Algorithmica (1994) 11: 116–132

Algorithmica

# Algorithms for Bichromatic Line-Segment Problems and Polyhedral Terrains[1]

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

**Abstract.** We consider a variety of problems on the interaction between two sets of line segments in two and three dimensions. These problems range from counting the number of intersecting pairs between $m$ blue segments and $n$ red segments in the plane (assuming that two line segments are disjoint if they have the same color) to finding the smallest vertical distance between two nonintersecting polyhedral terrains in three-dimensional space. We solve these problems efficiently by using a variant of the segment tree. For the three-dimensional problems we also apply a variety of recent combinatorial and algorithmic techniques involving arrangements of lines in three-dimensional space, as developed in a companion paper.

**Key Words.** Computational geometry, Line-segment intersection, Segment trees, Lines in space, Polyhedral terrains, Deterministic and randomized algorithms.

**1. Introduction.** In this paper we study a variety of problems on the interaction between two sets of line segments in two and three dimensions. These problems can all be solved efficiently by using a variant of the well-known *segment tree* (see, e.g., [PS]). We begin by listing the problems and their solutions. More background information and references are given later in the sections that address the individual problems.

PROBLEM 1. Let $B$ be a set of $m$ pairwise disjoint "blue" line segments in the plane, let $R$ be a set of $n$ pairwise disjoint "red" line segments, and define $N = m + n$. Report all bichromatic pairs of intersecting line segments. We solve

[1] Work on this paper by the first author has been supported in part by the National Science Foundation under Grant CCR-9002352. Work by the second author was supported in part by the National Science Foundation under Grant CCR-8714565. The fourth author has been supported in part by the Office of Naval Research under Grant N0014-87-K-0129, by the National Science Foundation under Grant NSF-DCR-83-20085, by grants from the Digital Equipment Corporation and the IBM Corporation, and by a grant from the US–Israeli Binational Science Foundation.

[2] Department of Computer Science, Princeton University, Princeton, NJ 08544, USA.
[3] Department of Computer Science, University of Illinois at Urbana–Champaign, Urbana, IL 61801, USA.
[4] DEC Systems Research Center, Palo Alto, CA 94301, USA, and Computer Science Department, Stanford University, CA 94305-4055, USA.
[5] Courant Institute of Mathematical Sciences, New York University, New York, NY 10012, USA, and School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

Received March 12, 1990; revised January 28, 1992. Communicated by D. T. Lee.

this problem in time $O(N \log N + k)$ and linear storage, where $k$ is the number of blue–red intersecting pairs (see Sections 2 and 3.1).[6]

PROBLEM 2. Given the same data as in Problem 1, count how many bichromatic pairs of intersecting line segments there are. The solution to this problem takes time $O(N \log N)$ and linear storage (Sections 2 and 3.1).

PROBLEM 3. Given the same data again and an integer $k$, find the $k$th bichromatic intersection point from the left. This point can be found in time $O(N \log N)$ and linear storage (Section 3.2).

PROBLEM 4. Given two convex polytopes, $\mathscr{A}$ and $\mathscr{B}$, in three dimensions with $m$ and $n$ edges, $N = m + n$, compute the number of faces bounding their *Minkowski sum*, that is, the polytope $\{p + q \mid p \in \mathscr{A}, q \in \mathscr{B}\}$. This takes time $O(N \log N)$ and linear storage (Section 3.3).

PROBLEM 5. A *polyhedral terrain* is the graph of a real bivariate continuous function, or, equivalently, a piecewise linear continuous surface in three dimensions that meets every vertical line in exactly one point. Given two nonintersecting polyhedral terrains with $m$ and $n$ edges, $N = m + n$, find the smallest vertical distance between them. We give an algorithm that solves this problem in $O(N^{4/3+\varepsilon})$ time and storage, for any $\varepsilon > 0$ (Section 5).

PROBLEM 6. Given the same data as in Problem 5 but allowing the two terrains to intersect, compute their *upper envelope*, that is, pointwise maximum. For this problem we have an algorithm that takes time $O(N^{3/2+\varepsilon} + k \log^2 N)$ and worst-case storage $O(N)$, where $\varepsilon$ is an arbitrary positive constant and $k$ is the number of edges of the resulting envelope (Section 6).

While Problems 1–4 can be solved using a variant of the segment tree plus some standard data structuring techniques, Problems 5 and 6 require additional techniques (developed in [CEG$^+$]; see also below) which employ (derandomized) *random sampling* of the data.

The common theme of the solutions to the above problems is that they all exploit the same data structure, the *hereditary segment tree*, which is introduced in this paper. It is very similar to the standard segment tree, except that it stores two sets of line segments in a manner that makes it easy to keep track of their interactions. The data structure is described in detail in Section 2, where we demonstrate its applications to Problems 1 and 2. Using certain modifications and extension of the basic technique of Section 2 we improve the solutions to Problems 1 and 2 in Section 3 and we also give solutions to Problems 3 and 4.

Section 4 discusses another extension of the basic hereditary segment tree used in Sections 5 and 6 to solve Problems 5 and 6. Both problems ask questions about

[6] Unless the base of a logarithm is explicitly noted it is assumed to be 2.

the interaction between two polyhedral terrains in three-dimensional space. We apply the hereditary segment tree to their $xy$-projections to obtain a compact representation of the way in which the terrain edges cross above or below one another. This enables us to break the problem into smaller subproblems, with the advantage that in each subproblem we can extend the edges to lines without changing the solution to the subproblem. These subproblems, among several related ones, have been studied in a companion paper [CEG$^+$] that deals with combinatorial and algorithmic problems for arrangements of lines in three dimensions. Combining the techniques developed in [CEG$^+$] with our hereditary segment tree, we obtain reasonably efficient solutions for Problems 5 and 6.

## 2. The Basic Hereditary Segment Tree.

In this section we introduce our basic data structure, and apply it to Problems 1 and 2. Recall that in these problems we have two collections, $B = \{b_1, b_2, \ldots, b_m\}$ and $R = \{r_1, r_2, \ldots, r_n\}$, of $m$ "blue" and $n$ "red" line segments in the plane, so that any two line segments of the same color are disjoint. Our goal is to compute (or just count) all bichromatic pairs of intersecting line segments. Define $N = m + n$. The straightforward application of our basic techniques leads to an algorithm that reports all intersecting bichromatic pairs in time $O(N \log^2 N + k)$, where $k$ is the number of such pairs, and uses $O(N \log N)$ storage, and to another algorithm that counts these pairs in time $O(N \log^2 N)$ and uses $O(N \log N)$ storage. Using additional methods we show in Section 3.1 how to remove a factor $\log N$ from each bound.

Our first step is to construct a segment tree, $\mathcal{T}$, on the interval decomposition of the $x$-axis induced by the $x$-coordinates of the endpoints of the given line segments. More specifically, $\mathcal{T}$ is defined as follows. Let us assume that the $x$-coordinates of the $2N$ endpoints are all distinct; if they are not we can get this property by simulating an arbitrarily small perturbation of the $x$-coordinates. As a result, the $2N$ $x$-coordinates decompose the $x$-axis into $2N + 1$ (atomic) intervals. We define these intervals as closed sets. $\mathcal{T}$ is a minimum height ordered binary tree whose $i$th leaf corresponds to the $i$th atomic interval from the left. Each interior node $\mu$ represents an interval $I_\mu$, which is the union of the intervals associated with the leaves of the subtree rooted at $\mu$. Alternatively, we can think of $\mu$ as representing the vertical slab $\sigma_\mu = \{(x, y) | x \in I_\mu, y \in \mathcal{R}\}$. In the standard segment tree, each node $\mu$ (internal and external) has an associated list $L_\mu$, consisting of all line segments $s$, with the property that the vertical projection of $s$ contains $I_\mu$ but does not contain $I_\kappa$, where $\kappa$ is the parent of $\mu$. In our version we maintain four lists with each node $\mu$. First we split $L_\mu$ into two *standard lists*, $B_\mu$ and $R_\mu$, consisting of the blue and red line segments in $L_\mu$. Furthermore, we associate with $\mu$ two additional *hereditary lists* $B_\mu^*$ and $R_\mu^*$. The list $B_\mu^*$ contains the blue line segments stored in $L_\nu$ for all proper descendants $\nu$ of $\mu$; $R_\mu^*$ is the red analog of $B_\mu^*$. In other words, whenever we store a line segment $s$ in some $B_\nu$ or $R_\nu$, we also store the line segment in the hereditary list of each proper ancestor $\mu$ of $\nu$ (see Figure 2.1).

Even though a line segment $s$ can be stored in the standard lists of more than one descendant of a node $\mu$, we still keep only one copy of $s$ in the appropriate

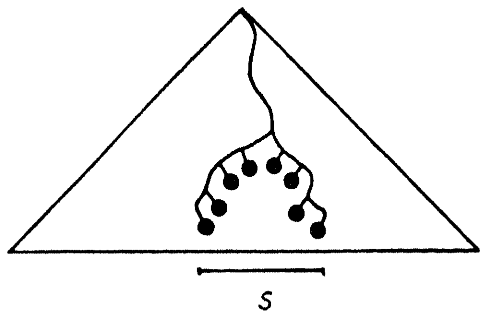

**Fig. 2.1.** The segment $s$ is stored in the standard lists of all marked nodes and in the hereditary lists of all ancestors of the marked nodes. These ancestors lie on two paths in the tree.

hereditary list of $\mu$. For the applications that follow, it is helpful to regard each element of a standard or hereditary list of a node $\mu$ as representing the subsegment $s \cap \sigma_\mu$ of the corresponding line segment $s$. As suggested by this interpretation, we refer to line segments stored in the standard lists as *long segments* and to those stored in the hereditary lists as *short segments*.

This is our basic hereditary segment tree, which is borrowed from [C1]. In some of the applications given below we modify it slightly to improve its performance. Modifications different from the ones to be described have been also used in [VW] for point enclosure problems.

In a standard segment tree, the total size of all the lists $L_\mu$ is $O(N \log N)$. Our first observation is that this is also the total size of all hereditary lists. Indeed, each line segment $s$ is stored in $O(\log N)$ standard lists. The interior nodes that store $s$ in their hereditary lists form two paths both starting at the root of $\mathscr{T}$; thus $s$ is stored in at most $O(\log N)$ hereditary lists and therefore $O(\log N)$ times altogether (see also [VW]).

We next show how to use the hereditary segment tree to solve Problems 1 and 2. We first construct the tree $\mathscr{T}$ for the given collections $B$ and $R$ in time $O(N \log N)$, using standard techniques. At each node $\mu$ of $\mathscr{T}$ we compute (or, in Problem 2, just count) the intersections between

(i) a line segment in $B_\mu$ and one in $R_\mu$,
(ii) a line segment in $B_\mu$ and one in $R_\mu^*$, and
(iii) a line segment in $B_\mu^*$ and one in $R_\mu$.

We refer to type (i) intersections as "long–long intersections," and to the two other types as "long–short intersections." According to our interpretation, in all three cases we consider only the portions of the line segments within $\sigma_\mu$.

Note that by repeating this step for all nodes $\mu$ of $\mathscr{T}$ we detect all bichromatic intersecting pairs; moreover, each pair is detected exactly once. Indeed, suppose $b_i \cap r_j \neq \varnothing$. Then there is a unique leaf $\lambda$ of $\mathscr{T}$ such that the point $p = b_i \cap r_j$ lies in $\sigma_\lambda$ (assuming that the $x$-coordinate of $p$ is different from all endpoints of atomic intervals). Moreover, $b_i$ is stored in the standard list of some unique node along the path from $\lambda$ to the root, and so is $r_j$. Let these nodes be $\kappa_i$ and $\kappa_j$, respectively.

If $\kappa_i = \kappa_j$, then $p$ will be detected as a long–long intersection at $\kappa_i$; otherwise, it will be detected as a long–short intersection at either $\kappa_i$ or $\kappa_j$, whichever is higher in $\mathscr{T}$. This observation is significant because when we deal with problems that involve counting intersections, rather than reporting them, we can simply sum up the numbers computed at the nodes.

COMPUTING LONG–LONG INTERSECTIONS. Let $m_\mu = |B_\mu|$ and $n_\mu = |R_\mu|$. Sort each of these lists in increasing $y$-order; since the line segments in each list are pairwise disjoint, each of these orders is fixed throughout the slab $\sigma_\mu$. Let $(b_1, b_2, \ldots, b_{m_\mu})$ and $(r_1, r_2, \ldots, r_{n_\mu})$ be the two sorted lists. Next we merge the two lists twice, once according to their order along the left boundary of $\sigma_\mu$, and once according to their order along the right boundary. This allows us to determine for each blue line segment $b \in B_\mu$ two indices, $i_b$ and $j_b$, such that $b$ lies between $r_{i_b}$ and $r_{i_b+1}$ along the left boundary and between $r_{j_b}$ and $r_{j_b+1}$ along the right boundary. However, then the red line segments intersecting $b$ within $\sigma_\mu$ are precisely $r_{i_b+1}, \ldots, r_{j_b}$ if $i_b < j_b$, and $r_{j_b+1}, \ldots, r_{i_b}$ otherwise. We can therefore enumerate all long-long intersections within $\sigma_\mu$ in time $O(m_\mu \log m_\mu + n_\mu \log n_\mu + k_\mu)$, where $k_\mu$ is the number of such intersecting pairs. Alternatively, we can calculate the number $k_\mu$ in time $O(m_\mu \log m_\mu + n_\mu \log n_\mu)$.

COMPUTING LONG–SHORT INTERSECTIONS. Without loss of generality, consider only intersections between $B^*_\mu$ and $R_\mu$. Let $m^*_\mu = |B^*_\mu|$ and $n_\mu = |R_\mu|$. As above, we sort the list $R_\mu$ in ascending $y$-order and write $(r_1, r_2, \ldots, r_{n_\mu})$ for the resulting sequence. Then, for each $b \in B^*_\mu$, we locate the two endpoints of $b$ (within $\sigma_\mu$) in the list $R_\mu$ using binary search twice. The line segments of $R_\mu$ that lie between these two locations are precisely those that intersect $b$ within $\sigma_\mu$. Thus, in time $O((m^*_\mu + n_\mu) \log n_\mu + k^*_\mu)$ we can find all $k^*_\mu$ intersecting pairs between $B^*_\mu$ and $R_\mu$. To count these pairs takes time $O((m^*_\mu + n_\mu) \log n_\mu)$. Intersections between $R^*_\mu$ and $B_\mu$ are handled symmetrically.

Summing up these bounds over all nodes $\mu$ of $\mathscr{T}$, taking also into account the cost of constructing $\mathscr{T}$, and observing that the total size of all lists is $O(N \log N)$, we obtain the bounds on reporting and counting bichromatic pairs of intersecting line segments mentioned at the beginning of this section.

## 3. Modifications, Extensions, and Applications.

Having introduced our basic data structure, we continue our presentation with a number of modifications, extensions, and further applications, proceeding down the list of problems given in the introduction. Section 3.1 presents the additional techniques needed to improve the performance of the two intersection algorithms of Section 2, and Sections 3.2 and 3.3 discuss Problems 3 and 4.

### *3.1. Improving the Bichromatic Intersection Algorithms.*

In this section we show how to improve both the storage and the time bound of the two intersection algorithms in Section 2 using standard data structuring techniques.

Let us first address the issue of reducing the storage requirements. This is done by noting that for most of the listed problems (including Problems 1 and 2) we do not need to maintain the entire segment tree, but only process it node by node. We therefore traverse the tree in preorder so that at any time we maintain only a single path of $\mathcal{T}$. Assuming we have the standard lists, $B_\kappa$ and $R_\kappa$, as well as the hereditary lists, $B_\kappa^*$ and $R_\kappa^*$, at a node $\kappa$, we can construct the lists of a child $\mu$ simply by scanning $B_\kappa^*$ and $R_\kappa^*$. To initialize those lists at the root $\rho$ we note that $B_\rho = R_\rho = \varnothing$ and $B_\rho^* = B$ and $R_\rho^* = R$. Notice that a line segment in $B_\kappa$ and $R_\kappa$ can be neither in a standard nor in a hereditary list of $\mu$. Let us now argue that the total size of all lists along a path is at most proportional to $N$. This is easy to see for the standard lists because a line segment can be in at most one standard list along a single path. The argument for the hereditary lists is based on the well-balancedness of the segment tree (and on the assumption that the $x$-coordinates of the endpoints are pairwise different). If a line segment is in $B_\kappa^*$ or in $R_\kappa^*$, then at least one of its endpoints lies in the vertical slab $\sigma_\kappa$ of $\kappa$. However, then the number of endpoints in $\sigma_\kappa$ is one less than the number of leaves in the subtree rooted at $\kappa$. Since the number of leaves in the subtrees of nodes on a path decreases geometrically, the total count and thus the total size of the hereditary lists of these nodes is $O(N)$.

To cut down the time from $O(N \log^2 N)$ to $O(N \log N)$ we need to achieve two goals. One is to avoid having to sort the standard lists at each node of $\mathcal{T}$, and the other is to speed up the search of elements of hereditary lists in the standard lists.

To address the first issue, we begin by finding a linear extension of the following relation on the blue line segments: $b_i \prec b_j$ if the $x$-projections of $b_i$ and $b_j$ overlap and $b_j$ lies above $b_i$ along a vertical line that intersects both. The relation can be computed in time $O(m \log m)$ by a simple left to right sweep, and a linear extension can then be found in time $O(m)$. We sort $R$ in a similar manner. Then we do the preorder traversal and the construction of the lists as described above. Since the standard lists and the hereditary lists of a node $\mu$ are derived by scanning $B_\kappa^*$ and $R_\kappa^*$, where $\kappa$ is the parent of $\mu$, they are in sorted order already. Doing it this way, we maintain the sortedness of all lists throughout the traversal. The total time consumed by this procedure is proportional to the total size of all lists which is $O(N \log N)$ as argued before.

The second step in speeding up the binary search can be achieved by "fractionally cascading" the standard lists (see [CG] for a complete description of the data structuring technique called *fractional cascading*). Here is what we mean by this. As we go down the current path we maintain each standard list in a padded-up form, so that it also contains some elements of the standard lists of ancestor nodes along the path. As we go down from a node $\kappa$ to one of its children $\mu$, we take every fourth element of the (padded) list $B_\kappa$, pass down these elements to $\mu$ and merge them with $B_\mu$. Similar action is performed on $R_\kappa$ and $R_\mu$. These operations neither increase the storage nor the time of the algorithm by more than a constant factor. Note that the fractional cascading technique is performed on-the-fly, which is somewhat different from the standard static version of the technique (as described in [CG]). Also it proceeds downward, which is the reverse of the more standard upward direction generally used when cascading on trees.

The benefits of the cascading technique are reaped when we back up from $\mu$ to $\kappa$. We can assume inductively that each endpoint of any (short or long) red line segment $r \in R_\mu^* \cup R_\mu$ has been located among the line segments in the standard blue list $B_\mu$. Since this list also contains a portion of the list $B_\kappa$, it is easy, by maintaining appropriate pointers between these lists and by investing an additional constant time per endpoint, to locate all these endpoints among the line segments in $B_\kappa$ as well. Similar speed-up is achieved in a completely symmetric manner for locating the blue endpoints in the list $R_\kappa$. Notice that we thus construct the lists in preorder and evaluate them in an order which is similar to postorder. More specifically, the evaluation at a node $\kappa$ is done in two steps, once when we back up from its left child and then again when the recursion returns from its right child. The preceding considerations imply our first main result.

THEOREM 3.1. *Given a set of $m$ blue and a set of $n$ red line segments in the plane, $N = m + n$, such that no two line segments with the same color intersect.*

(i) *The $k$ bichromatic pairs of intersecting line segments can be reported in time $O(N \log N + k)$ and storage $O(N)$.*
(ii) *The bichromatic pairs of intersecting line segments can be counted in time $O(N \log N)$ and storage $O(N)$.*

The first of these two results is not new; indeed, the algorithm of Mairson and Stolfi [MaS] achieves the same asymptotic bounds, and the time-complexity is also matched by the more general algorithm of Chazelle and Edelsbrunner [CE]. The second result is new. We note that the best-known algorithms for *counting* intersections between line segments are much inferior to the results of the theorem: they run in time close to $O(n^{4/3})$ [A1], [GOS], [C2], [C3].

*3.2. Order Statistics for Bichromatic Intersections.* We next turn to Problem 3. Let $B$ and $R$ be two collections of $m$ blue and $n$ red line segments ($N = m + n$) with no intersecting monochromatic pairs. Given an integer $k$, we wish to find the $x$-coordinate $\xi(k)$ of the intersection point that has the property that there are exactly $k - 1$ bichromatic intersecting pairs whose intersection points lie to the left of the vertical line $x = \xi(k)$. We assume that the $x$-coordinates of the intersection points are pairwise distinct as well as distinct from the endpoints of atomic intervals; otherwise, this property can be simulated [EM]. We also assume that $k$ is not larger than the total number of bichromatic intersecting pairs; we already know how to compute the latter number in time $O(N \log N)$ and linear storage.

An efficient solution has been recently obtained for the related problem where each line segment is a line and there is no bichromatic condition. To find $\xi(k)$ in this case takes time $O(N \log N)$ as noted by Cole *et al.* [CSSS]. In contrast, very little is known about the monochromatic case for line segments. The most efficient known way just to compute the total number of intersecting pairs runs in time

$O(N^{4/3}(\log N)^{1/3})$ and linear storage [C2]. However, with the guidance of the hereditary segment tree, we can solve the problem efficiently in the bichromatic case as follows.

Our first step is to locate a leaf, $\lambda$, of the tree $\mathcal{T}$ such that $\sigma_\lambda$ contains the $k$th intersection point. The simplest way to find $\lambda$ is by binary search over the $x$-coordinates of the endpoints of the given line segments. At each step of the search we are at some $x$-coordinate $\xi$, and our goal is to count the number of intersections to the left of the line $x = \xi$. This is done by clipping each line segment to its portion to the left of this line and by applying our solution to Problem 2 to the clipped collection. It follows that we can find $\lambda$ in time $O(N \log^2 N)$ and linear storage.

Suppose there are $k_0 < k$ intersections to the left of $\sigma_\lambda$, and define $k_1 = k - k_0$. We can now collect all line segments that cross the slab $\sigma_\lambda$ into a set $\Lambda$ and extend these line segments to full lines. Now we apply the algorithm of [CSSS] to find the $(k_1 + k_2)$th leftmost intersection in the arrangement of these lines, where $k_2$ is the number of intersections of these lines to the left of $\sigma_\lambda$ (which is easily computed in time $O(N \log N)$). This gives us an $O(N \log^2 N)$ algorithm for finding the $k$th leftmost bichromatic intersection.

Agarwal [A2] has suggested to us a way to cut down the running time to $O(N \log N)$. The bottleneck is the computation of the leaf $\lambda$. This can be done by binary search in $\mathcal{T}$, provided that we can determine quickly how many bichromatic intersections lie in a given slab $\sigma_\mu$. Suppose that we can do that in time $O(h_\mu + k_\mu d_\mu \log N)$, where

(i) $h_\mu$ is the sum of the sizes of all the lists (standard or hereditary, blue or red) stored at $\mu$ or at any of its ancestors,
(ii) $k_\mu$ is the total size of the hereditary lists at $\mu$ (i.e., $k_\mu = |B_\mu^*| + |R_\mu^*|$), and
(iii) $d_\mu$ is the number of ancestors of $\mu$.

From our previous discussion, we know that $h_\mu = O(N)$, and therefore the sum $\sum_\mu h_\mu$, taken over all nodes $\mu$ on the path $\pi$ from the root to $\lambda$, is $O(N \log N)$. Similarly, $k_\mu$ cannot exceed the number of endpoints in the slab $\sigma_\mu$. Since this number decreases geometrically as we go down the path $\pi$, we have

$$\sum_{\mu \in \pi} k_\mu d_\mu \log N \leq (\log N) \sum_{i \geq 0} (i+1)N/2^i = O(N \log N),$$

and therefore we can find $\lambda$, and hence the $k$th leftmost intersection, in $O(N \log N)$ time.

It remains now to show how to compute the number of bichromatic intersections in $\sigma_\mu$ in time $O(h_\mu + k_\mu d_\mu \log N)$. Note that by using the algorithm of Theorem 3.1(ii), we can assume that the intersections in $B_\mu \cup R_\mu \cup B_\mu^* \cup R_\mu^*$ have already been counted. So, it suffices to count the intersections in $\sigma_\mu$ from among $B_\nu \cup R_\nu$, $B_\nu^* \cup R_\nu$, and $R_\nu^* \cup B_\nu$, where $\nu$ is an ancestor of $\mu$.

From the ordering of the segments in $B_\nu$ and $R_\nu$ we can compute their ordering along the bounding lines of $\sigma_\mu$, and hence, in $|B_\nu| + |R_\nu|$ time, derive the number of intersections in $B_\nu \cup R_\nu$. Next, consider the intersections in $B_\nu \cup R_\nu^*$ (the case of $B_v^* \cup R_v$ is similar). The segments of $R_v^*$ fall into two categories: those with an endpoint in $\sigma_\mu$ and those crossing $\sigma_\mu$ all the way through. To handle the latter kind is exactly the same as what we just described with $B_v$ and $R_v$, and therefore can be done in $|B_v| + |R_v^*|$ time. For the other kind of segments we must perform a binary search in $B_v$ with respect to their endpoints (with the segments clipped to within $\sigma_\mu$). Since these segments also appear in $R_\mu^*$, we find that the time spent at node $v$ is at most proportional to $|B_v| + |R_v| + |B_v^*| + |R_v^*| + k_\mu \log N$. Summing over all ancestors $v$ of $\mu$, we find that, indeed, the number of bichromatic intersections in $\sigma_\mu$ can be determined in time $O(h_\mu + k_\mu d_\mu \log N)$. Again, the segment tree need not be computed in its entirety. Only linear space is required to apply the algorithm of Theorem 3.1(ii), after which, it suffices to maintain the segment tree lists alongside the path $\pi$ (as it is discovered top-down). This limits the space requirement to $O(N)$.

THEOREM 3.2. *The* $k$th *leftmost bichromatic intersection for a collection of* $m$ *blue pairwise disjoint line segments and a collection of* $n$ *red pairwise disjoint line segments,* $N = m + n$, *can be computed in time* $O(N \log N)$ *and linear storage.*

*3.3. The Minkowski Sum of Two Convex Polytopes.* Let $\mathscr{A}$ and $\mathscr{B}$ be two convex polytopes in three-dimensional space with $m$ and $n$ edges, respectively. The *Minkowski (vector) sum* $\mathscr{C} = \mathscr{A} + \mathscr{B}$ is defined as $\{p + q \mid p \in \mathscr{A}, q \in \mathscr{B}\}$. As is well known, $\mathscr{C}$ is also a convex polytope, and an alternative way of defining (or constructing) $\mathscr{C}$ is as follows:

> Consider the unit sphere $S^2$ in space, and define on it two spherical maps, $M_\mathscr{A}$ and $M_\mathscr{B}$, as follows. For each $u \in S^2$ let $f_\mathscr{A}(u)$ denote the vertex, edge, or facet of $\mathscr{A}$ that is the intersection of $\mathscr{A}$ with its supporting plane having an outward normal direction $u$. The map $M_\mathscr{A}$ is the corresponding decomposition of $S^2$ into maximal portions on each of which $f_\mathscr{A}$ is invariant. The map $M_\mathscr{A}$ is known as the *Gaussian image* of $\mathscr{A}$. The map $M_\mathscr{B}$ is defined in a similar manner. Note that vertices of $M_\mathscr{A}$ are the directions of outer normals to the faces of $\mathscr{A}$, edges of $M_\mathscr{A}$ are portions of great circles of directions perpendicular to edges of $\mathscr{A}$, and faces of $M_\mathscr{A}$ correspond to vertices of $\mathscr{A}$. Similar properties hold for $M_\mathscr{B}$. To obtain $\mathscr{C}$, we overlay the two maps and the resulting decomposition of $S^2$ is $M_\mathscr{C}$. The faces, edges, and vertices of $\mathscr{C}$ and their adjacency structure can then be read off $M_\mathscr{C}$ using the correspondence just explained.

Note that $\mathscr{C}$ can have up to $\Theta(mn)$ facets that correspond to intersections between edges of $M_\mathscr{A}$ and of $M_\mathscr{B}$. As a matter of fact, each such facet is the Minkowski sum of the corresponding edges of $\mathscr{A}$ and $\mathscr{B}$.

$M_\mathscr{C}$ can be computed in time $O(N + k)$, where $k$ is the number of intersecting edge pairs using an algorithm by Guibas and Seidel [GS]. In order to apply their algorithm which overlays two subdivisions of the plane into convex regions, we first have to project $M_\mathscr{A}$ and $M_\mathscr{B}$ onto a plane. If we carry the projection from the origin of $S^2$ the circle arcs map to straight line segments. This way only one hemisphere is captured at a time and we have to repeat the algorithm for the other hemisphere. Each subdivision of the plane can also be viewed as a collection of pairwise disjoint line segments (if they are taken as relatively open sets), so we can apply our solutions to Problem 2 which gives the following result.

THEOREM 3.3. *Given two convex polytopes with $m$ and $n$ edges in three-dimensional space, $N = m + n$, the number of facets bounding their Minkowski sum can be computed in time $O(N \log N)$ and linear storage.*

**4. The Two-Level Hereditary Segment Tree.** Next we modify the hereditary segment tree by adding another level of preprocessing. More specifically, we process the lists of the nodes into certain tree structures. This allows us to reduce problems for blue and red line segments to a moderately large number of such problems where each blue line segment intersects each red line segment.

So let $B$ be a set of $m$ pairwise disjoint blue line segments, let $R$ be a set of $n$ pairwise disjoint red line segments, and define $N = m + n$, as usual. Construct the hereditary segment tree, $\mathscr{T}$, for $B$ and $R$. Our goal is to reduce the possible types of interactions between the line segments in the standard and the hereditary lists at any node $\mu$ of $\mathscr{T}$ as mentioned above. In particular, we would like to divide the set of blue line segments at $\mu$ into (not necessarily disjoint) subsets $B_i$ and the set of red line segments at $\mu$ into matching subsets $R_i$ so that, for each $i$,

(*) every line segment in $B_i$ intersects every line segment in $R_i$ within the slab $\sigma_\mu$.

To reduce bichromatic interactions to the kind required, we proceed as follows. For each node $\mu$, we represent the standard list $B_\mu$ (and $R_\mu$) by a minimum height ordered binary tree $\mathscr{B}_\mu$ ($\mathscr{R}_\mu$). The leaves of $\mathscr{B}_\mu$ are associated with the blue line segments in $B_\mu$, in the same order, and for each node $\tau$ of $\mathscr{B}_\mu$ we define $C_\tau$ as the list of blue line segments associated with the leaves below $\tau$; this list is explicitly stored at $\tau$. Symmetrically, each node $\tau$ of $\mathscr{R}_\mu$ stores the list $C_\tau$ of red line segments associated with the leaves below $\tau$. Consider now, for example, the blue list $B_\mu$. For each red line segment $r \in R_\mu \cup R_\mu^*$ find the sublist of $B_\mu$ consisting of line segments that intersect $r$. Since this is a contiguous sublist, it is the union of $O(\log m)$ lists $C_\tau$ stored at nodes of $\mathscr{B}_\mu$, and so we pair $r$ with each of them. In this manner, each node $\tau$ acquires a list $D_\tau$ of red line segments that cross all of its blue line segments within $\sigma_\mu$. Thus, $(C_\tau, D_\tau)$ satisfies condition (*).

Let us do an analysis of the number of pairs $(C_\tau, D_\tau)$ generated this way, and of the total size of the lists $C_\tau$ and $D_\tau$. The number of pairs $(C_\tau, D_\tau)$ is equal to the number of nodes of the binary trees that represent the standard lists which, in turn, is at most twice the total size of standard lists attached to the nodes of the segment tree. We noted before that the total size of standard lists is $O(N \log N)$ which shows that the number of subproblems of type (*) is $O(N \log N)$. Each line segment, whether blue or red, belongs to at most $2 \log N$ sets $C_\tau$ or $D_\tau$ of a single tree representing a standard list. It follows that the total size of these sets is at most a factor $2 \log N$ bigger than the total size of standard and hereditary lists. In other words, the total size of the sets $C_\tau$ and $D_\tau$ is $O(N \log^2 N)$.

**5. The Smallest Distance Between Polyhedral Terrains.** Let $\Sigma_1$ and $\Sigma_2$ be two polyhedral terrains with $m$ and $n$ edges so that $\Sigma_2$ lies completely above $\Sigma_1$, and define $N = m + n$, as usual. We wish to determine the smallest vertical distance between $\Sigma_1$ and $\Sigma_2$, that is, the length of the shortest vertical line segment connecting a point on $\Sigma_1$ to a point on $\Sigma_2$. Chazelle and Sharir [CS] give an algorithm for this problem which runs in time $O(N^{2-\delta})$ for some very small $\delta > 0$; in this section we obtain an improved solution that runs in time $O(N^{4/3+\varepsilon})$, for any $\varepsilon > 0$.

We first observe that the desired smallest vertical distance is attained either between a vertex of $\Sigma_1$ and a facet of $\Sigma_2$, between a vertex of $\Sigma_2$ and a facet of $\Sigma_1$, or between a point of an edge of $\Sigma_1$ and a point of an edge of $\Sigma_2$. In the last case, the vertical projections onto the $xy$-plane of these two edges intersect. It is this case that consumes the largest amount of time.

We begin by projecting both $\Sigma_1$ and $\Sigma_2$ vertically onto the $xy$-plane and obtain two straight-edge plane maps, $\mathscr{M}_1$ and $\mathscr{M}_2$. Then we apply standard point-location techniques (see, e.g., [EGS]) to locate each vertex of $\mathscr{M}_1$ in $\mathscr{M}_2$ and vice versa. This allows us to compute the smallest vertical distance between a vertex of one terrain and a facet of the other in total time $O(N \log N)$.

However, we still have to consider the edge–edge intersections, whose number might be quadratic. To this end, regard $\mathscr{M}_1$ as a collection of pairwise disjoint (relatively open) blue line segments and $\mathscr{M}_2$ as a similar collection of red line segments, and construct a hereditary segment tree, $\mathscr{T}$, for the two collections. At each node $\mu$ of $\mathscr{T}$ we have the two usual standard lists, $B_\mu$ and $R_\mu$, and the two hereditary lists, $B_\mu^*$ and $R_\mu^*$. As before, by checking $B_\mu$ against $R_\mu$, $B_\mu^*$ against $R_\mu$, and $R_\mu^*$ against $B_\mu$, for all nodes $\mu$ of $\mathscr{T}$, we encounter every intersection between edges of $\mathscr{M}_1$ and $\mathscr{M}_2$.

Similar to Section 4, our goal is to reduce these interactions to a collection of interactions between subsets of blue line segments and of red line segments, with the property that whenever we test a blue subset $B_i$ against a red subset $R_i$ condition (*) of Section 4 holds. For every such pair of subsets, the smallest vertical distance between the corresponding edges of $\Sigma_1$ and $\Sigma_2$ can be obtained by extending all these edges to full lines in three-dimensional space. Doing this, we do not introduce any new bichromatic intersections in the projection. Thus, we can apply the so-called towering algorithm given in [CEG$^+$] which, given two sets, $B$ and $R$, of lines in three-dimensional space, verifies that all lines in $B$ lie above all lines in $R$. Also, if the verification is successful, it computes the smallest vertical distance between a line in $B$ and a line in $R$. All this is done using a simple randomized procedure whose expected time and storage requirements are $O(M^{4/3+\varepsilon})$, for any $\varepsilon > 0$, where $M$ is the total size of the two subsets. The algorithm can be derandomized without asymptotic loss in performance, using the techniques of [C2], [M1], and [M2].

To reduce bichromatic interactions to the kind required above, we proceed as in Section 4, constructing the pairs $(C_\tau, D_\tau)$, for $\tau$ a node of $\mathscr{B}_\mu$ or $\mathscr{R}_\mu$ at a node $\mu$ of $\mathscr{T}$, in the manner described there. Because the pairs $(C_\tau, D_\tau)$ satisfy condition (*) we can apply to it the towering algorithm of [CEG$^+$] mentioned earlier. As noted in Section 4, the total size of all these lists is $O(N \log^2 N)$. It follows that we can determine the smallest vertical distance between any pair of edges of $\Sigma_1$ and $\Sigma_2$ in time and storage $O(N^{4/3+\varepsilon})$, for any $\varepsilon > 0$. In summary, we have shown the following result.

THEOREM 5.1. *The smallest vertical distance between two nonintersecting polyhedral terrains with m and n edges, $N = m + n$, can be computed in time and storage $O(N^{4/3+\varepsilon})$, for any $\varepsilon > 0$.*

## 6. The Upper Envelope of Polyhedral Terrains.

Let $\Sigma_1$ and $\Sigma_2$ be two polyhedral terrains with $m$ and $n$ edges as in the preceding section but without the disjointness assumption. Our goal is to compute the *upper envelope*, $\Sigma$, of $\Sigma_1$ and $\Sigma_2$, that is, if we think of $\Sigma_1$ and $\Sigma_2$ as the graphs of continuous, piecewise linear, bivariate functions, then $\Sigma$ is the pointwise maximum of them. Special cases and variants of this problem, mainly when one of the terrains is convex, were treated in [MeS] and [S] where solutions that are more efficient than the solution in this section are given. Here we study the general problem. For simplicity we assume nondegenerate position throughout.

Clearly, $\Sigma$ is also a polyhedral terrain. Each vertex of $\Sigma$ is either a vertex of $\Sigma_1$, a vertex of $\Sigma_2$, or the intersection between an edge of one terrain and a facet of the other. Similarly, each edge of $\Sigma$ is either a portion of an edge of $\Sigma_1$ or $\Sigma_2$, or it is the intersection between a facet of $\Sigma_1$ and a facet of $\Sigma_2$. This implies that $k$, the number of vertices, edges, and facets of $\Sigma$, is $O(mn)$, and, indeed, this upper bound is tight. Our goal is to compute $\Sigma$ in an output-sensitive manner. We achieve this with an algorithm that runs in time $O(N^{3/2+\varepsilon} + k \log^2 N)$, where $N = m + n$.

The above observations make it clear that $\Sigma$ is easy to construct once we have computed all intersections between edges of $\Sigma_1$ and facets of $\Sigma_2$ and vice versa. As a matter of fact, it suffices to produce one point on each connected component of the intersection $\Sigma_1 \cap \Sigma_2$. From each such point the component of the intersection can be obtained by a straightforward tracing procedure (see [MeS] or [S] for details). Now, the upper envelope can be constructed by putting things together in the right way. We therefore concentrate on finding all edge–facet intersections between the two terrains. We first describe the data structure used to store $\Sigma_1$ and $\Sigma_2$ and then show how to use it to find edge–facet intersections.

*The Hereditary Segment Tree for $\Sigma_1$ and $\Sigma_2$.* Project $\Sigma_1$ and $\Sigma_2$ along the $z$-axis onto the $xy$-plane to get two planar subdivisions. For technical reasons we decompose each region of the subdivisions into trapezoids by drawing, from each vertex $v$, a vertical edge (parallel to the $y$-axis) upward until it hits the first edge above $v$ and one downward until it hits the first edge below $v$. With this modification we get two trapezoidal subdivisions, $\Delta_1$ from $\Sigma_1$ and $\Delta_2$ from $\Sigma_2$. Each trapezoid has at most two nonvertical edges which we classify as *lower* and *upper* depending on whether the trapezoid lies vertically above or below the edge. Notice, however, that the trapezoids are not necessarily glued edge-to-edge and therefore $\Delta_1$ and $\Delta_2$ might not always be cell complexes. Let $B$ be the set of lower and upper edges of all trapezoids in $\Delta_1$, and define $R$ as the set of lower

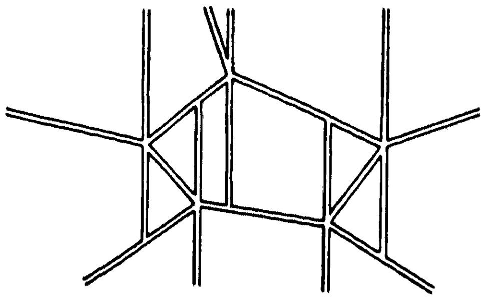

**Fig. 6.1.** The projection of a terrain onto the $xy$-plane is a planar subdivision. Each region is cut into trapezoids with unique upper and lower edges. Overlapping edges of neighboring trapezoids are interpreted as nonintersecting.

and upper edges of all trapezoids in $\Delta_2$. No two edges in $B$ (and in $R$) cross but there can be overlapping edges. In fact, each edge of the projection of $\Sigma_1$ (and of $\Sigma_2$) is now represented by two sequences of edges with the union of all edges in either sequence being the old edge. Thus, some of the edges in $B$ (and in $R$) are bound to overlap. Still, the effect of the overlap is uncritical and we can pretend that two overlapping edges are disjoint, for example, by conceptually shrinking each trapezoid by an infinitesimal amount (see Figure 6.1).

So we think of $B$ as a set of $O(m)$ pairwise disjoint blue line segments, and of $R$ as a set of $O(n)$ pairwise disjoint red line segments. The nice effect of cutting regions into trapezoids is that the lower and the upper edges of a single trapezoid are stored in exactly the same standard and hereditary lists when we construct the hereditary segment tree for $B$ and $R$. Thus, the line segments in the lists come in pairs, and the two line segments of a pair delimit the same trapezoid. As the reader might expect we indeed store $B$ and $R$ in their hereditary segment tree. However, for reasons that will become clear later we do not use a *binary* tree as usual but rather a tree where every node has $\delta$ children, where $\delta$ is a parameter to be determined later. We use a minimum height tree so its heigth is $O(\log_\delta N)$ and each edge is stored in $O(\delta \log_\delta N)$ standard and $O(\log_\delta N)$ hereditary lists.

Having constructed the hereditary segment tree in this manner, we next augment it, as in the preceding section, with pairs $(C_\tau, D_\tau)$ as follows. Let $B_\mu$ and $R_\mu$ be the standard lists at some node $\mu$ of $\mathcal{T}$. We store $B_\mu$ (and $R_\mu$) by a minimum height ordered binary tree whose leaves store the pairs of line segments, each one delimiting a common trapezoid; as in Section 4 we call this tree $\mathcal{B}_\mu$ (and $\mathcal{R}_\mu$). If $\tau$ is a node of either tree, then $C_\tau$ and $D_\tau$ are lists of line segments of opposite color so that each line segment in $C_\tau$ intersects every line segment in $D_\tau$.

What are the important properties of this data structure? Because the line segments in any list $B_\mu$ (or $R_\mu$) come in matching pairs, we can think of the list as a sequence of trapezoids each spanning the vertical strip $\sigma_\mu$ (see Figure 6.2). An endpoint of a line segment $r \in R_\mu \cup R_\mu^*$ is either within such a trapezoid (or on its left or right side) or between two adjacent trapezoids. Similarly, a list $C_\tau$ is a sequence of trapezoids spanning $\sigma_\mu$, but now we have the property that each line segment in $D_\tau$ intersects each line segment in $C_\tau$ (see Figure 6.2).

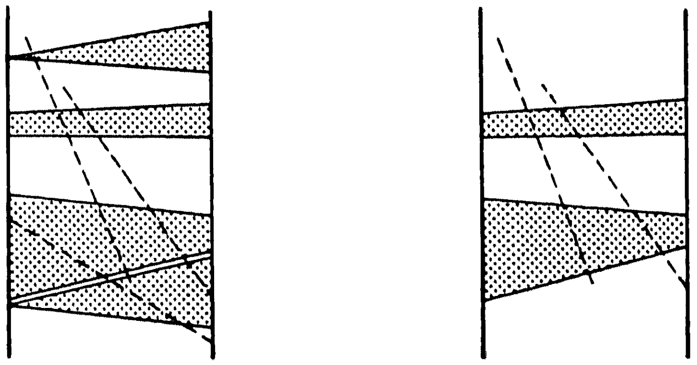

**Fig. 6.2.** To the left we see a sequence of four trapezoids in a vertical strip; each line segment of the opposite color intersects a contiguous sublist of their edges. To the right, the trapezoids are defined by a list $C_\tau$ in which case all line segments in $D_\tau$ intersect all line segments in $C_\tau$.

*How To Represent the Lists* $C_\tau$. From now on let us concentrate on edge–facet intersections between red edges and blue facets decomposed into trapezoids. To find all such intersections we store each list $C_\tau$ of blue line segments (or trapezoids, depending on which interpretation we prefer) in a data structure developed in [CEG+]. We recall the relevant properties of this data structure.

Given a set of $M$ lines in three-dimensional space, it is shown in [CEG+] how to store them in a data structure of size $O(M^{2+\varepsilon})$, for any $\varepsilon > 0$, so that given any query line $l$ it can be determined in $O(\log M)$ time whether $l$ lines above all $M$ lines, below all $M$ lines, or neither. We call this data structure the *envelope structure* of the set of $M$ lines. It takes time $O(M^{2+\varepsilon})$ to construct it. Given a list $C_\tau$ we first go back to the edges of $\Sigma_1$ that project onto the edges in $C_\tau$, then extend these edges to full lines, and finally store the envelope structure of the resulting set of lines. Because each line segment in $D_\tau$ intersects each line segment in $C_\tau$ its corresponding edge in $\Sigma_2$ lies above (or below) all edges of $\Sigma_1$ corresponding to line segments in $C_\tau$ if and only if the same is true for their extensions to lines.

A major drawback of the data structure, as described so far, is that it takes $O(N^{2+\varepsilon})$ time and storage. To improve the complexity we construct the envelope structure for a list $C_\tau$ only if it contains at most $N^{(1-\varepsilon)/2}$ line segments. A single list thus takes no more than $O(N)$ time and storage. We now set $\delta$, the number of children of any node in the segment tree, equal to $\lfloor N^{\varepsilon/2} \rfloor$. With this choice of $\delta$ all lists $C_\tau$ together take only time $O(N^{3/2+\varepsilon})$ for construction because each edge belongs to at most $O(\delta \log_\delta^2 N) = O(N^{\varepsilon/2})$ lists $C_\tau$ and the cost for each occurrence is $O(N^{(1+\varepsilon)/2})$.

*Finding Edge–Facet Intersections.* Using the envelope structures representing the lists of blue trapezoids, $C_\tau$, we can now find intersections between red edges and blue facets. Recall that a red line segment, $r$ (which is the projection of an edge of $\Sigma_2$ onto the $xy$-plane), is stored in the standard and hereditary lists of $O(\delta \log_\delta N) = O(N^{\varepsilon/2})$ nodes of the segment tree. Each occurrence of $r$ gives rise to $O(\log N)$ occurrences in lists $D_\tau$ of the binary trees that represent the standard lists of blue trapezoids at these nodes. To describe the algorithm let $\mu$ be a node of the segment tree with $r \in R_\mu \cup R_\mu^*$, and let $\tau$ be a node of $\mathscr{B}_\mu$ with $r \in D_\tau$. If $r$ lies above or below all line segments in $C_\tau$ (within $\sigma_\mu$), then we stop right away. Otherwise, we distinguish three cases.

1. If $\tau$ is a leaf, then test whether the edge corresponding to $r$ intersects the blue trapezoid in space represented at $\tau$. If it does, then report this intersection.
2. If $C_\tau$ is too large to warrant the construction of the envelope structure, then recursively query both children of $\tau$.
3. Here, $\tau$ is not a leaf and $C_\tau$ is small enough so that $\tau$ stores the envelope structure of the corresponding set of lines. In this case test whether the edge corresponding to $r$ lies above all edges corresponding to blue line segments in $C_\tau$ or below all such edges. If it lies above or below all the blue edges, then stop. Otherwise, recursively query both children of $\tau$.

For the analysis of the search algorithm let $k_r$ be the number of intersections between the edge corresponding to $r$ and the facets of $\Sigma_1$. The basic property of hereditary segment trees discussed in Section 2 is easily seen to imply that each such intersection is detected exactly once. The question that we have to answer is how much time each intersection costs and how much overhead we need for the line segment itself.

As noted earlier, $r$ belongs to the standard lists of $O(N^{\varepsilon/2})$ nodes $\mu$ of the segment tree and to the hereditary lists of only a constant number of nodes $\mu$, where the constant depends on $\varepsilon$. For each occurrence we query the tree $\mathscr{B}_\mu$. To get started we run down the tree (Case 2) until we hit the fringe nodes equipped with envelope structures; there are $O(N^{(1+\varepsilon)/2})$ of them. Thus, the overhead for each line segment is $O(N^{1/2+\varepsilon})$ and the rest of the time will be charged to the $k_r$ intersections caused by $r$.

If it turns out that the edge corresponding to $r$ neither lies above all edges corresponding to line segments in $C_\tau$ nor below all of them (Case 3), then by continuity of $\Sigma_1$ the edge corresponding to $r$ intersects $\Sigma_1$ at least once somewhere between the topmost and bottommost line segments in $C_\tau$. Now we need to distinguish two cases depending on whether the trapezoid that intersects $r$ at the point of discussion is in $C_\tau$ or it is not (which is possible because there are gaps between the trapezoids of $C_\tau$ due to edges that are either too long or too short to be in $C_\tau$). In both cases the intersection causes $O(\log N)$ queries to envelope structures in $\mathscr{B}_\mu$ and thus $O(\log^2 N)$ time. This is not all yet because the same intersection point can cause such queries also in other trees $\mathscr{B}_\mu$. We now show that the number of such trees is fortunately only at most some constant, which will lead to the final result of this section.

The $O(N^{\varepsilon/2})$ occurrences of $r$ in standard lists amounts to a decomposition of $r$ into the same number of pairwise disjoint subsegments, and each intersection point lies only on one of these subsegments and thus is charged only for the node $\mu$ that represents this subsegment of $r$. The occurrences of $r$ in hereditary lists do not decompose $r$ into disjoint segments, so an intersection point can cause queries for each such occurrence. However, as mentioned above there are only a constant number of occurrences in hereditary lists which shows that the time spent per intersection point is $O(\log^2 N)$. This implies a total time bound of $O(N^{3/2+\varepsilon} + k \log^2 N)$, where $k$ is the size of the output.

Finally, we note that instead of constructing the hereditary segment tree with envelope structures first and searching it later we can just traverse it (and its subtrees $\mathscr{B}_\mu$ and $\mathscr{R}_\mu$) in preorder and construct and search only the structures currently needed (see also Section 3.1). This leaves us with the same time bound but improves the bound on the storage to $O(N)$. We thus summarize the results of this section.

THEOREM 6.1. *Given two polyhedral terrains in three-dimensional space with a total of n edges, we can construct their upper envelope in time $O(n^{3/2+\varepsilon} + k \log^2 n)$ and storage $O(n)$, where $\varepsilon$ is an arbitrary but fixed positive constant and $k$ is the number of edges of the upper envelope.*

**Acknowledgments.** We wish to thank P. K. Agarwal for his comments and helpful suggestions.

## References

[A1] P. K. Agarwal, Intersection and decomposition algorithms for arrangements of curves in the plane, Ph.D. Thesis, New York University, 1989.

[A2] P. K. Agarwal, Private communication, 1991.

[C1] B. Chazelle, Reporting and counting segment intersections, *J. Comput. System Sci.* **32**(2) (1986), 156–182.

[C2] B. Chazelle, An optimal convex hull algorithm and new results on cuttings, *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, 1991, pp. 29–38.

[C3] B. Chazelle, Cutting hyperplanes for divide-and-conquer, *Discrete Comput. Geom.* **9** (1993), 145–158.

[CE] B. Chazelle and H. Edelsbrunner, An optimal algorithm for intersecting line segments in the plane, *J. Assoc. Comput. Mach.* **39** (1992), 1–54.

[CEG$^+$] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi, Lines in space–combinatorics, algorithms, Tech. Report, Dept. Computer Science, Courant Institute, New York, 1990.

[CG] B. Chazelle and L. J. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica* **1** (1986), 133–162.

[CS] B. Chazelle and M. Sharir, An algorithm for generalized point location and its applications, *J. Symbolic Comput.* **10** (1990), 281–309.

[CSSS] R. Cole, J. Salowe, W. Steiger, and E. Szemerédi, An optimal-time algorithm for slope selection, *SIAM J. Comput.* **18** (1989), 792–810.

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

[EM] H. Edelsbrunner and E. P. Mücke, Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms, *ACM Trans. Graphics* **9**(1) (1990), 66–104.

[GOS] L. J. Guibas, M. H. Overmars and M. Sharir, Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques, Tech. Report RUU-CS-88-26, Dept. Computer Science, University of Utrecht, 1988.

[GS] L. J. Guibas and R. Seidel, Computing convolutions by reciprocal search, *Discrete Comput. Geom.* **2** (1987), 175–193.

[M1] J. Matoušek, Approximations and optimal geometric divide-and-conquer, *Proc. 23rd Ann. ACM Symp. on Theory of Computing* 1991, pp. 505–511.

[M2] J. Matoušek, Efficient partition trees, *Proc. 7th Ann. ACM Symp. on Computational Geometry*, 1991, pp. 1–9.

[MaS] H. Mairson and J. Stolfi, Reporting and counting intersections between two sets of line segments, in *Theoretical Foundations of Computer Graphics and CAD* (R. A. Earnshaw, ed.), NATO ASI Series, Vol. F-40, Springer-Verlag, Berlin, 1988, pp. 307–325.

[MeS] K. Mehlhorn and K. Simon, Intersecting two polyhedra one of which is convex, in *Proc. Fundamentals of Computer Theory*, Lecture Notes in Computer Science, Vol. 199, Springer-Verlag, Berlin, 1985, pp. 534–542.

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

[S] M. Sharir, The shortest watchtower and related problems for polyhedral terrains, *Inform. Process. Lett.* **29** (1988), 265–270.

[VW] V. K. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees and dynamization, *J. Algorithms* 2 (1982), 160–176.