POINT LOCATION AMONG HYPERPLANES AND UNIDIRECTIONAL RAY-SHOOTING

Bernard Chazelle
Joel Friedman

CS-TR-333-91

June 1991

Point Location among Hyperplanes And Unidirectional Ray-Shooting

BERNARD CHAZELLE AND JOEL FRIEDMAN
Department of Computer Science
Princeton University
Princeton, NJ 08544

Abstract: We present an algorithm for locating a query point q in an arrangement of n hyperplanes in d-space. The size of the data structure is $O(n^d)$ and the time to answer any query is $O(\log n)$. Unlike previous data structures, our solution will also report, in addition to the face of the arrangement that contains q, the first hyperplane that is hit (if any) by shooting the point q in some fixed direction. Actually, if this ray-shooting capability is all that is needed, or if one only desires to know a single vertex of the face enclosing q, then the storage can be reduced to $O\big(n^d/(\log n)^{\lceil d/2\rceil-\varepsilon}\big)$, for any fixed $\varepsilon > 0$.

Bernard Chazelle wishes to acknowledge the National Science Foundation for supporting this research in part under Grants CCR–8700917 and CCR–9002352. Joel Friedman wishes to acknowledge the National Science Foundation for supporting this research in part under Grant CCR–8858788, and the Office of Naval Research under Grant N00014–87–K–0467.

1. Introduction

Let H be a set of n hyperplanes in $\mathbf{R}^d$ and let $\mathcal{A}(H)$ be the arrangement determined by these hyperplanes. We show how to preprocess H into a data structure of size $O(n^d)$ so that given any query point q the unique face of $\mathcal{A}(H)$ that contains q can be determined in $O(\log n)$ time. The data structure can be built deterministically in $O(n^{2d+2})$ time. It must be noted that the dimension d is assumed to be a constant and that the big-oh notation actually hides constant factors exponential in d.

Our result improves upon a solution of Clarkson [3], which requires $O(n^{d+\varepsilon})$ space, for any fixed $\varepsilon > 0$. One further advantage of our method is that if a direction is chosen ahead of time then the first hyperplane of H hit (if any) by shooting the query point in that direction can be found at no extra cost. Actually, if this ray-shooting capability is all that is needed, or if one only desires to know a single vertex of the face enclosing q, then the storage can be reduced to $O(n^d/(\log n)^{\lceil d/2\rceil-\varepsilon})$, for any fixed $\varepsilon > 0$.

Our data structures rely heavily on the probabilistic form of divide-and-conquer found in [3,5,8,11] and its derandomization given in [2]. The point location algorithm is developed in several stages by bootstrapping and mixing together four intermediate solutions. The first one, Solution A, is Clarkson's algorithm itself: it requires $O(n^{d+\varepsilon})$ space, for any fixed $\varepsilon > 0$. Solution B improves this space bound by making ε a function of n that goes to 0 as n goes to infinity. Solution C introduces the main data structuring scheme, which further reduces the space bound to $O((n \log n)^d)$. Solution D trades space for time by requiring only $o(n^d)$ storage but substantially more than $O(\log n)$ query time. Finally, Solution E presents our last data structure, which requires $O(n^d/(\log n)^{\lceil d/2\rceil-\varepsilon})$ space, for any fixed $\varepsilon > 0$, and $O(\log n)$ query time. Obviously, the space bound climbs to $O(n^d)$ if we require a full representation of the arrangement.

2. Some Results on Derandomization

We begin with a brief review of fundamental geometric concepts (see e.g., [3,6]). A set is *polyhedral* if it is the intersection of a finite number of closed halfspaces. The *arrangement* $\mathcal{A}(H)$ of a finite collection H of hyperplanes in $\mathbf{R}^d$ is the cell complex induced by the hyperplanes. An element of the complex is called a *face*, or a *k-face* if we want to specify its dimension k; a vertex is a 0-face, an edge is a 1-face, and a cell is a d-face. A face is the relative interior of some polyhedral set bounded by hyperplanes in H. A *triangulation* of $\mathcal{A}(H)$ is a simplicial cell complex which refines the arrangement $\mathcal{A}(H)$ (i.e., each face is a simplex of dimension d or less). For example, we can define a *canonical triangulation* [4] by first triangulating recursively the $(d-1)$-dimensional cross-section of the arrangement made by each hyperplane, and then for each cell of the arrangement, lifting all the k-simplices on its boundary ($k = 0, \ldots, d-1$) toward a chosen vertex (except for the simplices decomposing the faces incident upon the vertex in question).

We now recall some results from [2] which provide some important tools for our constructions. Let $\mathcal{H} = (V, E)$ be a hypergraph of n vertices. In our application, the vertices correspond to the hyperplanes of H; each edge is associated with a pair of distinct vertices in $\mathcal{A}(H)$ and indicates which hyperplanes of H separate the two vertices (i.e., intersect the relatively open segment connecting them but not its endpoints). For simplicity, we shall assume that the hyperplanes are in general position. By abuse of notation we identify V and H, so that we might speak of the hyperplanes of V. The two vertices of $\mathcal{A}(H)$ associated with a given edge $e \in E$ are specified by a certain subset $\sigma(e) \subseteq V$ of hyperplanes (each vertex contributing the hyperplanes of which it is the intersection). Let h be the maximum value of $|\sigma(e)|$ over all $e \in E$; in our application, $h = 2d$. We can verify that given any $W \subseteq V$ of size at most h the cardinality of $\sigma^{-1}(W)$ is bounded above by a constant. Given a subset $R \subseteq V$ of r hyperplanes, an edge e is called *t-deficient* if (i) $|e| \geq tn/r$, (ii) $\sigma(e) \subseteq R$, (iii) $R \cap e = \emptyset$. This simply means that the two vertices of $\mathcal{A}(R)$ with which e is associated are separated by at least tn/r hyperplanes of H and are incident upon a common face of $\mathcal{A}(R)$. Given R, let $\chi(t)$ be the number of t-deficient edges. We say that R is *conformal* if

$$\chi(t) \leq \frac{\alpha\chi(0) + 1}{2^{t/(2h)}},$$

for $t = 1, 2, \ldots, r$, where α is some appropriate constant.

To use the results of [2], we need to ensure that $\mathcal{H}$ is *robust*, meaning roughly that the average number of 0-deficient edges, when R is chosen uniformly at random, cannot decrease as r increases. That this should hold in our particular application is not entirely clear, but we can use a weaker form of robustness, which involves replacing $\chi(0)$, in the definition of conformality, by a nondecreasing function of r which upper bounds the actual value of $\chi(0)$ for any R. To do so, we need a result of Aronov, Matoušek and Sharir [1], which says that that the sum of the squares of the size complexities of the faces of an arrangement of r hyperplanes in d-space is $O\big(r^d(\log r)^{\lfloor d/2 \rfloor - 1}\big)$. Then, the result from [2] which we need says that if r is large enough, it is possible to compute a conformal R in time $O\big(rn^{h+1}\big)$ time. This means that R can be found such that the number of t-deficient edges vanishes exponentially in t. More precisely, the number of pairs of vertices incident upon a common face and intersected by at least tn/r hyperplanes of H is at most $O\big(r^d(\log r)^{\lfloor d/2 \rfloor - 1}/c^t\big)$, for some constant $c > 1$.

Lemma 1. *There exists a collection R of r hyperplanes in H such that, for any $t > 0$, the number of pairs of vertices incident upon a common face of $\mathcal{A}(R)$ and intersected by at least tn/r hyperplanes of H is $O\big(r^d(\log r)^{\lfloor d/2 \rfloor - 1}/c^t\big)$, for some constant $c > 1$. The set R can be computed in $O\big(rn^{2d+1}\big)$ time.*

Let's change our perspective somewhat and apply the same result to a geometric construction which we call the *vertical decomposition* of $\mathcal{A}(R)$. We decompose the closure of each cell c of $\mathcal{A}(R)$ into a cell complex by erecting "walls" along the x_1-direction as follows: Take each k-face (for all $k < d - 1$) of the closure c^* of c, and (1) lift it by taking its cartesian product with the x_1-axis and (2) intersect the lifted face with c^*. To simplify the discussion, let assume that c^* is bounded. Since c^* is convex, the k-faces to be lifted fall into three categories: those for which the lifting proceeds in the positive direction, those going in the negative direction, and those faces which cannot be lifted within c^* (the silhouette of c^*). None of these categories alone can cause an asymptotic blow-up in the complexity of the resulting structure. The first two categories, however, collide head-on, in the sense that within a given cell walls from distinct categories can cut each other. By the zone theorem for hyperplanes [7], however, it easily follows that the total size of the resulting structures, over all cells of $\mathcal{A}(R)$, remains $O(r^d)$.

To be convinced of this, it suffices to observe that the size (i.e., combinatorial complexity) of the vertical decomposition is proportional to the number of features in (i) the arrangement $\mathcal{A}(R)$ itself, (ii) the zone of every hyperplane in R, and more generally (iii) the zone of each k-wise intersection I of hyperplanes in R with respect to the cross-section of R with the $(d-k+1)$-flat passing through I that is parallel to the x_1-axis. The zone theorem for hyperplanes [7] shows that the size in (ii) is $O(r^{d-1})$ and that each zone in (iii) has size $O(r^{d-k})$. There are $O(r^k)$ k-wise intersections, so the size of the vertical decomposition is $O(r^d)$.

The walls of the decomposition partition each cell closure c^* into a cylindrical cell complex. Each face of the complex can be further decomposed by lifting a canonical triangulation of its base in the x_1-direction. This produces a cell complex whose faces are either simplices of dimension less than d (on the boundary of c^*) or are (topologically) products of such simplices with line segments. Note that although each cell is decomposed into a cell complex, their union does *not* produce a cell complex (since adjacent cells do not glue together properly). The size of the vertical decomposition in its final state is still $O(r^d)$.

Given the subset R, to compute the vertical decomposition can be done most simply by intersecting the cylinder erected from each face on the boundary of c^* with all the other faces. These intersections take place in $(d-1)$-space, so the work involved can be reduced to computing the intersection of up to n halfspaces in $(d-1)$-space, which can be done in $O(r^{\lfloor d/2 \rfloor} \log r)$ time by using Seidel's output-sensitive convex hull algorithm in dual space [12]. We know that at most $O(r^d (\log r)^{\lfloor d/2 \rfloor - 1})$ such intersections will be computed. Therefore, setting up the vertical decomposition takes $O(r^{3d/2+\varepsilon})$ time, for any fixed $\varepsilon > 0$.

We wish to ensure that no face in the decomposition of any c^* intersects more than $n(\log r)/r$ hyperplanes of H, up to within a constant factor. Since these faces have constant description size it suffices to guarantee that any segment within each c^* intersects $O(n(\log r)/r)$ hyperplanes. It is easily seen that ensuring this property for the segments joining pairs of vertices in c^* is actually sufficient. This can be done by appealing to Lemma 1.

Lemma 2. *For any $r \leq n$ large enough, there exists a collection R of r hyperplanes in H such that any face in the vertical decomposition of $\mathcal{A}(R)$ is intersected by $O(n(\log r)/r)$ hyperplanes of H. Computing R and the vertical decomposition can be done in $O(rn^{2d+1})$ time.*

We conclude this section with a trivial but useful technical result. Let f be a positive nondecreasing function such that, for any x, y large enough, $f(xy) \leq x^\alpha f(y)$, for some fixed constant $\alpha \geq 1$. Given $n \geq 2$, let $n_1, \ldots, n_p$ be a set of integers such that for any $t > 1$, $|\{ i : n_i > tn \}| \leq ap/b^t$, for some constants $a, b > 1$.

Lemma 3. *The sum $\sum_{1 \leq i \leq p} f(n_i)$ is at most proportional to $pf(n)$.*

3. The Point Location Algorithm

As we said earlier, we need four intermediate solutions, A through D, from which the final point location algorithm, Solution E, is derived.

Solution A

This is Clarkson's method [3]. Take a random sample R of $r = O(1)$ hyperplanes and triangulate the arrangement $\mathcal{A}(R)$. With at least constant positive probability, no face of the triangulation is intersected by more than $cn(\log r)/r$ hyperplanes, for some fixed constant c. If this condition does not hold, repeat the sampling until it does. From a result of Matoušek [9] an appropriate R can be found deterministically in $O(n)$ time. Next, for each face f of $\mathcal{A}(R)$ in turn, identify the subset $H_f \subseteq H$ of hyperplanes that intersect f (but do not contain it). If H_f is not empty then construct the data structure recursively, with the input consisting of H_f and the hyperplanes bounding f. Note that if the face f is of dimension $k < d$, then we need a data structure for searching in k-space, so in the recursive step we must provide not quite the hyperplanes which we specified above but rather their intersections with the affine closure of f.

To answer a query we first locate the point in the triangulation of $\mathcal{A}(R)$ by checking its inclusion in each of the faces, one by one. If the enclosing face is not intersected by other hyperplanes (in $H \setminus R$) then it is the answer which we are looking for and we are done. Otherwise, we proceed recursively in the data structure defined for that face. Ultimately, we must fall in the first case and thus be able to answer the query. Note that because the construction algorithm was careful to include the bounding faces of the simplices within which it recurses, the answer to the query is a simplex that falls entirely within some face of $\mathcal{A}(H)$: the correspondence can be precomputed and encoded in the data structure. If r is chosen to be a large constant, then this scheme requires $O(n^{d+\varepsilon})$ storage, for any fixed $\varepsilon = \varepsilon(r) > 0$, and $O(\log n)$ query time, respectively. The preprocessing time is also $O(n^{d+\varepsilon})$.

Solution B:

Clarkson's algorithm reports the name of the face that contains the query point, but it does not report the name of the hyperplane right "above" the query point. Assume that we have a system of reference $(O, x_1, \ldots, x_d)$, none of whose axes is parallel to any hyperplane of H. Given a query point q we want to know the first hyperplane encountered (if any) as we move q along Ox_1 in either direction. Let h be one of these hyperplanes and let q' be the projection of q onto h along x_1; the intersection of h and the plane (O, x_1, x_2) is a line parallel to which we can move q' until we reach another hyperplane of H, etc. Iterating in this manner, the point q is thus associated with 2 points along direction (O, x_1), 4 points on a plane parallel to (O, x_1, x_2), 8 points on a 3-flat parallel to (O, x_1, x_2, x_3), etc. Note that the final 2^d-tuple, which we call the *antenna* of q, consists of vertices of $\mathcal{A}(H)$ which need not be distinct. These various k-tuples not only allow us to locate q within $\mathcal{A}(H)$ but also provide useful additional information, as we shall see below. All our subsequent point location algorithms will compute these d tuples.

Let us describe a method for computing the 2-tuple of a query point. By applying the data structure recursively for each hyperplane it will then be easy to derive all the remaining tuples, and hence, the antenna. Take a subset R of r hyperplanes in H and compute the vertical decomposition of $\mathcal{A}(R)$. We need a data structure for locating a point within $\mathcal{A}(R)$ and, if it does not lie on any hyperplane of R, within the (vertical) decomposition of its enclosing cell in $\mathcal{A}(R)$ as well. This is done most simply by applying Solution A to the set of hyperplanes spanning the $(d-1)$-faces of the vertical decompositions of the closures of all the cells. Since there are a total of $O(r^d)$ such faces, this produces one giant data structure of size $O(r^{d(d+1)})$ (setting $\varepsilon = 1$). We complete the data structure by considering each face in the decomposition of each cell c and recursing within it, using as input the hyperplanes that intersect the face in question. Note that we do not recurse within the faces on the boundary of c^*, nor do we recurse within the faces produced by Solution A (those are used only as an intermediate device).

Under the conditions of Lemma 2, any face in which the construction algorithm recurses intersects only $O(n(\log r)/r)$ hyperplanes. Setting $r = n^{1/(2d)}$, the storage requirement $S(n)$ follows the recurrence: $S(n) = O(1)$, for $n = O(1)$, and

$$S(n) = c_1 n^{1/2} S\big(c_2 n^{1-1/(2d)} \log n\big) + O\big(n^{(d+1)/2}\big),$$

for some constants $c_1, c_2 > 0$. We easily verify by induction that $S(n)$ is at most proportional to $n^d 2^{(d \log\log n)^2}$ (all logarithms are to the base 2). The construction takes time $T(n)$, with $T(n) = O(1)$, for $n = O(1)$, and

$$T(n) = c_1 n^{1/2} T\big(c_2 n^{1-1/(2d)} \log n\big) + O\big(n^{2d+2}\big),$$

which certainly is in $O\big(n^{2d+2}\big)$.

To answer a query, we begin by locating the point q in the Clarkson-type data structure at the top level. If the point lies on one of the hyperplanes of R then this gives us the 2-tuple of the point and we are done. Otherwise, we find the cell of $\mathcal{A}(R)$ that contains q as well as its enclosing face in the vertical decomposition of the cell. If this face has no further intersections with other hyperplanes then its two bases (recall that it is a cylinder) gives the 2-tuple of the point q; otherwise, we must recurse within the data structure associated with that face. The query time $Q(n)$ follows a recurrence of the form: $Q(O(1)) = O(1)$ and $Q(n) = Q\big(n^{1-\nu}\big) + O(\log n)$, for some constant $\nu > 0$, which gives $Q(n) = O(\log n)$.

To go from the 2-tuple of the query point to its antenna, we need to augment the data structure as follows: for each hyperplane of H, form its intersections with the $n-1$ remaining hyperplanes and feed the resulting $(d-2)$-flats as input to a data structure built recursively in dimension $d-1$. This allows us to compute all the d k-tuples of a query point in $O(\log n)$ time, using $O\big(n^d 2^{(d \log\log n)^2}\big)$ storage and $O\big(n^{2d+2}\big)$ preprocessing time.

Solution C:

We now show how to reduce the space complexity to $O\big(n^d \log^d n\big)$ by bootstrapping the previous solution. This will also give us a chance to introduce the main data structuring technique used in this work. The idea is to apply Solution B to a slightly sublinear subset of the hyperplanes chosen carefully. We then argue that the set of tuples computed by the query algorithm with the sample set either provides the right answer directly or else points to a small number of hyperplanes from which the correct set of tuples can be derived by again applying Solution B.

The preprocessing is very simple to describe. Pick a subset R of r hyperplanes in H that satisfies the conditions of Lemma 1 and preprocess it along the lines of Solution B. Given a cell c of $\mathcal{A}(R)$, let $v_1, \ldots, v_k$ be the vertices incident upon it: form a k-by-k matrix whose (i,j)-entry is a pointer to the list of hyperplanes in H that intersect the line segment $v_i v_j$ (but not its endpoints). Preprocess each such list according to Solution B. To compute the 2-tuple of a query point q, we first determine its antenna K within $\mathcal{A}(R)$, using Solution B. If q is found to lie on a hyperplane of R then we are done. Otherwise, to see whether the antenna of q within $\mathcal{A}(H)$ differs from K, and if so, how, we form all pairs of vertices in K and we look up the corresponding entries in the matrix associated with the cell containing q. This gives us a constant number of lists of $O(n(\log r)/r)$ hyperplanes each, which we check by using Solution B.

We claim that the desired 2-tuple is formed by some of these hyperplanes or possibly by the hyperplanes of R corresponding to the 2-tuple computed by Solution B at the top level. To see this, let t be the 2-tuple of q within $\mathcal{A}(R)$. It suffices to show that if a hyperplane separates the two points in t then it must intersect the convex hull of K. This follows from the fact, trivially shown by induction, that the 2-tuple of a point within an arrangement lies in the convex hull of its 2^k-tuple, for any $k > 1$.

As we saw earlier, the total size p of all these matrices is $O\big(r^d(\log r)^{\lfloor d/2\rfloor-1}\big)$. From Lemmas 1 and 3 we easily find that the size of the data structure is at most on the order of

$$r^d 2^{(d\log\log r)^2} + r^d(\log r)^{\lfloor d/2\rfloor-1}(n/r)^d 2^{(d\log\log(n/r))^2},$$

which, setting r to be roughly $n/2^{(d\log\log n)^2}$, gives the conservative upper bound of $O\big(n^d\log^d n\big)$. The construction takes $O\big(n^{2d+2}\big)$ time. The query time is $O\big(\log r + \log(n(\log r)/r)\big)$, which is $O(\log n)$. Again, by recursive construction of the data structure, we can convert this algorithm for computing the 2-tuple of a query point into one that determines its antenna. This only affects constant factors in the complexity bounds stated above.

Solution D:

We use the same scheme but instead of using Solution B at both levels, we use Solution C at the top level and the naive algorithm at the bottom level. In other words, Solution C is applied to the sample of hyperplanes, and each list of hyperplane pointed to by the entries of the matrices are left unprocessed. The query algorithm searches those lists by examining each of their elements. Again, we can appeal to Lemma 3 and derive that the size of the data structure is now at most proportional to

$$r^d\log^d r + r^d(\log r)^{\lfloor d/2\rfloor-1}(n/r),$$

which, setting $r = n^{1-\varepsilon}$, gives an upper bound of $O\big(n^{d-(d-1)\varepsilon}(\log n)^{\lfloor d/2\rfloor-1}\big)$. Again, the construction time is $O\big(n^{2d+2}\big)$ time. The query time is $O\big(n^\varepsilon\log n\big)$. We use the usual format to go from the 2-tuple of the query point to its full set of tuples.

Solution E:

Finally, we combine Solutions C and D into a leaner, faster algorithm. The setting is still the same, but we now apply Solution C to the top level (the hyperplanes of R) and Solution D to the bottom level (the hyperplanes pointed to by the matrix entries). From one final use of Lemma 3, we find that the size of the data structure is at most proportional to

$$r^d \log^d r + r^d (\log r)^{\lfloor d/2 \rfloor - 1} \left(\frac{n}{r}\right)^{d-(d-1)\varepsilon} \left(\log \frac{n}{r}\right)^{\lfloor d/2 \rfloor - 1},$$

which if we set r to be roughly $n(\log n)^{1-1/(\varepsilon+\varepsilon^2)}$ gives $O\big(n^d/(\log n)^{\lceil d/2 \rceil - \varepsilon'}\big)$, for any fixed $\varepsilon' > 0$. The construction takes $O\big(n^{2d+2}\big)$ time. The query time is $O\big(\log r + (n(\log r)/r)^{\varepsilon} \log(n(\log r)/r)\big)$, which is $O(\log n)$. One last time we use recursion to augment the data structure to allow for the computation of not merely the 2-tuple of a query point but its antenna, too.

Theorem 5. *It is possible to preprocess a set of n hyperplanes in d-space using $O\big(n^d/(\log n)^{\lceil d/2 \rceil - \varepsilon}\big)$ storage, for any fixed $\varepsilon > 0$, so that given any query point, the first hyperplane (if any) that is hit by shooting from the point in a fixed direction can be determined in $O(\log n)$ time. Within the same time a vertex of the enclosing face can be found, and if an explicit representation of the arrangement of hyperplanes is available (which requires $O\big(n^d\big)$ storage) then the face enclosing the point can also be found in $O(\log n)$ time. The data structure can be built in $O\big(n^{2d+2}\big)$ time.*

6. Conclusions

An immediate application of the point location algorithm is halfspace range searching. By duality, we can preprocess n points in d-space so that given a query halfspace the number of points in it can be computed in $O(\log n)$ time. The data structure requires $O\big(n^d\big)$ storage and can be built in $O\big(n^{2d+2}\big)$ time.

The fact that $o(n^d)$ storage suffices to do ray-shooting (in a fixed direction) suggests that perhaps much lower space bounds can be achieved. At this point our bound is constrained by the best current bound on the sum of squares of face sizes in an arrangement of hyperplanes. Even if not optimal this bound is unlikely to decrease much more in the future, so a different approach might be needed. The preprocessing time of our algorithms can be reduced a little by a more careful analysis or perhaps by adapting some of Matoušek's recent results [9,10]. Unfortunately, the large size of our samples might prevent any dramatic savings. This might perhaps be partly alleviated, however, by bootstrapping the algorithm a large but bounded number of times.

Acknowledgments: We wish to thank Otfried Schwarzkopf for pointing out an error in an earlier version of our point location algorithm [Princeton Technical Report, TR # 181–88]. We also thank Herbert Edelsbrunner and Micha Sharir for helpful conversations.

REFERENCES

1. Aronov, B., Matousek, J., Sharir, M. *Sum of squares of cell complexities and the complexity of many cells in arrangements of hyperplanes,* in preparation, 1990.
2. Chazelle, B., Friedman, J. *A deterministic view of random sampling and its use in geometry,* Combinatorica 10 (1990), 229–249.
3. Clarkson, K.L. *New applications of random sampling in computational geometry*, Disc. Comp. Geom. 2 (1987), 195–222.
4. Clarkson, K.L. *A randomized algorithm for closest-point queries,* SIAM J. Comput. 17 (1988), 830–847.
5. Clarkson, K.L., Shor, P.W. *Applications of random sampling in computational geometry, II*, Disc. Comp. Geom. 4 (1989), 387–421.
6. Edelsbrunner, H. *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, Germany, 1987.
7. Edelsbrunner, H., Seidel, R., Sharir, M. *New proofs of the zone theorem for arrangements of hyperplanes,* manuscript, 1991.
8. Haussler, D., Welzl, E. *Epsilon-nets and simplex range queries*, Disc. Comp. Geom. 2, (1987), 127–151.
9. Matoušek, J. *Cutting hyperplane arrangements,* to appear in Disc. Comp. Geom. Prelim. version in 6th Ann. ACM Symp. on Comput. Geom. (1990), 1–9.
10. Matoušek, J. *Approximations and optimal geometric divide-and-conquer,* Proc. 23rd Ann. ACM Symp. Theory of Comput. (1991), 505–511.
11. Reif, J.H., Sen, S. *Optimal randomized parallel algorithms for computational geometry*, Proc. 16th Internat. Conf. Parallel Processing, St. Charles, IL, 1987. Full version, Duke Univ. Tech. Rept., CS–88–01, 1988.
12. Seidel, R. *Constructing higher-dimensional convex hulls at logarithmic cost per face,* Proc. 18th Ann. ACM Symp. Theory Comput. (1986), 404–413.