{n}{\sqrt{m}} + m \log n\right).$$

Remark 7.2. (i) In several special cases it is possible to obtain $\mathcal{D}_v$, in $O(n_v)$ time, by merging $\mathcal{D}_w$ and $\mathcal{D}_z$. In such cases the second term of the above bound reduces to $O(n \log^k n)$.

(ii) As mentioned above, we have recently obtained, in [A1], an algorithm that preprocesses $\mathcal{G}$, in time $O(n^{3/2} \log^{\omega+1} n)$, into a data structure of size $O(n \log^2 n)$ so that, given a point p, $\Phi(p)$ can be computed in $O(\sqrt{n} \log^2 n)$ time. (The query time can be reduced to $O(\sqrt{n} \log n)$ in several special cases.)

(iii) As in Section 4 the running time can be improved to

$$O\left(m^{2/3} n^{2/3} \log^{2/3} n \log^{\omega/3} \frac{n}{\sqrt{m}} + n \log^k n \log \frac{n}{\sqrt{m}} + m \log n\right)$$

by partitioning Δ_i' into $\lceil n_i/(\sqrt{m/r}\log^{1/2} n)\rceil$ triangles none of which contains more than $\sqrt{m/r}\log^{1/2} r$ points of $\mathscr{L}^*$, and by choosing

$$r = \max\left\{\frac{n^{2/3}}{m^{1/3}\log^{1/3} n \log^{2/3}(n/\sqrt{m})}, 2\right\}.$$

Various applications of the batched implicit point-location problem have been discussed in [GOS1]. The running time of all applications can be improved by using the algorithm provided in Theorem 7.1. We briefly describe a couple of these applications, and refer to [GOS1] for more details.

7.1. *Polygon Containment Problem—Batched Version*

Consider the following problem:

> Given a set **T** of n (possibly intersecting) triangles and a set P of m points, for every point p of P, count the number of triangles in **T** containing p, or more generally, for each point p, report all triangles containing p.

We review the solution technique of [GOS1]. Let $\mathscr{G}$ be the set of the edges of triangles in **T**, and let $\mathscr{L}$ be the set of lines containing the segments of $\mathscr{G}$. For each edge e of a triangle Δ, let $B(e)$ denote the semi-infinite trapezoidal strip lying directly below e. Define a function φ_e in the plane so that $\varphi_e(p) = 0$ if p lies outside $B(e)$, $\varphi_e(p) = 1$ if p is in $B(e)$ and Δ lies below the line containing e, otherwise $\varphi_e(p) = -1$.

It is easily seen that, for any point p, $\Phi(p)$ gives the number of triangles containing p, and φ_e satisfies properties (i) and (ii). As for property (iii), if a point p lies above all lines of $\mathscr{L}$, then $\Phi(p) = 0$, by definition. If p lies below all lines of $\mathscr{L}$, then we do the following. Let $\hat{e}$ denote the x-projection of an edge e of a triangle. It is easily checked that

$$\Phi(p) = \sum_{\{j: p \in \hat{e}_j\}} \varepsilon_j,$$

where ε_j is the nonzero value of φ_{e_j} at p. Note that the sum of the right-hand side does not change between two consecutive endpoints of the projected segments, and that the value of Φ over each interval can be computed in overall $O(n \log n)$ time, by scanning these projected segments from left to right. Hence, we can preprocess **T**, in time $O(n \log n)$, into a data structure $\mathscr{D}$ so that, for a point p lying below all lines of $\mathscr{L}$, $\Phi(p)$ can be computed in $O(\log n)$ time. Moreover, for a node v in $\mathscr{T}$, $\mathscr{D}_v$ can be obtained in $O(n_v)$ time by merging $\mathscr{D}_w$ and $\mathscr{D}_z$, where w, z are the children of v. Hence, Theorem 7.1 and the remark following it imply that

Corollary 7.3. *Given a set* $\mathbf{T}$ *of* n *triangles and a set* P *of* m *points, we can compute, for each point* $p \in P$, *the number of triangles in* $\mathbf{T}$ *containing* p *in time*

$$O\left(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}\frac{n}{\sqrt{m}} + (m+n)\log n\right).$$

7.2. *Implicit Hidden Surface Removal—Batched Version*

The next application of the implicit point-location problem is the following version of the hidden surface removal problem:

> Given a collection of objects in three-dimensional space, and a viewing point a, we wish to calculate the scene obtained by viewing these objects from a.

The hidden surface removal problem has been extensively studied by many researchers (see, e.g., [D] and [Mc]), because of its applications in graphics and other areas. For the sake of simplicity let us restrict our attention to polyhedral objects, whose boundary $\mathbf{T}$ is a collection $\{\Delta_1, \ldots, \Delta_n\}$ of n nonintersecting triangles. In the case of the *implicit* hidden surface removal problem, we do not want to compute the scene explicitly; instead we wish to determine the objects seen at given pixels [CS], [GOS1]. In this subsection we consider the following special case of the implicit hidden surface removal problem. Let $\mathbf{T} = \{\Delta_1, \ldots, \Delta_n\}$ be a collection of n nonintersecting horizontal triangles in $\mathbb{R}^3$ such that Δ_i lies in the plane $z = c_i$, where $c_1 \leq c_2 \leq \cdots \leq c_n$ are some fixed heights. Let $P = \{p_1, \ldots, p_m\}$ be a set of m points lying in a horizontal plane below all triangles of $\mathbf{T}$. The problem is to determine, for each point $p \in P$, the lowest triangle Δ_i hit by the vertical line passing through p.

We review the techniques used by Guibas *et al.* [GOS1]. A point $p \in P$ is said to be *blocked* by $\mathbf{T}$, if the vertical line from p intersects at least one triangle $\Delta_i \in \mathbf{T}$. First consider the following problem: Given a set $\mathbf{T}$ of n triangles and a set P of m points, determine which points of P are blocked by Δ. This problem can be solved by applying our implicit point-location algorithm to P and the xy-projection of the triangles in $\mathbf{T}$. Hence, we can compute the blocked points in $O(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}(n/\sqrt{m}) + (m+n)\log n)$ time.

Going back to the original problem, if the number of the points or the number of the triangles is ≤ 1, then we sovle the problem directly; otherwise we split $\mathbf{T}$ into two subsets $\mathbf{T}_1, \mathbf{T}_2$, so that $\mathbf{T}_1$ contains the lower half of the triangles $\Delta_1, \ldots, \Delta_{n/2}$ and $\mathbf{T}_2$ contains the upper half of the triangles $\Delta_{n/2+1}, \ldots, \Delta_n$. Apply the blocking algorithm to P and $\mathbf{T}_1$. Let $P_1 \subset P$ be the subset of points blocked by $\mathbf{T}_1$, and let $P_2 = P - P_1$. We recursively compute the lowest triangle in $\mathbf{T}_1$ (resp. in $\mathbf{T}_2$) above each of the points in P_1 (resp. P_2). Using the same analysis as in [GOS1], we can show that the total running time is

$$O\left(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}\frac{n}{\sqrt{m}} + m\log n + n\log^2 n\right).$$

Hence, we can conclude

Theorem 7.4. *Given an ordered collection* $\mathbf{T}$ *of* n *triangles in* $\mathbb{R}^3$ *and a set* P *of* m *points lying below all of them, we can determine, in* $O(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}(n/\sqrt{m}) + m\log n + n\log^2 n)$ *time, the triangle seen from each point of* P *in the upward vertical direction.*

Remark 7.5. (i) In fact this algorithm works for a more general case, where triangles in $\mathbf{T}$ have the property that they can be linearly ordered so that if a vertical line hits two triangles Δ_i and Δ_j with Δ_i lying below Δ_j, then $\Delta_i < \Delta_j$.

(ii) We can extend the above algorithm to the case where the points of P do not lie below all of the triangles in $\mathbf{T}$. Now at each level of recursion, for each point p of P_1, we also find the highest triangle Δ_p of $\mathbf{T}_1$ whose projection contains p. If Δ_p lies below p, then we remove p from P_1 and add it to P_2. Using the above algorithm we can find Δ_p, for each $p \in P_1$, in time

$$O\left(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}\frac{n}{\sqrt{m}} + m\log n + n\log^2 n\right).$$

Therefore the overall running time is

$$O\left(m^{2/3}n^{2/3}\log^{2/3} m \log^{2\omega/3}\frac{n}{\sqrt{m}} + m\log n + n\log^3 n\right).$$

8. Approximate Half-Plane Range Searching

The half-plane range-query problem is defined as: Given a set S of n points in the plane, preprocess it so that for any query line ℓ, we can quickly count the number of points in S lying above ℓ. In the dual setting, S becomes a set S^* of n lines, ℓ becomes a point ℓ^*, and the number of points lying above ℓ is the same as the level of ℓ^* in $\mathscr{A}(\mathscr{S}^*)$. Therefore, if we allow $O(n^2)$ space, the query can be obviously answered in time $O(\log n)$ by precomputing $\mathscr{A}(\mathscr{S}^*)$ and locating ℓ^* in it. Chazelle and Welzl [CW] recently gave an algorithm that answers a query in time $O(\sqrt{n}\log n)$ using only $O(n)$ space. A result of Chazelle [Ch2] shows that if we restrict the space to be linear, the query takes at least $\Omega(\sqrt{n})$ time in the semigroup model (in particular subtraction is not allowed, see [Ch2] for details), which implies that we cannot hope for a much better algorithm if we want to count the exact number of points. However, in several applications it suffices to count the number of points approximately (one such example is described in [Ma1]). Therefore, in the dual setting, the approximate half-plane range-query problem is: Given a set S^* of n lines and a parameter (not necessarily a constant) $\varepsilon > 0$, preprocess it so that, for any query point, we can quickly compute an approximate level for it in $\mathscr{A}(\mathscr{S}^*)$, namely a level that lies within $\pm\,\varepsilon n$ from the true level. It is

easily seen that the problem can be reduced to an instance of a point-location problem in an $(\varepsilon n/4)$-approximate leveling of $\mathscr{A}(\mathscr{S}^*)$ (see also [EW1] and [Ma2]). Hence by Corollary 6.6 of the first part of this paper [A3], we obtain

Theorem 8.1. *Given a set of n points in the plane and a positive real number $\varepsilon < 1$, we can preprocess it, in time $O((n/\varepsilon)\log n \log^{\omega}(1/\varepsilon))$, into a data structure of size $O(1/\varepsilon^2)$, so that, for any query line ℓ, we can obtain, in $O(\log n)$ time, an approximate count of the number of points in S lying above ℓ, which deviates from the true number by at most $\pm \varepsilon n$.*

9. Computing Spanning Trees with Low Stabbing Number

Let S be a set of n points in $\mathbb{R}^d$ and let $\mathscr{T}$ be a spanning tree on S whose edges are line segments. The *stabbing number* $\sigma(\mathscr{T})$ of $\mathscr{T}$ is the maximum number of edges of $\mathscr{T}$ that can be crossed by a hyperplane h. Chazelle and Welzl [CW] (see also [W]) have proved that, for any set of n points in $\mathbb{R}^d$, there exists a spanning tree with stabbing number $O(n^{1-1/d})$, and that this bound is tight in the worst case. For a family $\mathbf{T}$ of trees, the stabbing number $\sigma(\mathbf{T})$ is s if for every hyperplane h there is a tree $\mathscr{T} \in \mathbf{T}$ such that h intersects at most s edges of $\mathscr{T}$.

Edelsbrunner *et al.* [EGH*] gave a randomized algorithm with expected running time $O(n^{3/2}\log^2 n)$ to compute a family $\mathbf{T} = \{\mathscr{T}_1, \ldots, \mathscr{T}_k\}$ of $k = O(\log n)$ spanning trees with the property that, for any line ℓ, there exists at least one tree $\mathscr{T}_i$ such that ℓ intersects $O(\sqrt{n}\log^2 n)$ edges of $\mathscr{T}_i$. They also showed that a spanning tree on S with stabbing number $O(\sqrt{n})$ can be deterministically constructed in time $O(n^3 \log n)$. Recently Matoušek [Ma1] has improved the running time of these algorithms. He has given a randomized algorithm with expected running time $O(n^{4/3}\log^2 n)$ to construct a family of $O(\log n)$ spanning trees with the above property; this algorithm can be converted into a deterministic one with $O(n^{7/4}\log^2 n)$ running time. He has also given an $O(n^{5/2}\log n)$ deterministic algorithm (or a randomized algorithm with expected running time $O(n^{1.4+\delta})$, for any $\delta > 0$) to construct a single spanning tree with stabbing number $O(\sqrt{n})$. His algorithms actually compute spanning paths of S.

In this section we describe a deterministic algorithm for constructing a family $\mathbf{T}$ of $O(\log n)$ spanning trees with $\sigma(\mathbf{T}) = O(\sqrt{n})$. The crux of Matoušek's algorithms lies in the following lemma.

Lemma 9.1 [Ma1]. *Given a set S of n points in the plane, we can find a set $\mathscr{L}$ of $O(n)$ lines with the property that, for any spanning path $\mathscr{T}$ on S and for every line ℓ, there is a line $\ell' \in \mathscr{L}$ such that if ℓ' intersects s edges of $\mathscr{T}$, then ℓ intersects at most $s + O(\sqrt{n}\log n)$ edges of $\mathscr{T}$.*

Matoušek describes an $O(n^{7/4}\log^2 n)$ deterministic algorithm to compute this set of lines. Using Theorem 1.1 we can strengthen Lemma 9.1 as follows:

Lemma 9.2. *Given a set S of n points in the plane, we can deterministically construct a set $\mathscr{L}$ of $O(n)$ lines in time $O(n^{3/2}\log^{\omega+1} n)$ with the property that, for any spanning path $\mathscr{T}$ on S and for every line ℓ, there is a line $\ell' \in \mathscr{L}$ such that if ℓ' intersects s edges of $\mathscr{T}$, then ℓ intersects at most $s + O(\sqrt{n})$ edges of $\mathscr{T}$.*

Proof. Dualize the points of S; we obtain a set S^* of n lines. By Theorem 1.1, choosing $r = \sqrt{n}$, we can partition the plane into $O(n)$ triangles in time $O(n^{3/2}\log^{\omega+1} n)$, so that no triangle meets more than $O(\sqrt{n})$ lines of S. Pick up a point ℓ^* from each triangle, let $\mathscr{L}^*$ denote the set of these points, and let $\mathscr{L}$ be the set of their dual lines in the primal plane.

Arguing as in [Ma1], let ζ be an arbitrary line in the primal plane. By construction, there exists a line $\ell \in \mathscr{L}$ such that the segment $e = \overline{\zeta^*\ell^*}$ does not cross more than $O(\sqrt{n})$ lines of S^*. Going back to the primal plane, if an edge g of $\mathscr{T}$ intersects ζ but not ℓ, then one endpoint of g must lie in the double wedge e^* dual to e, but our construction implies that e^* contains at most $O(\sqrt{n})$ points of S. Thus, there are $O(\sqrt{n})$ edges of $\mathscr{T}$ that intersect ζ but not ℓ, and the lemma follows. □

We construct a family of $O(\log n)$ spanning paths with low stabbing number only for the lines in $\mathscr{L}$. Although the basic approach is the same as in [Ma1] or [EGH*], we need some additional techniques to improve the running time. Here we briefly sketch the main idea, and refer the reader to [Ma1] or [EGH*] for more details.

Suppose we have constructed $\mathscr{T}_1, \ldots, \mathscr{T}_{i-1}$, and have obtained a set $\mathscr{L}_i \subset \mathscr{L}$ such that $m_i = |\mathscr{L}_i| \le m/2^{i-1}$ (where $m = |\mathscr{L}| = O(n)$) and $\mathscr{L}_i$ is "bad" for all paths constructed so far, that is, a line in $\mathscr{L}_i$ intersects every tree at more than $C\sqrt{n}$ edges, for some constant C to be specified below. We show how to construct $\mathscr{T}_i$ and $\mathscr{L}_{i+1}$. Initially $\mathscr{L}_1 = \mathscr{L}$.

The spanning path $\mathscr{T}_i$ is constructed in $O(\log n)$ phases. In the beginning of the jth phase we have a current collection S_j of vertex-disjoint paths on S (in the beginning of the first phase the collection S_1 consists of all singleton paths on the points of S). Our algorithm ensures that $n_j = |S_j| \le n\cdot(\frac{2}{3})^{j-1}$. If $n_j \le \sqrt{n}$, we connect the endpoints of the paths in S_j to form a single spanning path on S, and we are done (see Fig. 9). Otherwise, if $n_j > \sqrt{n}$, we proceed as follows. Choose $r = c_1\sqrt{n_j}$ and partition the plane into $n_j/3$ triangles so that no triangle meets more

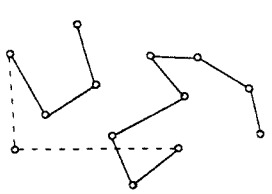

Fig. 9. Spanning path $\mathscr{T}_i$; connecting the endpoints of S_j.

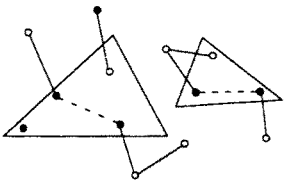

Fig. 10. Maximal matching of endpoints within Δ_i; solid circles denote the selected endpoints of S_j.

than $c_2(m_i/\sqrt{n_j})$ lines of $\mathscr{L}_i$, for appropriate constants c_1, c_2, which exist by Theorem 1.1. If a triangle contains endpoints of several paths in S_j, we obtain a maximal matching of these endpoints and connect each pair of matched points by an edge (see Fig. 10), thereby combining two paths in S_j into a new path. To avoid creating cycles, we only choose one endpoint of each path of S_j. The endpoints of the resulting paths form the set S_{j+1}. It can be easily proved that we add at least $n_j/3$ new edges to the current set of paths, which implies that $n_{j+1} \leq 2n_j/3$.

Lemma 9.3. *There are at least $m_i/2$ lines of $\mathscr{L}_i$ that intersect $\mathscr{T}_i$ in $\leq C\sqrt{n}$ edges, for some constant $C > 0$.*

Proof. We bound the total number of intersection points between edges of $\mathscr{T}_i$ and $\mathscr{L}_i$. In the jth phase we add at least $n_j - n_{j+1}$ edges, and each edge intersects at most $c_2(m_i/\sqrt{n_j})$ lines of $\mathscr{L}_i$. In the final phase we add at most $\sqrt{n}$ edges, each crossed by at most m_i lines. Therefore the total number of intersections I between $\mathscr{T}_i$ and $\mathscr{L}_i$ is at most

$$
\begin{aligned}
I &\leq \sum_{j=1}^{O(\log n)} (n_j - n_{j+1}) c_2 \frac{m_i}{\sqrt{n_j}} + \sqrt{n} \cdot m_i \\
&\leq c_2 m_i \sum_{j=1}^{O(\log n)} \sqrt{n_j} + \sqrt{n} \cdot m_i \\
&\leq c_2 m_i \sqrt{n} \sum_{j=1}^{O(\log n)} \left(\tfrac{2}{3}\right)^{(j-1)/2} + \sqrt{n} m_i \qquad \text{(because } n_j \leq n \cdot \left(\tfrac{2}{3}\right)^{j-1}\text{)} \\
&\leq \frac{c_2 m_i \sqrt{n}}{1 - \sqrt{\frac{2}{3}}} + \sqrt{n} \cdot m_i \\
&= ((3 + \sqrt{6}) c_2 + 1) \cdot m_i \sqrt{n}.
\end{aligned}
$$

Now it follows immediately that at least half of the lines in $\mathscr{L}_i$ intersect $\mathscr{T}_i$ in at most $C\sqrt{n}$ edges, for $C = (6 + 2\sqrt{6})c_2 + 2$. □

Lemma 9.3 implies that at most half of the lines are "bad." For every line $\ell \in \mathscr{L}_i$, we count the number of intersections between ℓ and $\mathscr{T}_i$, using our red-blue

intersection algorithm given in Section 6. We pick up those lines of $\mathscr{L}_i$ that intersect $\mathscr{T}_i$ at more than $C\sqrt{n}$ points. The resulting set is $\mathscr{L}_{i+1}$.

Next we analyze the running time of our algorithm. We first bound the time to compute $\mathscr{L}_i$ for $i \le k$. Since $m_i \le n$ and there are only n edges in $\mathscr{T}_i$, it follows from Theorem 6.1 that we can compute $\mathscr{L}_i$ in $O(n^{4/3}\log^{(\omega+2)/3} n)$ time. Moreover, $k = O(\log n)$, so the total time spent in computing the incidences between $\mathscr{T}_i$ and $\mathscr{L}_i$, over all k phases, is $O(n^{4/3}\log^{(\omega+5)/3} n)$.

As for the time spent in computing $\mathscr{T}_i$, we choose $r = c_1\sqrt{n_j}$ in the jth phase, therefore it requires $O(m_i\sqrt{n_j}\log m_i \log^{\omega} n_j + n_j \log n_j)$ time. (It is easily checked that this also bounds the time needed to distribute the path endpoints among the triangles, and to match them to obtain the new set of paths.) Hence the total time spent in computing $\mathscr{T}_i$ is at most

$$\begin{aligned}
&\sum_{j=1}^{O(\log n)} O(m_i\sqrt{n_j}\log m_i \log^{\omega} n_j + n_j \log n_j) \\
&\qquad = \sum_{j=1}^{O(\log n)} O(m_i\sqrt{n}\cdot(\tfrac{2}{3})^{(j-1)/2}\log m_i \log^{\omega} n + (\tfrac{2}{3})^{j-1} n \log n) \\
&\qquad = O(m_i\sqrt{n}\log m_i \log^{\omega} n + n\log n).
\end{aligned}$$

Summing over all i, we obtain

$$\begin{aligned}
&\sum_{i=1}^{k} O(m_i\sqrt{n}\log m_i \log^{\omega} n + n\log n) \\
&\qquad = \sum_{i=1}^{k} O(m_i\sqrt{n}\log^{\omega+1} n + n\log n) \\
&\qquad = O\left(\sum_{i=1}^{k} \frac{n}{2^{i-1}}\sqrt{n}\log^{\omega+1} n\right) \qquad \text{(because } m_i \le m/2^{i-1} \text{ and } m = O(n)\text{)} \\
&\qquad = O(n^{3/2}\log^{\omega+1} n).
\end{aligned}$$

Hence, we have

Theorem 9.4. *Given a set S of n points in the plane, we can deterministically construct, in time $O(n^{3/2}\cdot\log^{\omega+1} n)$, a family $\mathbf{T}$ of $k = O(\log n)$ spanning paths on S with the property that, for any line ℓ, there exists a path $\mathscr{T} \in \mathbf{T}$, such that ℓ intersects at most $O(\sqrt{n})$ edges of $\mathscr{T}$.*

Moreover, we have

Lemma 9.5. *The set of $O(\log n)$ spanning paths computed by the above algorithm have the property that, for any query line ℓ, we can determine, in $O(\log n)$ time, a path that ℓ intersects in at most $O(\sqrt{n})$ edges. This requires an additional linear preprocessing time and storage.*

Proof. Let Δ be the set of triangles computed in the proof of Lemma 9.2. Suppose the dual of ℓ^* lies in $\Delta_k \in \Delta$, and let ℓ_k^* be the point selected from Δ_k. Then ℓ_k is a good line for at least one path $\mathcal{T}_i$, i.e., it meets $O(\sqrt{n})$ edges of $\mathcal{T}_i$. By Lemma 9.2, ℓ also meets only $O(\sqrt{n})$ edges of that path. Moreover, for any given ℓ, we can find ℓ_k (and thus the corresponding path $\mathcal{T}_i$) in $O(\log n)$ time, using an efficient point-location algorithm; since the map formed by Δ has only $O(n)$ faces, linear preprocessing time and storage suffices [EGSt]). Hence, the lemma follows. □

Remark 9.6. (i) Note that the best-known deterministic algorithm for constructing a single spanning path with $O(\sqrt{n})$ stabbing number has $O(n^{5/2} \log^2 n)$ time complexity. Therefore it follows from Theorem 9.4 and Lemma 9.5 that the multitree structure is better than the single path structure for all purposes except that the storage requirement is worse by a factor of $O(\log n)$. In some applications, however, it may not be possible to use a multitree structure (e.g., reporting version of the simplex range searching problem [CW] and also the counting version if subtraction is not allowed).

(ii) The spanning path obtained by our algorithm may have intersecting edges. However, if the application requires the paths to be non-self-intersecting, we can apply a technique of [EGH*] that converts a polygonal path $\mathcal{T}$ with n edges into another, non-self-intersecting path $\mathcal{T}'$, in time $O(n \log n)$, with the property that a line intersects $\mathcal{T}'$ in at most twice as many edges as it intersects $\mathcal{T}$.

(iii) If we use the randomized version of our red-blue intersection algorithm, to count the intersections between the edges of $\mathcal{T}_i$ and $\mathcal{L}$, in Matoušek's randomized algorithm [Ma1] for constructing T, then $\sigma(\mathbf{T})$ can be improved to $O(\sqrt{n} \log n)$ without increasing the time complexity of his algorithm.

Chazelle and Welzl [CW] have shown that spanning trees with low stabbing number can be used to develop an almost optimal algorithm for answering simplex range queries. Other applications of spanning trees with low stabbing number include computing a face in an arrangement of lines [EGH*], ray shooting in nonsimple polygons [A1] and implicit point location [A1]. Our algorithm improves the preprocessing time as well as query time of most of these applications. For example, Edelsbrunner *et al.* [EGH*] have shown that given a set $\mathcal{L}$ of n lines, it can be preprocessed in $O(n^{3/2} \log^2 n)$ (randomized expected) time, into a data structure of size $O(n \log n)$, using a family $\mathbf{T}$ of $O(\log n)$ spanning trees with $\sigma(T) = s$, so that, for a query point p, the face in $\mathcal{A}(\mathcal{L})$ containing p can be computed in time $O(s \log^2 n + K)$, where K is the number of edges bounding the desired face. The result of Matoušek [Ma1] implies that the preprocessing can be done deterministically in $O(n^{7/4} \log^2 n)$ time. However, if we use our algorithm for constructing the spanning trees, we obtain

Corollary 9.7. *Given a set $\mathcal{L}$ of n lines, we can preprocess it deterministically in $O(n^{3/2} \log^{\omega+1} n)$ time into a data structure of size $O(n \log n)$ so that, for a query point p, we can compute the face in $\mathcal{A}(\mathcal{L})$ containing the point p in $O(\sqrt{n} \log^3 n + K)$ time, where K is the number of edges bounding the desired face.*

Another result of [EGH*], combined with our algorithm, implies that

Corollary 9.8. *Given a set $\mathscr{L}$ of n lines, we can preprocess it in $O(n^{3/2}\log^{\omega+1} n)$ time, into a data structure of size $O(n\log^2 n)$ so that, for any ray ρ emanating from a point p in direction d, we can compute, in time $O(\sqrt{n}\log n)$, the intersection point, between ρ and the lines of $\mathscr{L}$, that lies nearest to p.*

Similarly, using the result of [CW],[2] we obtain

Corollary 9.9. *Given a set S of n points in the plane, we can preprocess it deterministically, in $O(n^{3/2}\log^{\omega+1} n)$ time, into a data structure of size $O(n\log n)$ so that, for a query line ℓ, we can compute the number of points of S lying above ℓ in $O(\sqrt{n}\log n)$ time.*

Remark 9.10. Recently Matoušek and Welzl [MW] gave an alternative deterministic algorithm to perform such half-plane range queries. Their algorithm has the same storage and query-time bounds, and its preprocessing time is only $O(n^{3/2}\log n)$.

10. Space–Query-Time Tradeoff in Triangle Range Search

Finally we consider the following problem:

> Given a set S of n points in the plane, preprocess S so that, for a query triangle Δ, we can quickly count the number of points of S lying in Δ.

As just noted, the problem has been solved by Chazelle and Welzl [CW], using a spanning tree with low stabbing number, in $O(n)$ space and $O(\sqrt{n}\log n)$ query time. In this section we study the issue of tradeoff between the allowed space and query time. Chazelle [Ch2] has proved that if we allow $O(m)$ space, then the query time is at least $\Omega(n/\sqrt{m})$. (However, this lower bound applies to an arithmetic model involving operations in a semigroup; in particular no subtractions are allowed.) For $m = n^2$, a query can be easily answered in $O(\log n)$ time, so the interesting case is when $n < m < n^2$. In this section we show that our partitioning algorithm in conjunction with Chazelle and Welzl's technique yields an algorithm that counts the number of points lying in a query triangle in $O((n/\sqrt{m})\log^{3/2} n)$ time using $O(m)$ space, where $n^{1+\varepsilon_0} \le m \le n^{2-\varepsilon_1}$, for some constants $\varepsilon_0, \varepsilon_1 > 0$. The preprocessing time of our algorithm is bounded by $O(n\sqrt{m}\log^{\omega+1/2} n)$, which is faster than that of any previously known algorithm.

---

[2] The half-plane range-searching algorithm of Chazelle and Welzl uses a single spanning tree, but it works even if we use a family of $O(\log n)$ spanning trees instead of a single tree structure, though the space complexity rises to $O(n\log n)$.

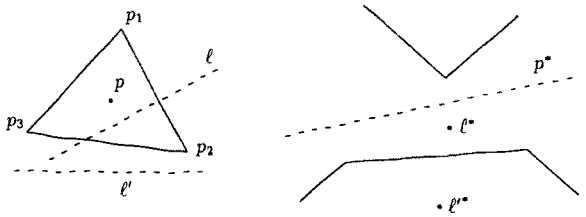

Fig. 11. A triangle Δ and its dual Δ^*.

We first establish this tradeoff for the half-plane range-search problem: "Given a set S of n points, preprocess S so that, for any query line ℓ, we can quickly count the number of points lying below ℓ."

Dualize S to a set of n lines, S^*, and partition the plane into $M = O(r^2)$ triangles $\Delta_1, \ldots, \Delta_M$ so that no triangle meets more than $O(n/r)$ lines of S^*. The dual of a triangle Δ is a 3-corridor, namely the region lying between the upper and the lower envelopes of the three lines dual to the vertices of Δ (see [HW] and Fig. 11). Let Δ^* denote the dual of Δ. A line p^* fully lies in Δ^* if and only if p lies in Δ, and a point ℓ^* is in Δ^* if and only if ℓ meets Δ. Let $S_i \subset S$ denote the points of S contained in Δ_i^*; by construction $|S_i^*| = O(n/r)$. For each Δ_i^*, construct a family $\mathbf{T}^i$ of $O(\log(n/r))$ spanning paths on the set S_i with the property that, for every line ℓ, there exists a path $\mathcal{T}_j^i \in \mathbf{T}^i$ such that ℓ intersects $O(\sqrt{n/r})$ edges of $\mathcal{T}_j^i$ (see Section 9). We preprocess every $\mathcal{T}_j^i \in \mathbf{T}^i$ into a data structure of size $O(n/r)$ for half-plane range searching, as described in [CW], so that a query can be answered in $O(\sqrt{n/r}\log(n/r))$ time.

To answer a query, we first find the 3-corridor Δ_i^* containing the query line ℓ. That is, we locate the triangle Δ_i containing the dual point ℓ^*. Let Φ_i denote the number of points in $S - S_i$ lying below ℓ, which we will have precomputed for each i. We thus only need to count the number of points of S_i lying below ℓ. By Lemma 9.5, we can find, in $O(\log(n/r))$ time, a path $\mathcal{T}_j^i \in \mathbf{T}^i$ that intersects ℓ in at most $O(\sqrt{n/r})$ edges. Moreover, the number of points of S_i lying below ℓ can be counted in $O(\sqrt{n/r}\log(n/r))$ time using $\mathcal{T}_j^i$, as in Corollary 9.9. Hence, the total query time is bounded by $O(\sqrt{n/r}\log(n/r))$. Since each $\mathbf{T}^i$ requires $O((n/r)\log(n/r))$ space, the total space used is $O(nr\log(n/r))$. We choose

$$r = \left\lceil \frac{m}{n\log(n/\sqrt{m})} \right\rceil$$

to achieve $O(m)$ space, and the query time is therefore $O((n/\sqrt{m})\log^{3/2}(n/\sqrt{m}) + \log n)$.

As for the preprocessing time, we spend $O(nr\log n\log^{\omega} r)$ time in partitioning the plane into M triangles. Let $W_i \subset S^*$ denote the set of lines lying below the triangle Δ_i, so $\Phi_i = |W_i|$. It is easily seen that for two adjacent triangles Δ_i, Δ_j, $W_i \oplus W_j \subset S_i^* \cup S_j^*$. Therefore, Φ_i, for each Δ_i, can be computed in time $O(nr)$, spending $O(n)$ time at the first triangle plus $O(n/r)$ time at each of the remaining

triangles. We can compute Φ_i, for each triangle Δ_i, in $O(nr \log n)$ time, and, by Theorem 9.4, we can construct $\mathbf{T}^i$ in $O((n/r)^{3/2} \log^{\omega+1}(n/r))$ time. It follows from [CW] that $\mathcal{T}_j^i$ can be preprocessed in $O((n/r) \log (n/r))$ time for answering half-plane range searching. Therefore the total time spent in preprocessing is

$$\begin{aligned} P(n) &= O(nr \log n \log^{\omega} r) + O\left(\left(\frac{n}{r}\right)^{3/2} \cdot r^2 \log^{\omega+1} \frac{n}{r}\right) \\ &= O\left(n \frac{m}{n \log (n/\sqrt{m})} \log n \log^{\omega} r + n^{3/2} \sqrt{\frac{m}{n \log (n/\sqrt{m})}} \log^{\omega+1} \frac{n}{\sqrt{m}}\right) \\ &= O\left(m \log^{\omega+1} n + n\sqrt{m} \log^{\omega+1/2} \frac{n}{\sqrt{m}}\right). \end{aligned}$$

By Chazelle's lower bound mentioned above, we obtain

Theorem 10.1. *Given a set S of n points in the plane and $n \log n \leq m \leq n^2$ storage, we can preprocess S, in $O(m \log^{\omega+1} n + n\sqrt{m} \log^{\omega+1/2} (n/\sqrt{m}))$ time, so that, for any query line ℓ, we can count the number of points of S lying below ℓ in time $O((n/\sqrt{m}) \log^{3/2} (n/\sqrt{m}) + \log n)$, using $O(m)$ space. This is optimal up to a polylog factor.*

Remark 10.2. (i) Matoušek's original algorithm [Ma2] can also be used to obtain the same tradeoff. However, since we use large values of r, our preprocessing is faster than that obtainable by Matoušek's algorithm. We have recently learnt that Chazelle [Ch3] has also independently obtained a similar result.

(ii) We can reduce a $\log^{1/2}(n/\sqrt{m})$ factor in the query time, if we compute a single spanning path instead of $O(\log n)$ paths. But then the (deterministic) time complexity of computing one such path rises to $O((n^3/r^3) \log (n/r))$.

(iii) Notice that the counting version of the half-plane range-query problem is more difficult than the reporting version; for the latter version, Chazelle *et al.* [CGL] have given an $O(\log n + K)$ algorithm to report all K points lying below the query line, using only $O(n)$ space and $O(n \log n)$ preprocessing.

Next, we extend the above algorithm to obtain a similar tradeoff for the *slanted range-search* problem: "Given a set S of n points, preprocess S so that, for a query segment e, we can count efficiently the number of points that lie in the semi-infinite trapezoidal strip lying directly below e." Let us denote the number of such points by $\Psi(e)$ (see Fig. 12).

Chazelle and Guibas [CG] have given an optimal algorithm for the reporting version of the slanted range-query problem, which reports all K such points in $O(\log n + K)$ time, using $O(n)$ space and $O(n \log n)$ preprocessing. Since the half-plane range-search problem is a special case of the slanted range-search problem, the lower bound on the query time for the slanted range-search problem, with $O(m)$ storage, is also $\Omega(n/\sqrt{m})$. Our tradeoff is obtained as follows.

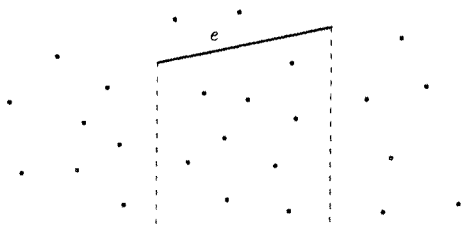

Fig. 12. Instance of a slanted range searching; $\Psi(e) = 9$.

Construct a binary tree $\mathscr{B}$ on the x-projections of the points in S as follows. Sort the points of S in increasing x order. Decompose the sorted set into n/c blocks, each containing at most c points, for some fixed constant $c > 0$, and associate each block with a leaf of $\mathscr{B}$. Each node v of $\mathscr{B}$ is thus associated with the set $S_v \subseteq S$ of points stored in the leaves of the subtree of $\mathscr{B}$ rooted at v. For each node v of $\mathscr{B}$ we preprocess the points in S_v for answering half-plane range queries, using the above algorithm, with $r = r_i$, where r_i is a parameter depending on the level i of v in $\mathscr{B}$. A segment e is called a *canonical segment* if there is a node $v \in \mathscr{B}$ such that the x-projection of e covers the x-projections of all the points in S_v, and of no other point in $S - S_v$. Observe that, for a canonical segment e, $\Psi(e)$ can be computed by solving a half-plane range query at the corresponding node. In general, a query segment e can be decomposed into $k \leq 2 \log n$ canonical subsegments $e_1, \ldots, e_k$, such that at most two of them correspond to nodes at the same level of $\mathscr{B}$ (see [PS]). Thus $\Psi(e) = \sum_{i=1}^{k} \Psi(e_i)$, which implies that $\Psi(e)$ can be computed by answering at most $2 \log n$ half-plane range queries.

Since the nodes of the same level are associated with pairwise disjoint sets of points, and we are choosing the same value of r for all nodes of the same level, the space $s(n)$ used by our algorithm is

$$s(n) = O\left(\sum_{i=1}^{\log n} nr_i \log n\right).$$

Let $m = n^\gamma$, where $1 + \varepsilon_0 \leq \gamma \leq 2 - \varepsilon_1$, for some constants $\varepsilon_0, \varepsilon_1 > 0$. If we choose $r_i = n_i^{\gamma-1}/(\log n)$, where $n_i = n/2^{i-1}$ is the size of each set S_v at level i, we have

$$\begin{aligned} s(n) &= O\left(n \log n \sum_{i=1}^{\log n} \frac{n_i^{\gamma-1}}{\log n}\right) \\ &= O\left(n \sum_{i=1}^{\log n} \left(\frac{n}{2^{i-1}}\right)^{\gamma-1}\right) \\ &= O(n^\gamma) \qquad \text{(because } \gamma > 1 + \varepsilon_0\text{)} \\ &= O(m). \end{aligned}$$

Next, the total time spent in answering a query is

$$\begin{aligned} Q(n) &= O\left(\sum_{i=1}^{\log n} \sqrt{\frac{n_i}{r_i}} \log n\right) \\ &= O\left(\sum_{i=1}^{\log n} \sqrt{n_i^{2-\gamma}} \log^{3/2} n\right) \\ &= O(n^{1-\gamma/2} \log^{3/2} n) \qquad \text{(because } \gamma \le 2 - \varepsilon_1\text{)} \\ &= O\left(\frac{n}{\sqrt{m}} \log^{3/2} n\right). \end{aligned}$$

As for the time required in preprocessing, we spend $O((n_i r_i + n_i^{3/2}\sqrt{r_i}) \log^{\omega+1} n)$ at a node of the ith level. Since there are 2^i nodes at level i, the overall preprocessing time is bounded by

$$\begin{aligned} P(n) &= \sum_{i=1}^{\log n} O(2^i(n_i r_i + n_i^{3/2}\sqrt{r_i}) \cdot \log^{\omega+1} n) \\ &= \sum_{i=1}^{\log n} O\left(\left(n \cdot r_i + \frac{n^{3/2}}{2^{i/2}} \cdot \sqrt{r_i}\right) \cdot \log^{\omega+1} n\right) \\ &= \sum_{i=1}^{\log n} O\left(\left(n \frac{n_i^{\gamma-1}}{\log n} + \frac{n^{3/2}}{2^{i/2}} \sqrt{\frac{n_i^{\gamma-1}}{\log n}}\right) \log^{\omega+1} n\right) \\ &= O\left(\left(\frac{n^\gamma}{\log n} + \frac{n^{1+\gamma/2}}{\sqrt{\log n}}\right) \log^{\omega+1} n\right) \\ &= O(n\sqrt{m} \log^{\omega+1/2} n) \qquad \text{(because } m \le n^2\text{)}. \end{aligned}$$

Therefore, we have

Theorem 10.3. *Given a set S of n points in the plane and $n^{1+\varepsilon_0} \le m \le n^{2-\varepsilon_1}$, for some constants $\varepsilon_0, \varepsilon_1 > 0$, we can preprocess S, in $O(n\sqrt{m} \log^{\omega+1/2} n)$ time, into a data structure of size $O(m)$ so that, for a query segment e, $\Psi(e)$ can be computed in $O((n/\sqrt{m}) \log^{3/2} n)$ time. This is optimal up to a polylog factor.*

Remark 10.4. (i) The remarks following Theorem 10.1 apply here as well.

(ii) If $n \log^2 n \le m \le n^{1+\varepsilon}$ for all $\varepsilon > 0$, then

$$Q(n) = O((n/\sqrt{m}) \log^2 n).$$

Similarly, if $m > n^{2-\varepsilon}$ for all $\varepsilon > 0$, then a more careful analysis shows that

$$Q(n) = O((n/\sqrt{m}) \log^{5/2} (n/\sqrt{m}) + \log n).$$

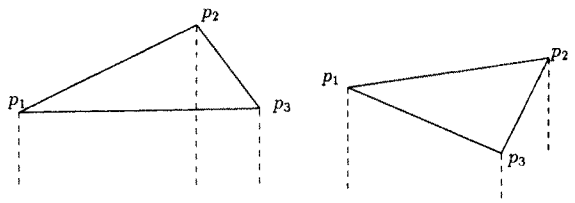

Fig. 13. Two types of triangles.

Finally, we show how to solve the triangle range-query problem using Theorem 10.3. Let Δ denote a triangle with vertices p_1, p_2, and p_3. Assume that p_1 is the leftmost vertex and $\overline{p_1p_2}$ lies above $\overline{p_1p_3}$ (see Fig. 13). If $x(p_2) \leq x(p_3)$, then the number of points in Δ is

$$\Psi(\overline{p_1p_2}) + \Psi(\overline{p_2p_3}) - \Psi(\overline{p_1p_3}),$$

and if $x(p_2) > x(p_3)$, then the number is

$$\Psi(\overline{p_1p_2}) - \Psi(\overline{p_1p_3}) - \Psi(\overline{p_2p_3}).$$

It thus follows from Theorem 10.3 that

Theorem 10.5. *Given a set S of n points in the plane and $n^{1+\varepsilon_0} \leq m \leq n^{2-\varepsilon_1}$ for some constants $\varepsilon_0, \varepsilon_1 > 0$, we can preprocess S, in $O(n\sqrt{m}\log^{\omega+1/2} n)$ time, into a data structure of size $O(m)$ so that, for a query triangle Δ, we can count the number of points contained in Δ in $O((n/\sqrt{m})\log^{3/2} n)$ time.*

Remark 10.6. (i) If $n\log^2 n \leq m < n^{1+\varepsilon}$ for any $\varepsilon > 0$, then the query time becomes $O((n/\sqrt{m})\log^2 n)$. Similarly, if $m > n^{2-\varepsilon}$ for all $\varepsilon > 0$, then $Q(n) = O((n/\sqrt{m})\log^{5/2}(n/\sqrt{m}) + \log n)$.

(ii) Notice that we use subtraction to count the number of points lying inside a triangle. It is not known whether Chazelle's lower bound [Ch2] can be extended to the case where we use subtraction, that is to the group model. Therefore, we do not know how sharp our bounds are in that model.

11. Conclusions

In this paper we have presented various applications of our partitioning algorithm, described in a companion paper [A3]. Most of the algorithms described in this paper have a similar flavor. In particular, we first give a simple but slower algorithm with running time roughly $m\sqrt{n}$ or $n\sqrt{m}$, and then combine it with our partitioning algorithm to obtain a faster algorithm. As mentioned in the Introduction, we do not need the second phase of our partitioning algorithm in several

applications, because the number of triangles produced in the first phase is sufficiently small to imply the asserted running time. For example, consider the problem of computing incidences between a given set of m points and a set of n lines in the plane, and suppose we perform only the first phase of our partitioning algorithm. Equation (2.2) implies that the running time of the algorithm is

$$T(m, n) \leq \sum_{i=1}^{M} O(m_i\sqrt{n_i}\log^{1/2} n_i + n_i \log n_i) + O((m + nr\log^{\omega} r)\log n),$$

where $\sum_{i=1}^{M} m_i = m$, $n_i = O(n/r)$, and $M = O(r^2 \log^{\omega} r)$. Therefore,

$$T(m, n) = O\left(\sqrt{\frac{n}{r}}\log^{1/2}\frac{n}{r}\sum_{i=1}^{M} m_i\right) + O((nr\log^{\omega} r + m)\log n)$$

$$= O\left(\frac{m\sqrt{n}}{\sqrt{r}}\log^{1/2}\frac{n}{r} + nr\log^{\omega} r\log n\right) + O(m\log n).$$

Again, if we choose

$$r = \max\left\{\frac{m^{2/3}}{n^{1/3}\log^{1/3} n\log^{2\omega/3}(m/\sqrt{n})}, 2\right\},$$

we get

$$T(m, n) = O\left(m^{2/3}n^{2/3}\log^{2/3} n\log^{\omega/3}\frac{m}{\sqrt{n}} + (m + n)\log n\right).$$

Similarly, we can show that we do not need the second phase of the partitioning procedure for the algorithms presented in Sections 3–7. However, we do need if for approximate half-plane range searching, constructing spanning trees with low stabbing number, and simplex range searching.

Although this paper describes algorithms for several problems, which improve previous, often randomized, techniques, there is no reason to believe that all the algorithms presented here are close to optimal. Some of these problems that deserve further attention are Hopcroft's problem, counting segment intersections, red-blue intersection, and constructing spanning trees with low stabbing number. One of the most intriguing open problems is whether there exists an $O(n \log n)$ algorithm (or for that matter any algorithm faster than those given above) for counting segment intersections, or for counting (or just detecting) red-blue segment intersections. The "red-blue" version of such an algorithm would also be able to detect an incidence between points and lines (Hopcroft's problem) in the same time. Another interesting open problem is to obtain a faster algorithm for constructing a spanning tree (or a family of spanning trees) with low stabbing

number, because that will improve the preprocessing time of various other problems (as in [EGH*] and [A1]).

Acknowledgments

I would like to thank my adviser Micha Sharir for encouraging me to work on this problem, for several valuable discussions, and for reading earlier versions of this paper. I am also thankful to Boris Aronov for helpful discussions. Thanks are also due to two anonymous referees for their useful comments.

References

[A1] P. K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *Proceedings of the 5th Annual Symposium on Computational Geometry*, 1989, pp. 315–325.

[A2] P. K. Agarwal, Intersection and decomposition algorithms for arrangements of curves in the plane, Ph.D. Thesis, Dept. Computer Science, New York University, New York, August 1989.

[A3] P. K. Agarwal, Partitioning arrangements of lines: I. An efficient deterministic algorithm, *Discrete and Computational Geometry*, this issue (1990), 449–483.

[AS] P. K. Agarwal and M. Sharir, Red-blue intersection detection algorithms with applications to motion planning and collision detection, *SIAM Journal on Computing* **19** (1990), 297–322.

[AEGS] B. Aronov, H. Edelsbrunner, L. Guibas, and M. Sharir, Improved bounds on the complexity of many faces in arrangements of segments, Technical Report 459, Dept. Computer Science, New York University, July 1989.

[BO] J. Bentley and T. Ottmann, Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers* **28** (1979), 643–647.

[B] K. Brown, Algorithms for reporting and counting geometric intersections, *IEEE Transactions on Computers* **30** (1981), 147–148.

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

[Ch1] B. Chazelle, Reporting and counting segment intersections, *Journal of Computer and Systems Sciences* **32** (1986), 156–182.

[Ch2] B. Chazelle, Lower bounds on the complexity of polytope range searching, *Journal of the American Mathematical Society* **2** (1989), 637–666.

[Ch3] B. Chazelle, Private communication.

[CE] B. Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 590–600. (Also to appear in *Journal of the Association for Computing Machinery*.)

[CEGS1] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Algorithms for bichromatic line segment problems and polyhedral terrains, Manuscript, 1989.

[CEGS2] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Lines in space: combinatorics, algorithms and applications, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing*, 1989.

[CF] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 539–549.

[CG] B. Chazelle and L. Guibas. Fractional cascading: II. Applications, *Algorithmica* **1** (1986), 163–191.

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

[CW] B. Chazelle and E. Welzl, Quasi-optimal range searching in spaces with finite VC-dimension, *Discrete and Computational Geometry* **4** (1989), 467–490.

[Cl1] K. Clarkson, New applications of random sampling in computational geometry, *Discrete and Computational Geometry* **2** (1987), 195–222.

[Cl2] K. Clarkson, Applications of random sampling in computational geometry II, *Proceedings of the 4th Annual Symposium on Computational Geometry*, 1988, pp. 1–11.

[CEG*] K. Clarkson, H. Edelsbrunner, L. Guibas, M. Sharir, and E. Welzl, Combinatorial complexity bounds for arrangements of curves and surfaces, *Discrete and Computational Geometry* **5** (1990), 99–160.

[CS] R. Cole and M. Sharir, Visibility problems for polyhedral terrains, *Journal of Symbolic Computation* **7** (1989), 11–30.

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

[D] F. Dévai, Quadratic bounds for hidden line elimination, *Proceedings of the 2nd Annual Symposium on Computational Geometry*, 1986, pp. 269–275.

[EG] H. Edelsbrunner and L. Guibas, Topologically sweeping an arrangement, *Journal of Computer and Systems Sciences* **38** (1989), 165–194.

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

[EGSh] H. Edelsbrunner, L. Guibas, and M. Sharir, The complexity and construction of many faces in arrangements of lines and segments, *Discrete and Computational Geometry* **5** (1990), 161–196.

[EGSt] H. Edelsbrunner, L. Guibas, and G. Stolfi, Optimal point location in monotone subdivisions, *SIAM Journal on Computing* **15** (1986), 317–340.

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

[EW1] H. Edelsbrunner and E. Welzl, Constructing belts in two-dimensional arrangements with applications, *SIAM Journal on Computing* **15** (1986), 271–284.

[EW2] H. Edelsbrunner and E. Welzl, On the maximal number of edges of many faces in an arrangement, *Journal of Combinatorial Theory, Series A* **41** (1986), 159–166.

[GOS1] L. Guibas, M. Overmars, and M. Sharir, Ray shooting, implicit point location, and related queries in arrangements of segments, Technical Report 433, Dept. Computer Science, New York University, March 1989.

[GOS2] L. Guibas, M. Overmars, and M. Sharir, Counting and reporting intersections in arrangements of line segments, Technical Report 434, Dept. Computer Science, New York University, March 1989.

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

[K] D. Kirkpatrick, Optimal search in planar subdivisions, *SIAM Journal on Computing* **12** (1983), 28–35.

[MS] H. Mairson and J. Stolfi, Reporting and counting intersections between two sets of line segments, in *Theoretical Foundations of Computer Graphics and CAD*, ed. R. Earnshaw, NATO ASI Series, F40, Springer-Verlag, Berlin, 1988, pp. 307–325.

[Ma1] J. Matoušek, Constructing spanning trees with low crossing numbers, to appear in *Informatique Théorique et Appliquée*.

[Ma2] J. Matoušek, Construction of ε-nets, *Discrete and Computational Geometry* **5** (1990), 427–448.

[MW] J. Matoušek and E. Welzl, Good splitters for counting points in triangles, *Proceedings of the 5th Annual Symposium on Computational Geometry*, 1989, pp. 124–130. (Also to appear in *Journal of Algorithms*.)

[Mc] M. McKenna, Worst case optimal hidden surface removal, *ACM Transactions on Graphics* **6** (1987), 9–31.

[Mu] K. Mulmuley, A fast planar partition algorithm, *Proceedings of the 29th Annual IEEE Symposium on Foundations of Computer Science*, 1988, pp. 580–589.

[PS] F. Preparata and M. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, Heidelberg, 1985.

[STa] N. Sarnak and R. Tarjan, Planar point location using persistent search trees, *Communications of the Association for Computing Machinery*, **29** (1986), 669–679.

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

[W] E. Welzl, Partition trees for triangle counting and other range searching problems, *Proceedings of the 4th Annual Symposium on Computational Geometry*, 1988, pp. 23–33.

Received May 20, 1989, and in revised form January 5, 1990.