QUASI-OPTIMAL UPPER BOUNDS FOR SIMPLEX RANGE SEARCHING AND NEW ZONE THEOREMS

Bernard Chazelle
Micha Sharir
Emo Welzl

CS-TR-290-90

October 1990

Quasi-Optimal Upper Bounds for Simplex Range Searching and New Zone Theorems

BERNARD CHAZELLE[1] , MICHA SHARIR[2], EMO WELZL[3]

[1] *Department of Computer Science, Princeton University*
[2] *Courant Institute of Mathematical Sciences, New York University and School of Mathematical Sciences, Tel Aviv University*
[3] *Fachbereich Mathematik, Freie Universität Berlin*

Abstract: This paper presents quasi-optimal upper bounds for simplex range searching. The problem is to preprocess a set P of n points in $\Re^d$ so that, given any query simplex q, the points in $P \cap q$ can be counted or reported efficiently. If m units of storage are available ($n < m < n^d$) then we show that it is possible to answer any query in $O(n^{1+\varepsilon}/m^{1/d})$ query time after $O(m^{1+\varepsilon})$ preprocessing. This bound, which holds on a RAM or a pointer machine, is almost tight. We also show how to achieve $O(\log^{d+1} n)$ query time at the expense of $O(n^{d+\varepsilon})$ storage, for any fixed $\varepsilon > 0$. To finetune our results in the reporting case we also establish new zone theorems for arrangements and merged arrangements of planes in 3-space, which are of independent interest.

A preliminary version of this paper has appeared in the Proceedings of the Sixth Annual ACM Symposium on Computational Geometry, June 1990, pp. 23–33.

1. Introduction

We consider the following problem, known as *simplex range searching*: Preprocess a set P of n points in $\Re^d$ so that, given any query simplex q, the points in $P \cap q$ can be counted or reported efficiently. We can put the many variants of this problem under the same umbrella by assuming a weight function on the points and asking for the cumulative weight of the points in $P \cap q$. For the sake of generality, it is best to disallow the use of subtraction (which will make our upper bounds more powerful): formally, this means choosing weights in an additive semigroup.

Our approach is broken up into three stages: We begin by investigating algorithms with polylogarithmic query time and polynomial-size data structures. We show how to achieve $O(\log^{d+1} n)$ query time at the expense of $O(n^{d+\varepsilon})$ storage, for any fixed $\varepsilon > 0$. This result generalizes the two-dimensional solution of Paterson and Yao [27] to higher dimensions at the expense of some extra storage. It is used as a subroutine by the main algorithm, which we discuss in a second stage. This algorithm provides us with a family of tradeoffs in any fixed dimension. We show that with $O(m)$ storage ($n < m < n^d$) it is possible to answer any query in $O(n^{1+\varepsilon}/m^{1/d})$ query time after $O(m^{1+\varepsilon})$ preprocessing. This bound, which holds on a RAM or a pointer machine, is almost optimal: indeed, it comes very close to Chazelle's lower bound of $\Omega((n/\log n)/m^{1/d})$ [4]. It is actually the only tradeoff of this kind for arbitrary dimensions. The two-dimensional case, however, has already been solved almost optimally [1,8,17], as far as query time goes (all of these papers solve the problem only for linear, or close to linear storage). Nevertheless, our preprocessing costs are much smaller than those of previous techniques. Even for the special case where m is linear or quasi-linear, our result improves on previous solutions in dimension greater than 3 (Haussler and Welzl [22], Yao and Yao [31]). Because of the extra n^{ε} factor, however, it falls slightly short (in terms of query time) of the solutions given by Chazelle and Welzl [8]. But on the other hand, except in two and three dimensions, the upper bounds in [8] hold only in the arithmetic model whereas our results hold on any general-purpose computer; that is, they include all costs of searching through the data structures and other auxiliary operations. Moreover, among solutions that give quasi-optimal query time, ours is the only one that is also quasi-optimal in terms of preprocessing cost. Another important advantage of our technique is that it supports multi-level data structures. This allows us to use it for more complex types of queries (see a remark to that effect at the end of Section 3). See also [14,27,30,29] for previous related work.

The extra n^{ε} factor we just mentioned is most likely an undesirable artifact of our methodology. Indeed, we show how to remove this factor in the reporting version of the problem in two and three dimensions. This requires an intricate geometric analysis leading to improved zone theorems for arrangements and merged arrangements of planes in 3-space. Considering the central importance of arrangements in computational geometry, we believe that these theorems are interesting in their own right. Their generalizations to any fixed dimension have been elusive and we leave them as open problems.

A final note concerns the nature of our preprocessing algorithms. Most of them use randomization, but the recent results of Matoušek [23] allow us to derandomize all preprocessing steps. The resulting algorithms have worst-case performance that is asymptotically the same as the expected

cost of the corresponding randomized procedures, but are more complicated to carry out. We therefore prefer to present our technique using the randomized approach; our theorems, though, will state the bounds in a way that indicates that deterministic preprocessing is also possible.

2. Simplex Range Searching in Polylogarithmic Time

We will show how to perform simplex range searching on n points in $\Re^d$ in $O(\log^{d+1} n)$ query time, using a data structure of size $O(n^{d+\varepsilon})$ which can be built in (randomized expected) time $O(n^{d+\varepsilon})$, for any fixed $\varepsilon > 0$. From now on, any reference to the term "expected time", refers to a Las Vegas algorithm, meaning a probabilistic algorithm that is guaranteed always to produce the right solution but whose running time is a random variable. Thus, no distribution on the input needs to be assumed.

We begin with the special case of simplex range searching, where the "simplex" is in fact a closed halfspace. (This is, indeed, a special case of the main problem, since we can trade a halfspace for a very large simplex with one facet in the bounding hyperplane.) The dual version of the problem presents us with a collection of n weighted hyperplanes in d-space: Given a query point, we must compute the added weight of all the hyperplanes lying above the point in question. (For duality, we use the standard point-to-hyperplane and hyperplane-to-point maps: $(p_1, \ldots, p_d) \mapsto x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} + p_d$ and $x_d = p_1 x_1 + \cdots + p_{d-1} x_{d-1} + p_d \mapsto (-p_1, \ldots, -p_{d-1}, p_d)$; the term "above" refers to the x_d-direction.) To solve our range searching problem, we simply precompute the answer at each of the vertices of the arrangement of hyperplanes and preprocess the arrangement for fast point location. The latter can be done in $O(n^{d+\varepsilon})$ expected time, using the $O(n^{d+\varepsilon})$-size data structure of [9] (for any fixed $\varepsilon > 0$). The first part of the preprocessing involves computing the cumulative weight of the planes lying above each vertex of the arrangement. The naive method takes $O(n^{d+1})$ time and can be improved upon by batching computations. We need to make a brief digression to develop this point.

Let $\ell_1, \ldots, \ell_p$ be a sequence of $p \geq n$ lists of at most n numbers each. Assume that each ℓ_i differs from ℓ_{i+1} in at most a constant number of places (meaning that the two lists have bounded edit distance). We wish to compute the sum of the numbers in each list, not allowing subtractions. Here is one way to do it (not necessarily the most efficient). To begin with, we store the first list in, say, a red-black tree and compute partial sums at each node. In this way, summing up any interval in ℓ_i can be performed in $O(\log n)$ time in a straightforward fashion. In particular, we can use this scheme to compute the sum of all the numbers in ℓ_1. To deal with ℓ_2 we insert into or delete from the tree all the numbers in the symmetric difference of ℓ_1 and ℓ_2. Having now obtained an admissible representation for ℓ_2 in $O(\log n)$ time, we can find the sum of all its numbers in a single lookup. Iterating in this fashion, we complete the computation in $O(p \log n)$ time.

Returning to halfspace range searching, we see the utility of our addition scheme by connecting together the vertices of the arrangement in an Eulerian tour of size $p = O(n^d)$ (as in [13]). It is easy to ensure that the hyperplanes lying above any two consecutive vertices in the tour are the same except for at most a constant number of them. We can therefore precompute the cumulative weight of the planes above each vertex in $O(n^d \log n)$ time. In this way, we can answer any halfspace range searching query in $O(\log n)$ time, using $O(n^{d+\varepsilon})$ storage and $O(n^{d+\varepsilon})$ expected preprocessing time.

Now we would like to generalize this data structure to the problem of simplex range searching. The dual version of this problem specifies $d+1$ labeled points $(p_0, \rho_0), \ldots, (p_d, \rho_d)$, where ρ_i stands for "above", "through", or "below". The question is to sum the weights of all the hyperplanes satisfying relation ρ_i with respect to p_i, for each $i = 0, \ldots, d$. Let us ignore the fact that the number of points just happens to be one plus the dimension of the ambient space. Let's call that number k, instead. If $k = 1$, this is a halfspace query and we apply our previous solution. Assume now that $k > 1$.

Let $H(p)$ denote the set of hyperplanes lying above point p. If $v_1, v_2, \ldots$ denote the vertices of the arrangement of n hyperplanes, representing each $H(v_i)$ explicitly would require $O(n^{d+1})$ storage. Using the fact that these sets share large subsets among themselves, we can design a more compact representation by breaking up each $H(v_i)$ into a logarithmic number of "shared" *canonical* subsets. Furthermore, we want to ensure that large canonical sets are few and far between. The reason for this is that canonical sets will be used as inputs to other polynomial-size data structures. Thus, the fewer large canonical sets, the better. Using the theory of random sampling, we know that with high probability a random pick of r hyperplanes has the property that if we triangulate each cell of the arrangement of these hyperplanes, each resulting simplex is crossed by $O(n(\log r)/r)$ hyperplanes (Clarkson [9]). We choose r to be a large constant. For each cell c of the triangulation we compute the set $H(c)$ of hyperplanes that lie strictly above c (and so, in particular, do not intersect the cell). Next, we recurse on this process with respect to each cell and the hyperplanes intersecting it. The resulting data structure can be modeled as a tree of branching degree $O(r^d)$, where a node at depth j is associated with

(i) a certain subset $N(v)$ of the hyperplanes of size $O(n(\log r)^j/r^j)$,

(ii) a certain triangulation of an arrangement of r hyperplanes of $N(v)$, and

(iii) for each cell the set of hyperplanes in $N(v)$ lying completely above it.

Computing the data structure takes expected time $T(n)$, where $T(n) \leq ar^d n + br^d T(cn(\log r)/r)$ and $T(O(1)) = O(1)$, for some appropriate constants a, b, c. (In implementing this recursive partitioning scheme, it is best to verify that each random sample is a good one, in the sense that its triangulated arrangement has only "sparse" cells, as above. If this is not the case, we throw away the sample and try another one. This increases the expected cost by only a constant factor.) It follows that both the expected construction time and the storage requirement are $O(n^{d+\varepsilon})$, for any fixed $\varepsilon > 0$, which means that for any $\varepsilon > 0$ we can choose $r = r(\varepsilon)$ sufficiently large so that $T(n)$ becomes $O(n^{d+\varepsilon})$, with a constant of proportionality depending on ε.

Given a point p, we can compute $H(p)$ by locating p in the triangulation associated with the root of the tree and retrieving the set of hyperplanes lying above the enclosing cell. Next, we pursue the search in the child of the root corresponding to the enclosing cell. Since r is a constant, the entire computation can be performed in $O(\log n)$ time. Note that the sets retrieved form a *partition* of $H(p)$ into a logarithmic number of subsets.

Returning now to our query problem, we will consider each set of hyperplanes stored in some node of the tree and apply the same scheme with respect to the arrangement that it forms. (More precisely, we apply to this arrangement the same scheme with $k-1$ instead of k.) We thus iterate on

this process $k-2$ times; the final sets are processed as in the case $k = 1$. This technique of attaching recursively defined auxiliary structures to the nodes of a tree is a standard staple of multidimensional searching [25], so we feel no need to elaborate any further at this point. Let $T_k(n)$ be the expected time needed to compute the data structure. We have $T_1(n) = O(n^{d+\varepsilon})$ and, for $k > 1$,

$$T_k(n) \leq an^{d+\varepsilon_0} + b \sum_{0 \leq i \leq j} r^{d+di} T_{k-1}(n(\log r)^i / r^i),$$

for some $a, b > 0$ depending on the initial ε_0, where $n(\log r)^j / r^j = O(1)$. We easily check that for any fixed $\varepsilon > 0$, there is an appropriate choice of the constant r such that $T_k(n) \leq cn^{d+\varepsilon}$, for $c = c(\varepsilon) > 0$. Obviously, the same upper bound holds for the space requirement. The query time is clearly $O(\log^k n)$. Let us summarize our findings so far. (We also make a claim on the reporting version of the problem which we leave as an exercise; we also remind the reader that our preprocessing can be derandomized by Matoušek's recent techniques [23].)

Theorem 2.1. *Simplex range searching on n points in $\Re^d$ weighted in a semigroup can be performed in $O(\log^{d+1} n)$ query time, using a data structure of size $O(n^{d+\varepsilon})$, for any fixed $\varepsilon > 0$. Preprocessing takes $O(n^{d+\varepsilon})$ randomized expected or deterministic worst-case time. In the reporting version of the problem, the query time has an extra additive cost linear in the number of points to be reported.*

3. Trading Off Storage and Query Time

Before we begin, it is best to state and prove a technical result which we will use repeatedly. Given n hyperplanes in $\Re^d$ (assumed to be in general position for simplicity), consider the triangulation of their arrangement obtained in the following recursive manner (Clarkson [10]). First, triangulate the $(d-1)$-dimensional arrangements formed by the intersection of each hyperplane with the $n-1$ others. Then, choose the topmost vertex v in each cell (i.e., a d-face) and triangulate the cell by lifting the triangulations of its incident faces (except those incident upon v) centrally toward v. (This method works provided that we use a "generic" coordinate frame in which no face of the arrangement is orthogonal to the x_d-direction.) Any further use of the word triangulation will always refer to that particular type. Constructing such a triangulation takes $O(n^d)$ time. Now, we claim that any hyperplane (not necessarily one of those given to us) intersects $O(n^{d-1})$ cells of the triangulation. Why is that so? We can easily prove by induction that given any cell of the original arrangement, its number of faces before and after triangulation differs by at most a constant factor. Therefore the number of intersected cells in the triangulation is at most proportional to the maximum size of a zone in the untriangulated arrangement, which is known to be $O(n^{d-1})$ [15].

Let us now turn our attention to range searching. As before, we are given as input a collection of n points in $\Re^d$. As we saw in the previous section we can reduce simplex range searching to halfspace range searching by putting together auxiliary structures. We can therefore focus our attention on halfspace range searching. Our basic strategy is to build a triangulation of d-space with the property that no hyperplane can *see* too many of the n input points. (To see a point in this context means

to cut through the cell enclosing it.) Then, given a query halfspace, we can count the points in it by identifying the cells completely within the halfspace and those only partly inside. (Obviously, those totally outside can be ignored.) We count the points within the former cells in time proportional to the number of such cells. To count the points in the other kind of cells, we proceed recursively within each cell separately. Unfortunately, we are unable to find a triangulation that satisfies our requirements. So, instead, we compute a small number of different triangulations with the property any given hyperplane cannot see too many points for at least one of these triangulations. To compute this set of triangulations, we select a representative (random) sample of all possible hyperplanes and compute the desired triangulations with respect to them only. We then argue that what is good for them is also good for any other hyperplane. (This technique is similar to that used by Matoušek in [24].)

Pick a random sample of r points, for some constant r large enough, and construct a triangulation $\mathcal{T}$ of its dual arrangement. With high probability, no cell will be crossed by more than roughly $O(n(\log r)/r)$ of the n dual hyperplanes. We can compute such a triangulation together with the corresponding distribution of hyperplanes among its cells in $O(nr^{d-1})$ (randomized expected) time. To do this, we first compute the triangulation in $O(r^d)$ time and find the cells cut by each dual hyperplane. From the observation we made at the beginning of this section it follows that a standard navigation algorithm will do the work in $O(r^{d-1})$ time per hyperplane. If we discover that a cell is cut by too many hyperplanes, we throw away the random sample and start all over (as we did also in Section 2). With high probability we will succeed within a constant number of attempts.

Next, we dualize each vertex of $\mathcal{T}$ back into primal space. This gives us a set Π of $O(r^d)$ hyperplanes which, as we shall see, provides a good sparse approximation of the $\binom{n}{d}$ hyperplanes defined by the n points. If a hyperplane in Π is dual to a vertex which (because $\mathcal{T}$ is a triangulation of the arrangement) is incident upon many cells then it somehow becomes more "representative" than others. So, for technical reasons, we increase its importance by duplicating it a certain number of times. Specifically, we make Π into a multiset Π^* by making as many copies of a hyperplane as the number of cells to which its dual vertex is incident in $\mathcal{T}$. Obviously, the size N of Π^* is still $O(r^d)$.

Now, we compute the triangulations mentioned in the overview. We pick a random sample of Π^* of size r uniformly and compute a triangulation of the arrangement formed by the sample. (Geometrically, of course, the duplication has no effect; it is just a way of making some hyperplanes more likely to be picked.) Next, we locate each of the n points in the triangulation naively and precompute the number of points enclosed by each cell. This takes $O(nr^d)$ time (we could do it faster but since r is a constant it does not matter). Let us say that a hyperplane *sees* one of these points if it crosses the interior of its enclosing cell (or any one of them, if there are several). With high probability, the interior of each cell will be crossed by $O(N(\log r)/r)$ hyperplanes of Π^*, therefore the total count of visible point/hyperplane pairs is $O(nN(\log r)/r)$. It follows that at least a constant fraction of the hyperplanes of Π^* see only $O(n(\log r)/r)$ points. For technical reasons (which will be clear in a second) we choose this fraction to be $1 - \frac{1}{2(d+1)}$. These *favorable* hyperplanes form a (multi-)subset $\bar{\Pi}^*$ of Π^*. We say that the triangulation is *sparse* for the hyperplanes of $\bar{\Pi}^*$ (in the sense that these hyperplanes see only few of the n points, as just stated). Computing $\bar{\Pi}^*$ takes

$O(Nr^{d-1})$ time by navigating through the triangulation, cutting through it along each of the $\leq N$ hyperplanes of Π.

Let us now say that a cell of $\mathcal{T}$ is *good* if *all* its $d+1$ vertices are dual to hyperplanes in $\bar{\Pi}^*$. If not every cell is good, then we collect all $d+1$ vertices of each of the bad cells, dualize them and form the multiset Π_1^*, to which we then apply the previous procedure over. Note that Π_1^* and $\bar{\Pi}^*$ are usually not disjoint. Also, observe that because of the multiplicity of vertices every bad cell consumes at least one vertex dual to a distinct nonfavorable hyperplane, therefore the number of vertices in the bad cells (which is the size of Π_1^*) is at most $(d+1)N/(2d+2) = N/2$, which is half the size of Π^*. We iterate on this process, computing multisets $\Pi_1^*, \ldots, \Pi_j^*$, until the size Π_j^* drops below r, at which point the "random" sample picks each of them and thus makes them all favorable. This shows that after completion this process will have generated $O(\log r)$ triangulations, each of size r^d. The expected time for computing all these triangulations is $O((nr^d + r^{2d-1}) \log r)$.

We claim that any hyperplane h in $\Re^d$ has at least one sparse triangulation. Note that if that is the case, then by precomputing the number of points in each cell, we can easily find which triangulations are sparse for h in $O(r^{d-1} \log r)$ time, and use one of those in our query. But why should our claim be true? Let p be the dual point of h. Our construction ensures that the cell C of $\mathcal{T}$ that encloses p is good for at least one triangulation S. Therefore, with respect to S, the duals of its $d+1$ vertices can only see $O(n(\log r)/r)$ points. Can the dual h of p somehow see any other of the n points? If not, we are done. If yes, then we can argue that such a point must dualize to a hyperplane that crosses the cell C. There are at most $O(n(\log r)/r)$ such points, so our claim is valid. But how do we argue that a point that is not seen by any of the $d+1$ hyperplanes dual to the vertices of C and yet is seen by h must have its dual hyperplane cross C? Consider these $d+1$ hyperplanes and let D denote the *corridor* enclosed between their upper and lower envelopes. It is easily checked that h must be fully contained in D. Thus, if h sees a point q that none of the $d+1$ hyperplanes can see, q must lie in D. It follows that q lies below one of these hyperplanes and above another one, so its dual hyperplane must separate at least two vertices of C and therefore cross C.

We are now ready to tackle simplex range searching. As we did before, we consider the more general problem of answering queries of the form $(p_1, \rho_1), \ldots, (p_k, \rho_k)$, for any $k = 1, \ldots, d+1$. Let $D_k(P)$ denote the data structure we are seeking when the input consists of a set P of weighted points in $\Re^d$. For consistency we define $D_0(P)$ to be the sum of all the weights in P. The data structure for $D_k(P)$ consists of a tree with various auxiliary structures attached to its nodes. The root of the tree has $O(\log r)$ children (call them subroots), one for each triangulation obtained from the hyperplanes of Π^* as above. We apply the following routine to each of the $O(\log r)$ triangulations. Let $P_1, P_2, \ldots$ be the set of points in each cell of the chosen triangulation. Alongside its associated subroot, we store the data structures $D_{k-1}(P_1), D_{k-1}(P_2)$, etc. Next, we provide the subroot with a distinct child for each P_i, and we recurse in this fashion in each child whose set P_i contains less than $c|P|(\log r)/r$ points (for some c large enough): that is, we attach $D_k(P_i)$ to the child in question. The other children become what we call *fat* leaves. The growth of the tree below a node stops as soon as the point-set associated with it is of size below some magic parameter $\sigma > 0$ (to be determined later). Then we switch to the data structure of Theorem 2.1 (set for the same value of k).

To answer a query $(p_1, \rho_1), \ldots, (p_k, \rho_k)$, first we select a triangulation (at a certain subroot of the tree) that is sparse for the dual hyperplane of p_1, and we compute the cells that lie above (or below, depending on ρ_1) as well as those whose interiors intersect the hyperplane. With respect to the point-sets in the former cells we pursue the search, now involving only $(p_2, \rho_2), \ldots, (p_k, \rho_k)$, in the corresponding D_{k-1}-structures stored at the chosen subroot. For the intersected cells, we simply pursue the search, with our original query $(p_1, \rho_1), \ldots, (p_k, \rho_k)$, in the relevant children of the subroot. Note that the intersected cells contain $O(n(\log r)/r)$ points, therefore there is no risk of ever encountering a fat leaf and getting stuck there. When we reach a (nonfat) leaf of the tree we simply use the algorithm of Theorem 2.1.

What is the complexity of all this, and first of all, does it really work? Let S be the set of points above (or below, depending on ρ_1) the dual hyperplane of p_1. If we just limit ourselves to the master tree (the one containing all the D_{k-1}-structures), it is clear that all the nodes from which we pursue the search into a D_{k-1}-structure or into the type of data structure associated with non-fat leaves, are associated with sets of points which together form a partition of S. Some of these subsets are associated with non-fat leaves and are handled by appealing to Theorem 2.1. The others are dealt with by switching recursively to D_{k-1}-structures. This is admissible since the first condition (ρ_1) is already satisfied. By easy induction on k we can show that each point which satisfies the constraints of the query is accounted for exactly once.

How fast is a query answered? If $t_k(n)$ denotes the worst-case query time, we have $t_1(n) = O(\log n)$, if $n < \sigma$, and otherwise, $t_1(n) = O(r^d) + \sum_i t_1(n_i)$, where the sum is taken over $O(r^{d-1})$ terms (corresponding to the cells in the zone of the query hyperplane) and $\sum_i n_i = O(n(\log r)/r)$. The worst case occurs when all the n_i's are equal, which gives $t_1(n) = O(r^d) + ar^{d-1} t_1(bn(\log r)/r^d)$, for some constants $a, b > 0$. Up to within constant factors, this yields

$$t_1(n) \leq r^{j(d-1)} t\big(n(b(\log r)/r^d)^j\big) + \sum_{0 \leq i < j} r^d r^{(d-1)i}.$$

If ℓ is such that $n(b \log r)^{\ell-1}/r^{d\ell-d} \geq \sigma > n(b \log r)^\ell / r^{d\ell}$, then

$$t_1(n) \leq \sum_{0 \leq i \leq \ell} r^d r^{(d-1)i} \log \sigma.$$

Choosing r to be a constant large enough and assuming that $\sigma < n^{1-\varepsilon_0}$, for arbitrarily small $\varepsilon_0 > 0$, we immediately find that $t_1(n) \leq (n/\sigma)^{(1-1/d)(1+\varepsilon_1)}$ (up to within a constant factor) for any fixed $\varepsilon_1 > 0$. Each internal node of the tree requires $O(r^d)$ storage. One problem, however, is that the input points might be heavily duplicated because of the $O(\log r)$ triangulations generated at each level. Fat leaves were meant to overcome this difficulty. Indeed, this ensures that the total number of points in the data structure, counting duplications, is $O(n(a \log r)^\ell)$, for some constant a, that is, $O(n^{1+\varepsilon_2})$, for any $\varepsilon_2 > 0$. Since r is a constant, it follows from Theorem 2.1 that the storage m needed is at most proportional to $\sum_i n_i^{d+\varepsilon_3}$, for any fixed $\varepsilon_3 > 0$, where $\sum_i n_i = O(n^{1+\varepsilon_2})$ and $n_i < \sigma$. Therefore $m = O(n^{1+\varepsilon_2} \sigma^{d-1+\varepsilon_3})$. Setting $\sigma = \big(m/n^{1+2\varepsilon_2}\big)^{1/(d-1+\varepsilon_3)}$ gives $t_1(n) \leq n^{1+\varepsilon}/m^{1/d}$, for any $\varepsilon > 0$.

If $k > 1$, using conservative estimates for simplicity, the inequality $t_1(n) \leq \sum_{0 \leq i \leq \ell} r^d r^{(d-1)i} \log \sigma$, becomes (up to within constant factors)

$$t_k(n) \leq \sum_{i \geq 0} r^d (\log \sigma)^k \sum_j t_{k-1}(n_j^{(i)}),$$

where for each (level in the master tree) i we have

(i) $t_{k-1}(n_j^{(i)})$ can be replaced by 1 if $n_j^{(i)} < \sigma$,

(ii) $\sum_j n_j^{(i)} = O(n(\log r)^i / r^i)$, and

(iii) the number of $n_j^{(i)}$'s (for fixed i) is $O(r^{(d-1)i})$.

From the previous paragraph, we can assume by induction that (up to within a constant factor) $t_{k-1}(n) \leq (n/\sigma)^{(1-1/d)(1+\varepsilon_1)}$, for any $\sigma < n^{1-\varepsilon_0}$. In that case, we can make all the $n_j^{(i)}$'s equal, for fixed i. The first term of the outer sum ($i = 0$) is the dominant term of a geometric series, from which it follows that $t_k(n) \leq r^d (n/\sigma)^{(1-1/d)(1+\varepsilon_2)}$, for any fixed $\varepsilon_2 > 0$ (up to within a constant factor). We can assume by induction that when given n points as input a D_{k-1}-structure creates $O(n^{1+\varepsilon_3})$ duplications. Since the total number of duplicated points in the master tree, as we already saw, is $O(n^{1+\varepsilon_4})$ for any $\varepsilon_4 > 0$, this gives us (very conservatively) a total count of $O(n^{(1+\varepsilon_3)(1+\varepsilon_4)}) = O(n^{1+\varepsilon_5})$ duplicated points (including all auxiliary structures). Using the same reasoning as before, it follows that the storage m is $O(n^{1+\varepsilon_5} \sigma^{d-1+\varepsilon_6})$. We have the same type of time and space bounds as before (with different ε_i's) for an appropriate assignment of σ, so we conclude that, for any fixed k, $t_k(n) \leq n^{1+\varepsilon}/m^{1/d}$, for any $\varepsilon > 0$. What is the expected preprocessing time? Building the root and subroots of the tree requires $O((nr^{d-1} + r^{2d-1}) \log r)$ time, which is $O(n)$. It follows trivially from Theorem 2.1 that the expected preprocessing time is $O(m^{1+\varepsilon})$.

Returning now to simplex range searching we conclude that given m units of storage it is possible to build a data structure of that size in expected time $O(m^{1+\varepsilon})$, so that any query can be answered in time $O(n^{1+\varepsilon}/m^{1/d})$, for any $\varepsilon > 0$. This result is quasi-optimal, since a lower bound of $\Omega((n/\log n)/m^{1/d})$ was established by Chazelle [4] in the Fredman-Yao arithmetic model.

An interesting application of our algorithm gives an improved solution to Hopcroft's problem: Given n points and n hyperplanes in d-space, find whether there is any contact between points and hyperplanes. We can even solve the on-line version of that problem, where points are given one at a time. Set $m = n^{\frac{2d}{d+1}}$. The preprocessing takes $O(n^{\frac{2d}{d+1}+\varepsilon})$ time and each query can be answered in time $O(n^{\frac{d-1}{d+1}+\varepsilon})$. Thus, the problem can be solved in a total randomized expected time of $O(n^{\frac{2d}{d+1}+\varepsilon})$, for any fixed $\varepsilon > 0$.

We thus summarize (again noting that all our preprocessing steps can be derandomized, if so desired):

Theorem 3.1. *Simplex range searching on n points in $\Re^d$ weighted in a semigroup can be performed in $O(n^{1+\varepsilon}/m^{1/d})$ query time, for any fixed $\varepsilon > 0$, using a data structure of size m (for any m between n and n^d) which can be computed in $O(m^{1+\varepsilon})$ time. (As a consequence, Hopcroft's problem can be solved in $O(n^{\frac{2d}{d+1}+\varepsilon})$ time.) In the reporting version of the problem, the query time has an extra additive cost linear in the number of points to be reported.*

Remark. The partitioning scheme that we use is rather powerful, because it can, in certain applications, be combined with other data structures to form a *multi-level structure* (our structure itself is a multi-level one, but additional levels of other useful structures can be further attached to it). This allows us to extend our approach to more complex types of queries. These ideas will be pursued in a forthcoming paper [2].

4. An Improved Zone Theorem for Planes in Three Dimensions

We now digress for a while from our main theme to develop several results related to arrangements of planes in 3-space. These results will be applied in the next section to enhance the performance of our range searching algorithms in certain cases.

Let $\mathcal{A}(\Pi)$ denote the arrangement formed by a collection Π of n planes $\pi_1, \ldots, \pi_n$ in 3-space. The *combinatorial complexity* of $\mathcal{A}(\Pi)$, namely the total number of its vertices, edges, faces, and cells, is $O(n^3)$. An important concept in arrangements, alluded to earlier, is the notion of a *zone*. Let π be a new plane. The zone of π in $\mathcal{A}(\Pi)$ is the collection of all cells of $\mathcal{A}(\Pi)$ that are crossed by π, and the complexity of the zone is the number of vertices, edges and faces bounding these cells. It is well known that the complexity of a zone in 3-dimensional arrangements is $O(n^2)$ (see [15,19]).

Arrangements of planes are useful structures in many applications [15]. However, for some applications (like the one in this paper), the arrangement itself is too coarse a partitioning of 3-space, mainly because its cells do not necessarily have constant combinatorial complexity. A typical such application (as in the previous sections) involves the technique of ϵ-nets developed by Haussler and Welzl [22], or the related random sampling technique of Clarkson and Shor [9,12]. Triangulating the arrangement using Clarkson's recursive scheme (see the beginning of Section 3) is a useful step in reducing the complexity of the cells. Sometimes, however, we wish to merge (superimpose) two or several arrangements. It is then less than clear what happens to the complexity of the resulting cell complex. Of course, it might no longer be a triangulation, but this can be easily fixed. A more serious problem is to determine whether the superimposed complex still has cubic complexity and whether the traditional zone theorems [15] continue to hold. To simplify this task we build a *vertical decomposition* of the arrangement. This is done by decomposing each cell c of $\mathcal{A}$ into vertical prisms, by drawing a vertical "wall" from each edge of c through c until it meets the boundary of c again. This produces a collection of vertical prisms which are then further decomposed into subprisms of constant complexity by triangulating their bases (see below for more details).

For two-dimensional arrangements, vertical decompositions do not introduce significant technical difficulties, mainly because, as implied by Euler's formula, they do not increase the complexity of the arrangement by more than a constant factor. More strongly, they do not increase the complexity of any single cell by more than a constant factor. Thus the overall complexity of the arrangement continues to be quadratic, and the number of trapezoidal subcells crossed by a line remain linear. In three dimensions, however, the situation is considerably more complicated. Let us denote the vertical decomposition of the arrangement of our original collection Π by $\mathcal{C}(\Pi)$. It is known (and will also be argued below) that the overall complexity of $\mathcal{C}$ is still $\Theta(n^3)$, but the proof is no longer trivial. (To appreciate the difficulty, note that it is possible for a single cell c of the original arrangement, which

has only $\Theta(n)$ complexity, to be decomposed into $\Theta(n^2)$ subcells in the vertical decomposition.) See also [3,5,7,11] for the study of vertical decompositions of arrangements of triangles, polyhedra, and curved surfaces. We can extend the notion of a zone of a plane π to vertically decomposed arrangements $\mathcal{C}$ by defining it as the subcollection of cells of $\mathcal{C}$ crossed by π. We will show that the complexity of any zone is $O(n^2 \log n)$, thus only slightly degrading the bound on standard zone complexity.

We then apply this result to derive a nearly cubic bound on the complexity of the space decomposition obtained by superimposing two vertically–decomposed arrangements of n planes each. Notice that since each cell in the original vertical decompositions has constant complexity, the same holds for the cells in the superimposed partitioning. Actually, we first establish a stronger property—we show that the complexity of the zone of a plane in the superimposed partitioning is $O(n^2 \log^3 n)$. Since each feature of the superimposed partitioning can be "charged" to the zone of one of the $2n$ planes, this implies an $O(n^3 \log^3 n)$ bound on the overall complexity of the superposition. We note however that for this result to hold we need to exercise some care in the manner in which we construct the vertical decompositions: as it turns out, not every vertical decomposition is suitable for this purpose.

4.1. Vertical Decomposition of an Arrangement of Planes

Let $\Pi = \{\pi_1, \ldots, \pi_n\}$ be a collection of n planes in 3-dimensional space. Let $\mathcal{A} = \mathcal{A}(\Pi)$ denote their arrangement, namely the cell decomposition of space induced by these planes (see the introduction and [15] for basic definitions and properties of arrangements). Each 3–D cell of $\mathcal{A}$ is a convex polyhedron whose combinatorial complexity (namely the number of faces, edges, and vertices along its boundary) can be as large as $O(n)$. The total number of cells in $\mathcal{A}$, as well as their overall combinatorial complexity, is $O(n^3)$ (it is actually $\Theta(n^3)$ unless the planes are in degenerate position). We now formalize the notion of a vertical decomposition. Variants of this method are described in [3] for arrangements of triangles, in [7] for simple nonconvex polytopes, in [11] for arrangements of spheres, and in [5] for arrangements of arbitrary algebraic surfaces (including also higher–dimensional arrangements). In the case of planes, this triangulation further partitions each cell c of $\mathcal{A}$ in the following two–step manner:

(i) From each edge e of c, lying, say, on the bottom portion of the boundary, draw a vertical strip from e upwards until it meets the top boundary of c. Similarly, draw vertical strips from each edge on the top boundary of c downwards. These "walls" partition c into a collection of vertical prisms, each bounded from above and from below by a single face (which is a portion of some face of c). Let $\mathcal{C}_0(\Pi)$ denote the resulting decomposition of $\mathcal{A}$, when this procedure is applied to all cells of $\mathcal{A}$.

(ii) Next, project each subcell c of $\mathcal{C}_0(\Pi)$ onto the xy–plane, to obtain a convex polygon c^*. Decompose c^* into subpolygons of constant complexity, whose number is proportional to the combinatorial complexity of c^*. This can be done in many ways, but we choose the following specific manner, which results in a simplified proof of the improved zone theorem and is needed in the analysis of the merging of two triangulated arrangements. Suppose c^* is a k–gon $v_0 v_1 \ldots v_{k-1}$

obtained as the xy-projection of some cell of $\mathcal{C}_0(\Pi)$. We triangulate $c^\star$ recursively by drawing the chords v_0v_2, v_2v_4, v_4v_6, ... removing the resulting triangles $v_0v_1v_2$, $v_2v_3v_4$, ..., and repeating this step on the resulting polygon. We now take each resulting triangle and intersect the infinite vertical prism based on it with c. This yields a decomposition of c into prisms. Applying this decomposition to each cell of $\mathcal{C}_0(\Pi)$, we obtain our final collection of subcells decomposing $\mathcal{A}$, which we denote by $\mathcal{C}(\Pi)$. It is easily checked that any line in the xy–plane cuts only $O(\log k)$ subtriangles of a projected k–gonal cell c^*.

We will refer to $\mathcal{C}(\Pi)$ as a *vertical decomposition* of $\mathcal{A}$. Adapting previous analysis techniques [11,26], we obtain

Lemma 4.1. *$\mathcal{C}(\Pi)$ consists of $O(n^3)$ cells.*

Proof: The size of $\mathcal{C}(\Pi)$ is easily seen to be bounded by the total combinatorial complexity of all cells in $\mathcal{C}_0(\Pi)$ (regardless of which triangulation we use in step (ii)). The latter quantity is proportional to the original complexity of $\mathcal{A}$ (namely $O(n^3)$) plus the number of vertical edges of these cells. Let $e = pq$ be such an edge. Then either (i) p is a vertex of $\mathcal{A}$; or (ii) q is a vertex of $\mathcal{A}$; or (iii) there exist two edges, e_1, e_2, of $\mathcal{A}$ such that $p \in e_1$, $q \in e_2$, and no other plane of Π crosses e. Since each vertex of $\mathcal{A}$ induces only two vertical edges of type (i) or (ii), the total number of such edges is $O(n^3)$.

To estimate the number of edges of type (iii), consider a fixed intersection line, ℓ, of a pair of planes in Π. Let v be the vertical plane passing through ℓ. Each vertical edge of type (iii) whose bottom endpoint lies on ℓ corresponds to a "breakpoint" in the lower envelope of the arrangement of $n-2$ rays in v, each of which is the intersection of a plane of Π (other than the two intersecting at ℓ) with the half–plane of v lying above ℓ. Since the lower envelope of $n-2$ rays has at most $O(n)$ breakpoints, it follows that the number of vertical edges of type (iii) whose bottom endpoint lies on ℓ is $O(n)$. Using a symmetric argument for the edges whose top endpoint lies on ℓ, and repeating the analysis for all $O(n^2)$ intersection lines of pairs of planes in Π, we complete the proof of the theorem. □

Remark. Another way to prove the preceding Theorem is to notice that the overall complexity of all cells of $\mathcal{C}_0(\Pi)$ contained in a single cell c of $\mathcal{A}(\Pi)$ is $O(|c|^2)$, where $|c|$ denotes the combinatorial complexity of c. As shown in [15] (see Theorem 5.5 there), we have $\sum_c |c|^2 = O(n^3)$, where the summation extends over all cells of $\mathcal{A}$. The theorem is an immediate consequence of this.

4.2. An Improved Zone Theorem

Let π be a new plane and let Z_π denote the *zone* of π in $\mathcal{A}$, i.e., the collection of all cells of $\mathcal{A}$ crossed by π. As is well known (see [19,15]), the overall combinatorial complexity of these cells is $O(n^2)$. However, our goal in this paper is to bound the number of cells of $\mathcal{C}(\Pi)$ crossed by π, or, more generally, that are contained in the zone Z_π. We were able to obtain a close to quadratic bound on the complexity of the latter kind. However the bound is slightly worse than that for the former type of zone, the analysis is much more complicated, and the result is not needed anyway in our application. Nevertheless, we state the result without proof for the interested reader:

The number of subcells of $\mathcal{C}(\Pi)$ that are contained in the standard zone in $\mathcal{A}$ of a plane π is $O(n^2 \cdot 2^{2\sqrt{2}\sqrt{\log n}})$.

Having said that much, we now concentrate only on the subcells that are actually crossed by π. This leads to a much simpler analysis, and it suffices for the range searching application that will be considered later. To distinguish between the two problems, we refer to the collection of cells of $\mathcal{C}(\Pi)$ crossed by π as the *crossing zone* of π. Even with this restriction, the number of such cells appears to be much larger than the complexity of the zone of π in $\mathcal{A}$ (a single cell in $\mathcal{A}$ can be split into $\Theta(n^2)$ subcells in $\mathcal{C}$), but we will show that, asymptotically, the complexity does not grow that bad. Our proof is somewhat similar to the technique used in [16,18,26] to analyze the complexity of envelopes of triangles in 3-space.

Theorem 4.2 (New Zone Theorem). *The number of cells of $\mathcal{C}(\Pi)$ crossed by a plane π is $O(n^2 \log n)$.*

Proof: To bound this number, we will first estimate another quantity, which we denote by $\zeta = \zeta(\pi)$ and define to be the number of cells of $\mathcal{C}_0(\Pi)$ crossed by π. Combining this bound with arguments based on planarity will yield a bound on the former quantity.

To prove the lemma, we first extend it as follows. Given the crossing plane π, we split each plane $\pi_i \in \Pi$ into two halfplanes at the intersection line $\ell_i = \pi_i \cap \pi$, We thus obtain a collection $\mathcal{U}$ of n upper halfplanes and a collection $\mathcal{L}$ of n lower halfplanes, Now suppose that $\mathcal{U}$ and $\mathcal{L}$ are unrelated—that is, all halfplanes still "terminate" on π, but the delimiting lines of the halfplanes in $\mathcal{U}$ do not necessarily coincide with delimiting lines of matching halfplanes of $\mathcal{L}$. We take the lower envelope of the halfplanes of $\mathcal{U}$ and the upper envelope of the halfplanes of $\mathcal{L}$ and project both envelopes vertically onto π. This gives us two convex subdivisions, $\mathcal{M}_{\mathcal{L}}$, $\mathcal{M}_{\mathcal{U}}$, of π which we superimpose to obtain another convex subdivision $\mathcal{M}$. If $\mathcal{U}$ and $\mathcal{L}$ are the collections of halfplanes produced from π as above, then it is easily checked that the number of faces of $\mathcal{M}$ is equal to the number of cells of $\mathcal{C}_0(\Pi)$ crossed by π.

Lemma 4.3. *Given any two collections of n upper halfplanes and of n lower halfplanes respectively, which satisfy the above conditions, the complexity of the resulting planar map $\mathcal{M}$ is $O(n^2 \log n)$.*

Proof: Partition $\mathcal{U}$ into two subsets $\mathcal{U}_1$, $\mathcal{U}_2$ of roughly equal size, and partition $\mathcal{L}$ similarly into two subsets $\mathcal{L}_1$, $\mathcal{L}_2$. Let $\mathcal{M}_{ij}$ denote the map obtained by merging $\mathcal{M}_{\mathcal{U}_i}$ with $\mathcal{M}_{\mathcal{L}_j}$, for $i, j = 1, 2$. Each face f of $\mathcal{M}_{ij}$ is a maximal connected region on π with the property that each point $p \in f$ sees the same halfplane in $\mathcal{U}_i$ directly above it and the same halfplane in $\mathcal{L}_j$ directly below it; here we ignore the presence of the halfplanes in $\mathcal{U}_{3-i}$, $\mathcal{L}_{3-j}$. We fix a pair $\mathcal{U}_i$, $\mathcal{L}_j$, and refer to the halfplanes in these subcollections as red (upper and lower) halfplanes, and to the halfplanes in $\mathcal{U}_{3-i}$, $\mathcal{L}_{3-j}$ as blue halfplanes. We now add the blue halfplanes back to the arrangement one at a time, and bound the *increase* in the number of (maximal connected) regions on π, each consisting of points that see the same red plane above them and another red plane below (in the presence of the already added blue halfplanes); we refer to such regions as "red–red" faces. Repeating this process four times for all possible combinations of $\mathcal{U}_i$, $\mathcal{L}_j$ will give a recurrence on the size of $\mathcal{M}$.

Suppose we have already added some number of blue halfplanes, and let f be a red–red face in the current map $\mathcal{M}'$. Thus there is a pair $\pi_1 \in \mathcal{U}_i$, $\pi_2 \in \mathcal{L}_j$ of red halfplanes such that f sees π_1 above it and π_2 below. When the next blue halfplane π' is added it either leaves f undisturbed, or "chops off" a portion of f but still leaves only one connected portion of f where π_1 and π_2 are still visible, or cuts f into two such portions. Thus an increase in the number of red–red faces can be obtained only in the third case. But it is easily checked that this case can arise only if the line ℓ' delimiting π' (on π) cuts f. But the number of faces of $\mathcal{M}'$ crossed by ℓ' is only $O(n)$. Indeed, consider the vertical plane V containing ℓ'. The red halfplanes and the blue halfplanes added so far cut V in $\leq 2n$ rays, all emerging from points on ℓ'. Each face f crossed by ℓ' marks on ℓ' a maximal interval where the lower envelope of the upper rays and the upper envelope of the lower rays are both attained by two fixed rays; as is well known, the number of such intervals is $O(n)$. Hence the total increase in the number of red–red faces caused by adding the blue halfplanes is $O(n^2)$.

We thus obtain the following recurrence relationship for the maximum number of faces, $F(n)$, in a planar map $\mathcal{M}$ obtained as above for collections of $2n$ halfplanes:

$$F(n) \leq 4F(n/2) + O(n^2)$$

whose solution is $O(n^2 \log n)$. □

Remark: Can this technique be modified to yield a similar bound on the complexity of a zone in an arrangement of n triangles in space?

Returning to the proof of Theorem 4.2, we conclude from Lemma 4.3 that the number of cells of $\mathcal{C}_0(\Pi)$ crossed by π is $O(n^2 \log n)$. To obtain the number of cells of $\mathcal{C}(\Pi)$ crossed by π, consider the collection of cells of $\mathcal{C}_0(\Pi)$ crossed by π. Each vertical edge of such a cell is either crossed by π, in which case it corresponds to a vertex of the map $\mathcal{M}$, or else is (vertically) hidden from π by another plane.

Let c be such a cell and suppose it has k vertical edges. The number of subcells in $\mathcal{C}(\Pi)$ into which c is decomposed is $O(k)$. If such a subcell c' is crossed by π and π cuts one of its three vertical edges, then this edge corresponds to a vertex of the map $\mathcal{M}$. Otherwise π cuts the top and bottom faces of c' and their edges. If it cuts at least one edge of $\mathcal{C}_0(\Pi)$ that bound these faces of c', then the intersection is again a vertex of $\mathcal{M}$. Otherwise π cuts only "secondary" vertical walls (i.e., those based on diagonals added to the xy-projection of c) on the top and bottom faces of c'. In this case, if, say the top face of c' lies on a plane π' then the line $\ell' = \pi \cap \pi'$ also cuts the secondary wall. But since ℓ' intersects the boundaries of cells in $\mathcal{C}_0(\Pi)$ at $O(n)$ points, it follows from the second decomposition step that ℓ' can intersect only $O(n \log n)$ secondary walls. Repeating this argument over all planes π', and observing that the overall complexity of $\mathcal{M}$, being a planar convex subdivision with $O(n^2 \log n)$ faces, is also $O(n^2 \log n)$, we conclude that the number of cells of $\mathcal{C}(\Pi)$ crossed by π is $O(n^2 \log n)$. □

Remark: The factor $\log n$ that appears in our bound is an artifact of our divide–and–conquer technique. We do not know whether the actual complexity of a crossing zone is quadratic or can be super-quadratic.

4.3. Merging Two Triangulated Arrangements

Let Π_1, Π_2 be two collections of n "blue" planes and n "red" planes, respectively. Suppose we construct $\mathcal{C}(\Pi_1)$ and $\mathcal{C}(\Pi_2)$ and then superimpose them. We obtain a convex subdivision of 3–space, which we will denote by $\mathcal{C}_{12}$, each cell of which is an intersection of a cell of $\mathcal{C}(\Pi_1)$ with a cell of $\mathcal{C}(\Pi_2)$, and is thus of constant combinatorial complexity. (If desired, we can split each such cell into $O(1)$ subcells having the same structure as the cells of $\mathcal{C}(\Pi_1)$ and of $\mathcal{C}(\Pi_2)$.) Our goal is to derive a sharp upper bound on the combinatorial complexity of the superimposed subdivision $\mathcal{C}_{12}$. Unfortunately, unless we use the special type of triangulation to form $\mathcal{C}(\Pi_1)$ and $\mathcal{C}(\Pi_2)$ as above, $\mathcal{C}_{12}$ may have $\Omega(n^4)$ cells. We digress to describe such an example.

Example: We next demonstrate that Theorem 4.5 below may fail miserably if we use the "vertical" triangulation of Section 2, in the sense that the superposition of two triangulated arrangements $\mathcal{C}(\Pi_1)$, $\mathcal{C}(\Pi_2)$, of two collections of n planes each, can have $\Omega(n^4)$ cells. The first collection Π_1 in our example consists of the xy plane, of $n/2-1$ planes parallel to the x–axis, whose equations are

$$y\cos\frac{2\pi j}{n} - z\sin\frac{2\pi j}{n} = 1,$$

for $j = 1,\ldots,n/2$, and of $n/2$ additional planes parallel to the plane $x+z=1$ and sufficiently separated from one another. The first $n/2$ planes all bound an $(n/2)$–gonal prism P whose yz cross section is the bottom half of a regular n–gon, while the other $n/2$ planes cut P in slanted parallel polygonal sections. These planes are sufficiently separated from one another so that the xy–projections of these slanted cuts are pairwise disjoint. A top view of this construction is given in Figure 1.

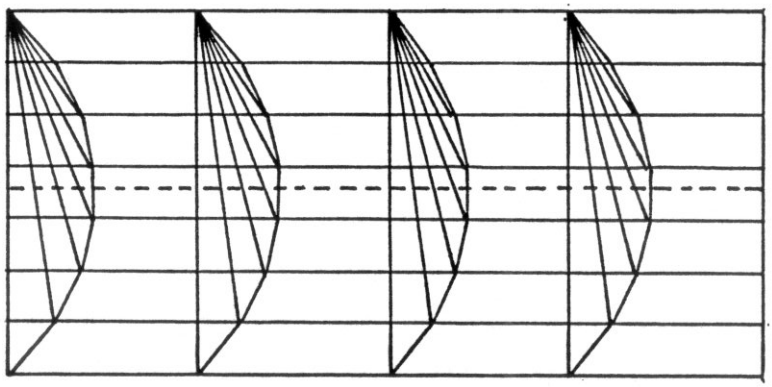

Figure 1

A top view of the construction of a "bad" superposition
of two vertically decomposed arrangements

Note that if $\mathcal{C}(\Pi_1)$ is formed using the vertical triangulation of Section 2, any line parallel to the x–axis and lying sufficiently close to and below it, cuts $\Theta(n^2)$ cells of $\mathcal{C}(\Pi_1)$; this is illustrated in Figure 1.

The second collection Π_2 consists of n planes parallel to the x–axis and sufficiently close to it, with the property that the yz–cross section of $\mathcal{A}(\Pi_2)$ has $\Theta(n^2)$ vertices in a sufficiently small neighborhood of the origin. Thus there are $\Theta(n^2)$ intersection lines of the planes in Π_2, each of which cuts $\Theta(n^2)$ cells of $\mathcal{C}(\Pi_1)$, implying that the superposition of $\mathcal{C}(\Pi_1)$ and $\mathcal{C}(\Pi_2)$ has $\Theta(n^4)$ cells, which establishes our claim on the unsuitability of vertical decompositions for merging. □

Recall that our second decomposition step is such that any line in the xy–plane cuts only $O(\log k)$ subtriangles of a projected k–gonal top or bottom face of a cell of $\mathcal{C}_0(\Pi_1)$ or $\mathcal{C}_0(\Pi_2)$. Consequently, if a line ℓ in 3–space cuts a cell c of $\mathcal{C}_0(\Pi_1)$ then ℓ cuts at most $O(\log k)$ cells of $\mathcal{C}(\Pi_1)$ contained in c, where k is the combinatorial complexity of c; a similar property holds for $\mathcal{C}(\Pi_2)$. Note that this property fails for other vertical triangulations (it is this failure that causes the generation of $\Omega(n^4)$ subcells in the above example).

Lemma 4.4. *The number of cells in $\mathcal{C}_{12}$ crossed by any plane π is $O(n^2 \log^3 n)$.*

Proof: Let Z' denote the collection of all cells of $\mathcal{C}_{12}$ crossed by π, and let $\mathcal{A}'$ denote the subdivision of π formed by its intersections with the boundaries of the cells in Z'. Note that the complexity of Z' is bounded by the complexity of $\mathcal{A}'$. Consider the intersection of π with $\mathcal{C}_0(\Pi_1)$ and with $\mathcal{C}_0(\Pi_2)$. These are two convex subdivisions of π and we wish to bound the complexity of the subdivision $\mathcal{A}_0$ obtained by overlaying them on top of each other.

We claim that this bound, when multiplied by $O(\log^2 n)$, yields a bound on the complexity of Z'. Indeed, what is missing in $\mathcal{A}_0$ are intersections of π with the "secondary" vertical walls erected during the second vertical decomposition step for both Π_1 and Π_2. Call these intersections secondary edges. Notice first that each edge of $\mathcal{A}_0$ intersects at most $O(\log n)$ secondary edges, because it is fully contained in the cross section of a single cell of $\mathcal{C}_0(\Pi_1)$ and a single cell of $\mathcal{C}_0(\Pi_2)$. Hence if $\mathcal{A}_0$ has k edges then the number of intersections between its edges and secondary edges is $O(k \log n)$. These points thus partition the secondary edges into $O(k \log n)$ subedges in total, and, by the same argument as above, each subedge can intersect at most $O(\log n)$ other secondary edges. This establishes our claim.

We can therefore focus our attention on $\mathcal{A}_0$ alone. It suffices to obtain a bound for the number of vertices of $\mathcal{A}_0$. Each such vertex is either the intersection of π with an edge of $\mathcal{C}_0(\Pi_1)$, or with an edge of $\mathcal{C}_0(\Pi_2)$, or an intersection of π with a face of one decomposition and with a face of the other. Since Z' is contained in the zones of π in both subdivisions, and since each of these zones has only $O(n^2 \log n)$ edges, it suffices to consider only vertices z of the latter type. The following cases can arise:

(i) One of the faces inducing z lies on one of the original planes. Suppose, without loss of generality, that z lies on a plane $\pi_1 \in \Pi_1$. Then z lies on the line $\ell_1 = \pi \cap \pi_1$. If the other face inducing z also lies on a plane of Π_2 then z is the intersection of ℓ_1 with that plane, and there can be only n such points. Otherwise, z is the intersection of ℓ_1 with some vertical face f of $\mathcal{C}_0(\Pi_2)$. It follows that there is a real edge of $\mathcal{A}(\Pi_2)$ directly above or directly below z. Arguing as in the proof of Lemma 4.1, the number of such "breakpoints" z along ℓ_1 is only $O(n)$. Summing over all planes π_1, we conclude that there are only $O(n^2)$ vertices z of this type.

(ii) The two faces defining z are vertical faces in the subdivisions $\mathcal{C}_0(\Pi_1)$ and $\mathcal{C}_0(\Pi_2)$, respectively. Again, there exist a real edge of $\mathcal{A}(\Pi_1)$ and a real edge of $\mathcal{A}(\Pi_2)$, each lying either directly above or directly below z. Without loss of generality assume that an edge of $\mathcal{A}(\Pi_1)$ lies above z and an edge of $\mathcal{A}(\Pi_2)$ lies below z. Clip each plane of Π_1 to the half plane lying above π, and each plane of Π_2 to the half plane lying below π. It follows that the problem has been reduced to that of estimating the combinatorial complexity of the "crossing zone" of π in the vertical decomposition $\mathcal{C}_0(\Pi^*)$, where Π^* is a collection of $2n$ halfplanes, n of which lie above π and the other n lie below π, and all halfplanes "terminate" on π itself; here $\mathcal{C}_0(\Pi^*)$ is defined in complete analogy to the manner in which it was defined above for full planes. But this is exactly the setup of Lemma 4.3, which therefore implies that the complexity of this crossing zone is $O(n^2 \log n)$.

As argued above, this completes the proof of Lemma 4.4. □

Theorem 4.5. *The number of cells in $\mathcal{C}_{12}$ is $O(n^3 \log^3 n)$.*

Proof: Since vertical faces and edges of $\mathcal{C}(\Pi_1)$ and of $\mathcal{C}(\Pi_2)$ cannot cross transversally, it follows that each vertex v of $\mathcal{C}_{12}$ must lie on a plane of Π_1 or of Π_2. Consider, without loss of generality, only vertices that lie on a plane π of Π_1. But, by definition, these vertices are features of the crossing zone of π in $\mathcal{C}_{12}$, so their number is $O(n^2 \log^3 n)$ from the previous lemma. Summing this over all planes π yields the asserted bound. □

We have the following easy generalization of Lemma 4.4:

Theorem 4.6. *If we merge q vertical decompositions $\mathcal{C}(\Pi_i)$, for $i = 1, \ldots, q$, each involving n planes, the number of cells of the merged arrangement that are crossed by a plane is $O(n^2 q^2 \log^3 n)$.*

Remark: Note that the cells of the merged arrangement in the last theorem are not necessarily of constant complexity—each can be bounded by $O(q)$ faces. We can further decompose such a cell into $O(q)$ subcells of constant complexity each. In this case, although the overall complexity of the decomposition does not increase by more than a constant factor, the number of cells crossed by a given plane is $O(n^2 q^3 \log^3 n)$. We have the following generalization of Theorem 4.5.

Corollary 4.7. *The number of constant-complexity subcells that arise as we merge q vertically decomposed subarrangements of n planes each, is $O(n^3 q^3 \log^3 n)$.*

5. Simplex Range Reporting in Two and Three Dimensions

The zone theorems of the previous section allow us to sharpen somewhat our complexity bounds for the reporting version of simplex range searching in two and three dimensions. We confine most of our discussion to the three-dimensional case, which contains all the important ideas.

The construction of $O(\log r)$ sparse triangulations, as described in Section 3, can be easily adapted to the particular type of vertical decompositions discussed here. The only differences arise in analyzing the complexity of the construction. Given a collection Π of r planes as input, the vertical decomposition $C(\Pi)$ is obtained by first computing the arrangement $\mathcal{A}(\Pi)$, which takes $O(r^3)$ time [15,19], then erecting the vertical walls, which takes time proportional to the sum of the squares of the cell sizes, and finally triangulating the xy-projections of each resulting cell, which takes time linear in the size of the decomposition. As we already observed in Section 4.1 the sum of the squares of the cell sizes is $O(r^3)$ [15], therefore the total construction time is $O(r^3)$.

Let us now briefly review the steps in the construction of the sparse triangulations. The triangulation $\mathcal{T}$ need not be replaced by a vertical decomposition, therefore the complexity of computing Π^* is still $O(nr^2)$. Next we pick a random sample of r planes in Π^* and compute its vertical decomposition. To locate each of the n points inside it, it suffices to intersect the decomposition by a plane passing through each point and locate the point in question naively among the planar cells of the intersection. By virtue of Theorem 4.2, this reduces the complexity of the search to $O(r^2 \log r)$ per point. Similarly, computing how many points a given plane of Π^* can see takes $O(r^2 \log r)$ time. Since we have $O(\log r)$ stages, the overall preprocessing time is $O(nr^2 \log^2 r + r^5 \log^2 r)$. A similar proof to the one given in Section 3 shows that, given any plane π, for at least one of the $O(\log r)$ vertical decompositions, π can see only $O(n(\log r)/r)$ points.

Our strategy now is to avoid the duplication of input points by merging together the $O(\log r)$ vertical decompositions. How do we do that? One solution is first to intersect each plane of each decomposition (there are $O(r \log r)$ such planes) with all the other decompositions. Using a standard navigational scheme, each intersection operation can be performed in time proportional to the number of cells crossed by the plane, which by Theorem 4.2 is $O(r^2 \log r)$. This produces $O(\log r)$ convex subdivisions of the plane in question which we must now merge together. Using, say, Guibas and Seidel's algorithm [21] we can merge two planar convex subdivisions in time linear in the input and output sizes. Because of Theorem 4.6 we know that the final subdivision will have size $O(r^2 \log^5 r)$. In the worst case the same vertices of the subdivision might be looked at during successive merges, so the running time is $O(r^2 \log^6 r)$. Since there are $O(r \log r)$ planes to consider, the total running time is $O(r^3 \log^7 r)$ and the storage requirement is $O(r^3 \log^6 r)$. This gives us the adjacency information along each of the $O(r \log r)$ planes. To compute the vertical edges we may have to sort the vertices along the vertical lines to which they belong. Since we already know those vertical lines (intersections of vertical walls), all vertical adjacencies, and finally a full representation of the merged decomposition, can be obtained in $O(r^3 \log^7 r)$ time and $O(r^3 \log^6 r)$ space. As noted in Corollary 4.7 we can further decompose each cell into constant-complexity subcells in one last pass of cost linear in the size of the decomposition.

To summarize, in $O(nr^2 \log^2 r + r^5 \log^2 r)$ time we have computed a three-dimensional decomposition $\mathcal{C}$ consisting of $O(r^3 \log^6 r)$ constant-complexity cells, such that any given plane intersects $O(r^2 \log^6 r)$ cells. From our discussion of Section 3, we know that (by sampling enough times) we can guarantee that any given plane sees only $O(n(\log r)/r)$ points with respect to at least one of the original $O(\log r)$ decompositions. The idea of merging the decompositions together is that now this property is true for $\mathcal{C}$ with respect to *any* plane.

We complete the data structure by considering each nonempty cell in turn and arbitrarily breaking up the points inside into groups, each consisting of roughly σ points. Each group of points is preprocessed according to Theorem 2.1. Also, alongside each nonempty cell of the decomposition we store the points that it encloses. Finally, we pick an arbitrary sample point in each nonempty cell and we build the same data structure recursively with respect to these sample points. This last step might sound peculiar at first but it will afford us a useful bootstrapping mechanism. The (randomized expected) preprocessing time amounts to $O(nr^2 \log^2 r + r^5 \log^2 r)$ (for constructing $\mathcal{C}$), $O(nr^2 \log^6 r)$ for locating the points inside $\mathcal{C}$ (by traversing $\mathcal{C}$ along planes containing the points) and $O((r^3 \log^6 r + n/\sigma)\sigma^{3+\varepsilon})$ (for preprocessing the groups of points). Note that by choosing $r^3 \log^6 r$ and n/σ smaller than n by at least a constant factor, we make the recursion costs follow a geometric series and hence be negligible. Therefore the total preprocessing time is on the order of

$$nr^2 \log^6 r + r^5 \log^2 r + (r^3 \log^6 r + n/\sigma)\sigma^{3+\varepsilon}.$$

The storage requirement is at most proportional to

$$(r^3 \log^6 r + n/\sigma)\sigma^{3+\varepsilon}.$$

To answer a query, we begin by identifying which cells intersect the bounding facets of the query tetrahedron (type A) and which ones fall completely inside (type B). This is done as follows. First, we apply the algorithm recursively with respect to the sample points and weed out from the answer those reported sample points whose enclosing cells are of type A. From the remaining sample points reported we easily extract the actual input points enclosed in their respective cells. This allows us to find readily which of the n points lie inside the cells of type B.

Now, we are ready to identify the cells of type A. To do so we intersect $\mathcal{C}$ with each of the four planes bounding the query tetrahedron and we check all the intersected cells. All type-A cells will be encountered in this fashion. The time to do this can be kept proportional to the number of intersected cells which we know from our previous remark to be $O(r^2 \log^6 r)$. Finally we handle all type-A cells by using the precomputed data structures associated with the groups of points in them. Because a plane can see only $O(n(\log r)/r)$ points in $\mathcal{C}$, the query time overhead $t(n)$ (i.e., not counting the time to report the points) satisfies the recurrence $t(O(1)) = O(1)$ and

$$t(n) \leq t(br^3 \log^6 r) + c\left(r^2 \log^6 r + \frac{n \log r}{r\sigma}\right) \log^4 \sigma,$$

for some constants $b, c > 0$.

Recall that m denotes the amount of storage available. If we set $\sigma = \alpha(m/n)^{\frac{1}{2+\varepsilon}}$, where α is a small enough constant, and define r by the equation $\sigma = n/(r^3 \log^6 r)$, we can verify that as long as m/n is large enough and $m < n^3$, both $r^3 \log^6 r$ and n/σ are sufficiently smaller than n, as required earlier. The storage requirement, which is on the order of $(r^3 \log^6 r + n/\sigma)\sigma^{3+\varepsilon}$, is $O(n\sigma^{2+\varepsilon})$ and therefore does not exceed the amount m allowed (for α small enough). The preprocessing time, which is on the order of $nr^2 \log^6 r + r^5 \log^2 r + (r^3 \log^6 r + n/\sigma)\sigma^{3+\varepsilon}$, becomes (conservatively) $O(n\sigma^{2+\varepsilon} + n(n/\sigma)^{2/3} \log^2 r + (n/\sigma)^{5/3})$, which is at most proportional to

$$m + \frac{n^{5/3+2/(6+3\varepsilon)} \log^2 n}{m^{2/(6+3\varepsilon)}} + \frac{n^{5/3+5/(6+3\varepsilon)}}{m^{5/(6+3\varepsilon)}},$$

that is, $O(m+n^{2+\varepsilon}/m^{1/3})$. Finally, since $br^3\log^6 r$ is at most a fraction of n (for m/n large enough), the recurrence for $t(n)$ will be applied only $O(\log n)$ times, showing that

$$\frac{n(\log r)(\log\sigma)^4\log n}{r\sigma} = (n/\sigma)^{2/3}(\log r)^3(\log\sigma)^4\log n = (n/\sigma)^{2/3}(\log n)^8,$$

which is also on the order of

$$n^{\frac{6+2\varepsilon}{6+3\varepsilon}}(\log n)^8/m^{\frac{2}{6+3\varepsilon}} = O(nf(n,m)/m^{1/3}),$$

where $f(n,m) = (m/n)^{\varepsilon'}\log^8 n$, for another arbitrarily small $\varepsilon' > 0$. This is an improvement over Theorem 3.1 when m/n is small enough, but not over [8] when $m = \Omega(n\log n)$.

Theorem 4.7. *Given m units of storage, with m/n larger than some appropriate constant and $m < n^3$, simplex range reporting on n points in $\Re^3$ can be done in $O(nf(n,m)/m^{1/3}+k)$ query time, where $f(n,m) = (m/n)^{\varepsilon}\log^8 n$, for arbitrarily small (fixed) $\varepsilon > 0$, and k is the number of points to be reported. Preprocessing takes $O(m+n^{2+\varepsilon}/m^{1/3})$ randomized expected time.*

6. Conclusion

This paper gives an (almost) definitive answer to a classical multidimensional searching problem. We use two novel tools to achieve this goal: one is a way to enclose a set of points into the cells of a triangulation so that no hyperplane can "see" too many points. The other is an arrangement merging strategy with good zone theorems. On this subject, the obvious open problem is to generalize the zone theorems to higher dimension. Regarding range searching, it would be interesting to trade all n^{ε} factors for polylogarithmic ones. We also note that the constants of proportionality appearing in our bounds are rather large, and it would be interesting to find methods for improving them, especially in low dimensions.

Acknowledgments: We wish to thank Pankaj Agarwal for helpful discussions.

Work on this paper by Bernard Chazelle has been supported by NSF Grant CCR-87-00917. Work on this paper by Micha Sharir has been supported by Office of Naval Research Grant N00014-87-K-0129, by National Science Foundation Grants DCR-83-20085 and CCR-8901484, and by grants from the U.S.-Israeli Binational Science Foundation, the NCRD - the Israeli National Council for Research and Development, and the Fund for Basic Research administered by the Israeli Academy of Sciences. Work by Emo Welzl has been supported by Deutsche Forschungsgemeinschaft Grant We 1265/1-2. Micha Sharir and Emo Welzl have also been supported by a grant from the German–Israeli Binational Science Foundation. Last but not least, all authors thank DIMACS, an NSF Science and Technology Center, for additional support under Grant STC–88–09648.

REFERENCES

1. Agarwal, P. *Intersection and decomposition algorithms for arrangements of curves in the plane,* PhD Thesis, New York University, 1989.
2. Agarwal, P., Sharir, M. in preparation.
3. Aronov, B., Sharir, M. *Triangles in space, or building (and analyzing) castles in the air,* Proc. 4th ACM Symp. Comput. Geom. (1988), 381–391.
4. Chazelle, B. *Lower bounds on the complexity of polytope range searching,* J. Amer. Math. Soc. 2 (1989), 637–666.
5. Chazelle, B., Edelsbrunner, H., Guibas, L.,J., Sharir, M. *A singly-exponential stratification scheme for real semi-algebraic varieties and its applications,* Proc. 16th Internat. Colloq. on Automata, Languages and Programming (1989), 179–193.
6. Chazelle, B., Friedman, J. *A deterministic view of random sampling and its use in geometry,* Combinatorica 10 (1990).
7. Chazelle, B., Palios,L. *Triangulating a non-convex polytope,* Proc. 5th ACM Symp. Comput. Geom. (1989), 393–400.
8. Chazelle, B., Welzl, E. *Quasi-optimal range searching in spaces of finite Vapnik Chervonenkis dimension,* Disc. Comput. Geom. 4 (1989), 467–489.
9. Clarkson, K.L. *New applications of random sampling in computational geometry,* Disc. Comput. Geom. 2 (1987), 195–222.
10. Clarkson, K.L. *A randomized algorithm for closest-point queries,* SIAM J. Comput. 17 (1988), 830–847.
11. Clarkson, K.L., Edelsbrunner, H., Guibas, L.J., Sharir, M., Welzl, E. *Combinatorial complexity bounds for arrangements of curves and surfaces,* Proc. 29th Ann. IEEE Symp. Found. Comput. Sci. (1988), 568–579.
12. Clarkson, K.L., Shor, P. *Applications of random sampling in computational geometry II,* Disc. Comput. Geom. 4 (1989), 387–421.
13. Cole, R. *Searching and storing similar lists,* J. Algorithms 7 (1986), 202–220.
14. Cole, R., Yap, C.K. *Geometric retrieval problems,* Inform. and Control 63 (1984), 39–57.
15. Edelsbrunner, H. *Algorithms in Combinatorial Geometry,* Springer–Verlag, Heidelberg 1987.
16. Edelsbrunner, H. *The upper envelope of piecewise linear functions: Tight bounds on the number of faces,* Disc. Comput. Geom. 4 (1989), 337–343.
17. Edelsbrunner, H., Guibas, L.J., Hershberger, J., Seidel, R., Sharir, M., Snoeyink, J., Welzl, E. *Implicitly representing arrangements of lines or segments,* Proc. 4th ACM Symp. Comput. Geom. (1988), 56–69.
18. Edelsbrunner, H., Guibas, L.J., Sharir, M. *The upper envelope of piecewise linear functions: Algorithms and applications,* Disc. Comput. Geom. 4 (1989), 311–336.
19. Edelsbrunner, H., O'Rourke, J., Seidel, R. *Constructing arrangements of lines and hyperplanes with applications,* SIAM J. Comput. 15 (1986), 341–363.

20. Edelsbrunner, H., Welzl, E. *Halfplanar range search in linear space and $O(n^{0.695})$ query time,* Inform. Process. Lett. 23 (1986), 289-293.
21. Guibas, L.J., Seidel, R. *Computing convolutions by reciprocal search,* Disc. Comput. Geom. 2 (1987), 175–193.
22. Haussler, D., Welzl, E. *Epsilon nets and simplex range queries,* Disc. Comput. Geom. 2 (1987), 127–151.
23. Matoušek, J. *Cutting hyperplane arrangements,* Proc. 6th Annu. ACM Symp. Comput. Geom. (1990), 1–9.
24. Matoušek, J. *Spanning trees with low crossing number,* manuscript, 1989.
25. Mehlhorn, K. *Data structures and algorithms 3: Multidimensional Searching and Computational Geometry,* Springer-Verlag (1984).
26. Pach, J., Sharir, M. *The upper envelope of piecewise linear functions and the boundary of a region enclosed by convex plates,* Disc. Comput. Geom. 4 (1989), 291–309.
27. Paterson, M., Yao, F.F. *Point retrieval for polygons,* J. Algorithms 7 (1986), 441–447.
28. Pollack, R., Sharir, M., Sifrony, S. *Separating two simple polygons by a sequence of translations,* Disc. Comput. Geom. 3 (1988), 123–136.
29. Willard, D.E. *Polygon retrieval,* SIAM J. Comput. 11 (1982), 149–165.
30. Yao, F.F. *A 3–space partition and its applications,* Proc. 15th Ann. ACM Symp. Theory Comput. (1983), 258–263.
31. Yao, A.C., Yao, F.F. *A general approach to d-dimensional geometric queries,* Proc. 17th Ann. ACM Symp. Theory Comput. (1985), 163–168.