Efficiently computing the closest points to a query line / segment

Eli Zeitlin * Michael Segal †

Abstract

In this paper we present an algorithm which preprocesses a set P of n points and a point q in $O(n \log n)$ time with $O(n)$ space to report the point from P which has the minimum perpendicular distance from a query line which passes through a point q in $O(\log n)$ time. We also present an algorithm for finding k nearest neighbors of given query segment among a set P. Our algorithm requires $O(n^{2+\varepsilon})$ time and space for preprocessing, and it answers the query for a given segment in $O(k + \log^3 n)$ time.

1 Introduction

Given a set $P = \{p_1, p_2, \ldots, p_n\}$ of n points arbitrarily distributed on a plane and special point q, we study the problem of finding the point nearest (in the sense of perpendicular distance) to a given line which passes through a point q. The given line usually comes in the form of query.
An algorithm of preprocessing time and space $O(n \log n)$ was proposed by Mitra and Chaudhuri [1] which can answer the query in $O(\log^2 n)$ time. Later, the same problem was solved using geometric duality by Nandy, Das and Goswami [2] they use randomized technique to construct a data structure of size $O(n)$ in $O(n \log n)$ expected time, which answers the query in $O(\log n)$ time. Here we show that we can improve the preprocessing time

*Department of Computer Science, Tel-Aviv University, Tel-Aviv 0000, Israel. e-mail: zeitline@post.tau.ac.il

†Communication Systems Engineering Department, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel. e-mail: segal@cse.bgu.ac.il

to be $O(n \log n)$ deterministic time, with the same query time $O(\log n)$ and space complexities $O(n)$ as in [2]. We use geometric duality [3] for solving this problem. Our algorithm is based on maintaining zone (or horizon) of the dual point q in an arrangement of the duals of the points in P. By using cutting trees we show how to solve the $k-$closest points problem to arbitrary query segment. Our results are: preprocessing time and space $O(n^{2+\varepsilon})$, query time $O(k + \log^3 n)$. We also show algorithm with following properties, space $O(n)$, preprocessing time $O(n^2(\log n)^{\varepsilon})$, query time $O(k + \log^7 n)$.

The paper is organized as follows: In section 2, we recall the duality and provide some definitions. In section 3, we show the preprocessing step for the line problem using $O(n \log n)$ deterministic time and $O(n)$ space. In section 4, we show how to report the point with the minimum perpendicular distance when the line query is given in $O(\log n)$. Subsequently, in section 5 we address a natural generalization of the problem, and show how to report $k-$nearest points to the query line. In section 6, we show how to answer efficiently $k-$nearest points to arbitrary query segment.

2 Geometric preliminaries

First we mention that we need to maintain an array with the points in P, sorted with respect to their x-coordinates. This requires $O(n)$ space and can be constructed in $O(n \log n)$ time. If the query line is vertical, we find the position of its x coordinate by performing a binary search in the array P and report the closest point. So, the query time complexity is $O(\log n)$.
We now consider the case where the query line is non-vertical. We use geometric duality for solving the problem. Here:

1. A point $p = (a, b)$ of the primal plane is mapped to the line $p^* : y = ax - b$ in the dual plane

2. A non-vertical line $l : y = mx - c$ of the primal plane is mapped to the point $l^* = (m, c)$ in the dual plane

The incidence and order relationships between a point p and a line l in the primal plane remain preserved among their duals in the dual plane [3].
Let H be the set of dual lines corresponding to the points in P. Let $\mathcal{A}(H)$ denote the arrangement of the set of lines H. The number of vertices, edges and faces in $\mathcal{A}(H)$ are all $O(n^2)$ [4]. Given a query line l which passes through

a point q, the problem of finding its nearest point can be solved as follows: Use the point location algorithm of [5] to locate the cell $\mathcal{A}(H)$ containing l^* (the dual of the line l) in $O(\log n)$ time. As the cells of the arrangement $\mathcal{A}(H)$ are split into trapezoids, the lines p_i^* and p_j^* lying vertically above and below l^*, can be found in constant time. The distance of a point p and the line l in the primal plane can be obtained from the dual plane as follows: Draw a vertical line from the point l^* which meets the line p^* at a point $\alpha(l^*, p^*)$ in the dual plane. The *perpendicular distance* of the point p and the line l in the primal plane is equal to $d(l^*, \alpha(l^*, p^*))/\sqrt{1 + (x(l^*))^2}$, where $d(\cdot, \cdot)$ denotes the distance between two points, and $x(l^*)$ is the x-coordinate of the point l^*.
Thus, the point nearest to l in the primal plane will be any one of p_i or p_j depending on whether $d(l^*, \alpha(l^*, p_i^*)) <$ or $> d(l^*, \alpha(l^*, p_j^*))$. The preprocessing time and space required for creating and sorting $\mathcal{A}(H)$ are both $O(n^2)$ [5]. From now onwards, $d(l^*, \alpha(l^*, p^*))$ will be referred as the *vertical distance* of the line p^* from the point l^*.

Definition 1 The *zone* of a line l in the arrangement $\mathcal{A}(H)$ induced by a set H of lines in the plane is the set of faces of $\mathcal{A}(H)$ whose closure intersects l.

Figure 1 gives an example of a zone. The complexity of a zone is defined as the total complexity of all faces it consists of, that is, the sum of the number of edges and vertices of these faces. And it's known to be $O(n)$ [6]

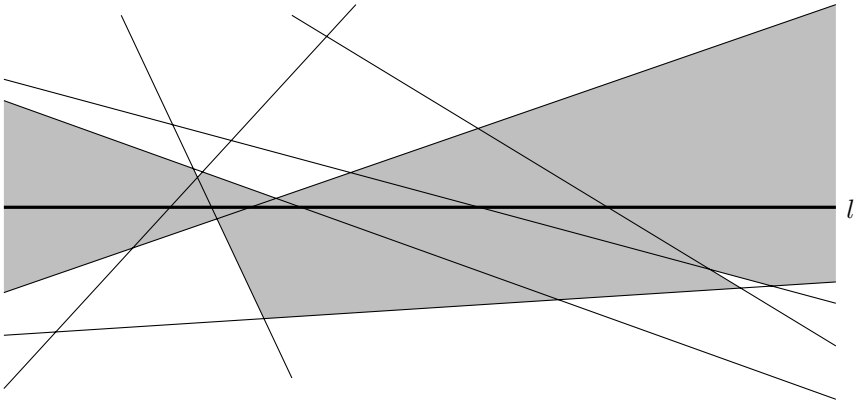

Figure 1: The zone of a line in an arrangement of lines

3 Preprocessing step of the algorithm

In this section we present an Algorithm **A** to preprocess the set of points P and point q which uses $O(n \log n)$ deterministic time and $O(n)$ space. Thus we improve the previous result from [2].
Without loss of generality we can assume that:

1. Point $q = (0, 0)$, because we can move all the points in P by $-q$ in linear time

2. Query line which passes through a point $q = (0, 0)$ will be non-vertical, because we can find the closest point to axis y in $O(\log n)$ (see: *Geometric preliminaries*)

The point $q = (0, 0)$ in primal plane transformed to line $q^* : y = 0$ in the dual plane. So, our aim in the preprocessing phase is to build up a zone of axis x (a line q^*) in an arrangement $\mathcal{A}(H)$.

Algorithm A:

- *Comments.* This algorithm builds up a zone of a line q^* in an arrangement $\mathcal{A}(H)$ and two arrays of segments on line q^* from vertical decomposition.

- *Step I.* Transform each point $p = (a, b) \in P$ to line $p^* : y = ax - b$ in the dual plane and add the line p^* to H. Compute the n intersection points

of the H lines with axis x and sort them by x-coordinate in $O(n \log n)$ time. The remainder of this algorithm will run in $O(n)$ time.

- *Step II.* Compute the left and right horizon trees of lines to the top / bottom of axis x.
- *Step III.* Compute the vertical decomposition (projection) of the zone. Save the new edges on line q^* into array in increasing order.

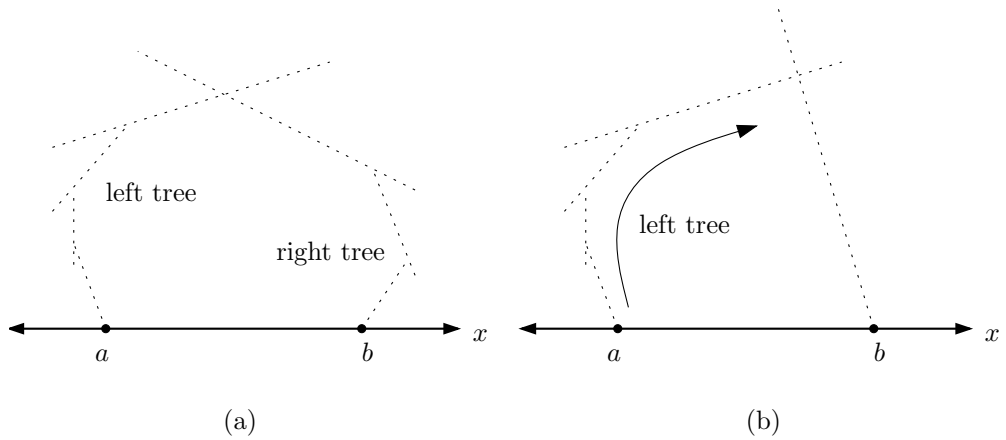

Figure 2: The faces between a and b is the intersection of their bays (a); inserting the line from a can be charged to the circled points of the right bay (b)

Below we briefly sketch the idea of building the zone / horizon.

Given the left and right horizon trees, we claim that each face adjacent to axis x may be traversed in time linear in its size. If we can show this, then it will follow from the zone theorem that we can walk all the faces in $O(n)$ time.

To prove the claim, consider any two consecutive points a and b along axis x with a before b as in figure2(a). Then the face between a and b is the intersection of the left horizon tree bay of a and the right horizon tree bay of b. To construct the face, we need to walk up the bays from a and b, being careful not to walk too far in either bay. A sufficient rule is to visit the points of the two bays in order of increasing y-coordinate; on each step we need to check if we have crossed the line from the other bay.

Now it suffices to show how to construct the horizon tree in the linear time; we will describe the method for the left horizon tree. We construct the tree by inserting the lines in increasing order of their *intersections* with axis x. Suppose we have just inserted the line from point a, and we now want to insert the line from the next point b (figure2(b)). We simply walk up the bay from a, checking whether the line from b intersects each edge. Then the time to do insert this line is $O(1)$ plus the number of vertices passed over in the bay from a. Since such a vertex will thereafter be hidden from above by the line from b, it will never again be charged, and hence the total charges and running time are $O(n)$.

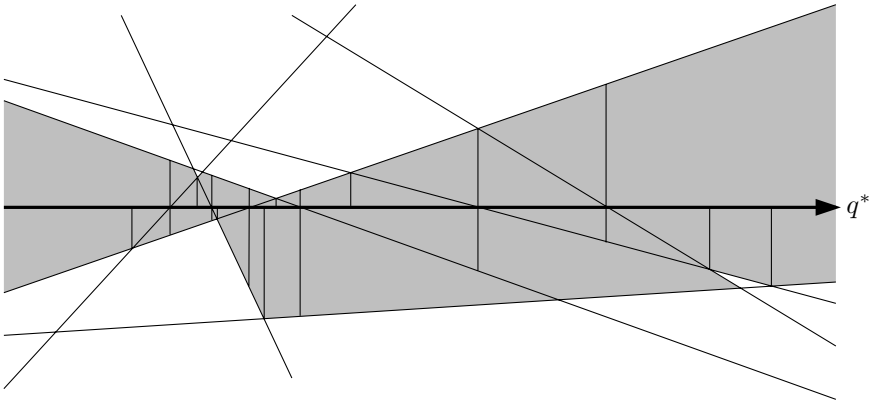

Figure 3: Vertical decomposition of the zone

Step III of the algorithm performs vertical decomposition of the computed zone in the previous step by travelling in each horizon tree from left to right (refer to Figure 3 for an illustration). By holding two simple arrays of segments, one for upper decomposition (above q^*) and second for lower decomposition (below q^*) we fill them with appropriate edges created on line q^* during the vertical decomposition process. For each segment in the arrays we save a pointer to the line strictly above/below the segment. The running time of this step is $O(n)$.

4 Answering the nearest point to the query line

In this section we present an Algorithm **Q** to answer the query efficiently in $O(\log n)$ time.

Algorithm Q:

- *Comments.* This algorithm computes the nearest point $p \in P$ to given non-vertical line l. Without loss of generality we can assume that $q = (0, 0)$, so the line l is passed through $(0, 0)$.
- *Step I.* Transform the line $l : y = mx$ to point $l^* = (m, 0)$.
- *Step II.* Perform a binary search in two arrays to find two segments i and j where the point $(m, 0)$ falls. Get p_i^* and p_j^*, the lines strictly above/below the segments accordingly.
- *Step III.* Report the nearest point is p_i to l in the primal plane if $d(l^*, \alpha(l^*, p_i^*)) < d(l^*, \alpha(l^*, p_j^*))$ otherwise report p_j.

Step I and III of the algorithm takes $O(1)$, when Step II obviously takes $O(\log n)$.

5 Answering the $k-$nearest points to the query line

In this section, we address a natural generalization of the above problem where objective is to report $k-$nearest points to the query line in the same environment. We assume that k is known in preprocessing step.
Now, we want to find $k-$nearest lines from point strictly above/below the point $l^* = (m, 0)$ in the dual plane. We split each line $h_i \in H$ into two parts h_i^a and h_i^b, where h_i^a is the portion of h_i above q^*, and h_i^b is the portion of h_i below q^*. Let $H^a = \{h_1^a, h_2^a, \ldots, h_n^a\}$ and $H^b = \{h_1^b, h_2^b, \ldots, h_n^b\}$. We use $\mathcal{A}_{\leq k}(H^a)$ and $\mathcal{A}_{\leq k}(H^b)$ to denote $(\leq k)$-levels above and below q^*, respectively.

Theorem 1 ***($\leq k$)-Zone Theorem [7]*** *The complexity of the $(\leq k)$-zone of a line in an arrangement of n lines in the plane is $O(nk)$.*

???????
A randomized algorithm is proposed in [8] which can computes $\mathcal{A}_{\leq k}(H^a)$ and $\mathcal{A}_{\leq k}(H^b)$ in $O(n \log n + nk)$ expected time.

6 Answering the $k-$nearest points to the query segment

In this section, we address a natural generalization of the above problem where objective is to report $k-$nearest points to the arbitrary query segment. We assume that k is known in preprocessing step.

Duality observation
Let us first look on the problem in the dual plane. The points are mapped to lines. The query segment s is mapped to double wedge s^*, which bounded by the duals of the endpoints of s. For a segment s, the *slab of* s, or $slab(s)$, is the region bounded by the perpendiculars to s through the endpoints of s. The $slab(s)$ which bounded by the line l (perpendicular line through left endpoint of s) and line r (perpendicular line through right endpoint of s) is mapped to segment $\overline{l^* r^*}$ in the dual plane. Figure 4 shows an example of the problem in the primal and dual plane.

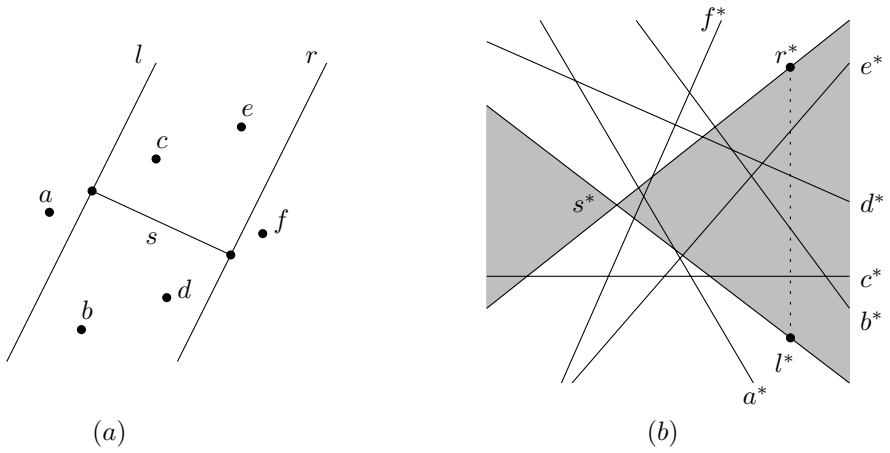

Figure 4: (a) Primal plane - Six points, segment s and $slab(s)$.
(b) Dual plane - Six lines, double wedge s^* and vertical segment $\overline{l^* r^*}$

Lemma 1 *The slab*(s) *is mapped to* ***vertical*** *segment* $\overline{l^*r^*}$ *in the dual plane.*

Proof. The *slab*(s) is bounded by lines l and r accordingly. Lines l and r are parallel, then $l : y = mx - c_l$ and $r : y = mx - c_r$. So we get in the dual plane two points $l^* = (m, c_l)$ and $r^* = (m, c_r)$ that defines vertical segment $\overline{l^*r^*}$. □

Lemma 2 *The point* $p \in$ *slab*(s) *in the primal plane if and only if the line* p^* *in the dual plane intersect the vertical segment* $\overline{l^*r^*}$.

Proof. Since the incidence and order relationships between a point p and a line l in the primal plane remain preserved among their duals in the dual plane [3]. We conclude that any point $p \in slab(s)$ in the primal plane mapped to line p^* which intersect vertical segment $\overline{l^*r^*}$ in the dual plane. The opposite is also clear, the line p^* which intersect vertical segment $\overline{l^*r^*}$ mapped to point p in the primal plane, such that $p \in slab(s)$. □

The $k-$closest points to the endpoints of s are easy to obtain from the order-k Voronoi diagram of P. The maximum complexity of the order-k Voronoi diagram of a set of n point sites in the primal plane is $\Theta(k(n-k))$ [10]. Currently the best known algorithm for computing the order-k Voronoi diagram runs in $O(n \log^3 n + k(n-k))$ time [11] with query time $O(k + \log n)$.

We therefore concentrate on finding the closest points in the *slab of* s. As we mentioned in section 2 in order to find closest point to line we need to draw vertical line from point s^* (this is line in primal plane that contains the entire segment s), the first line we meet in the arrangement is the closest point to this line. But now we have a segment and not a line, so we need to take into consideration only the points that are in the *slab of* s. From Lemma 2 this are all the lines from the arrangement which interest vertical segment $\overline{l^*r^*}$. We will use cutting trees to efficiently find all the lines which intersect the vertical segment $\overline{l^*r^*}$.

Cutting Trees

Cutting trees have been used in the context of range-searching [6]. We explore its applicability to the problem at hand.

Let L a set of n lines in the plane, and let r be a parameter with $1 \leq r \leq n$. A line is said to cross a triangle if it intersects the interior of the triangle. A $(1/r) - cutting$ for L is a set $\Xi(L) := \{t_1, t_2, \ldots, t_m\}$ of possibly

unbounded triangles with disjoint interiors that together cover the plane, with the property that no triangle of the partitioning is crossed by more that n/r lines from L. The size of the cutting $\Xi(L)$ is the number of triangles it consists of.

The following important result by Chazelle [9] shows how to construct a $(1/r) - cutting$. These cuttings can be used to design a (multi-level) cutting tree for simplex range searching.

Theorem 2 *For any set L of n lines in the plane, and any parameter r with $1 \leq r \leq n$, a $(1/r) - cutting$ of size $O(r^2)$ exists. Moreover, such a cutting (with for each triangle in the cutting the subset of the lines from L that cross it) can be constructed with a deterministic algorithm that takes $O(nr)$ time. The resulting data structure has $O(\log n)$ query time and it uses $O(n^{2+\varepsilon})$ storage (for any given $\varepsilon > 0$).*

The following lemma is taken from [6] (chapter 16).

Lemma 3 *Let L be a set of n lines in the plane. Using cutting tree, the lines from L below a query point can be selected in $O(\log n)$ time in $O(\log n)$ canonical subsets. For any $\varepsilon > 0$, a cutting tree on L can be constructed that uses $O(n^{2+\varepsilon})$ storage.*

Preprocessing
Let P^* be a set of n lines in the dual plane created from n points set P. We build a two-level cutting tree on a set P^* for selecting the lines below a point r^* and above l^*. The tree is defined as follows.

- The set P^* is stored in a cutting tree T
- With each node v of the first-level tree T, we store its upper canonical subset in a second-level cutting tree T_v^{assoc}

The idea is that first level of the tree is used to select the lines below r^* in a number of canonical subsets. The associated structures (or, trees at the second level) storing the selected canonical subsets are then used to select the lines that lie above l^*. To the second level trees we associate the data structure from [2]. Here is time to recall that algorithm in [2] requires $O(n^2)$ time and $O(n^2/\log n)$ space for preprocessing, and it answers the query for an arbitrary line in $O(k + \log n)$ time.
So, at the end of preprocessing step we have three-level tree.

Lemma 4 *Let P^* be a set of n lines in the plane. Using a two-level cutting tree, the lines from P^* below a query point r^* and above l^* can be selected in $O(\log^2 n)$ time in $O(\log^2 n)$ canonical subsets. For any $\varepsilon > 0$, such a two-level cutting tree on P^* can be constructed that uses $O(n^{2+\varepsilon})$ storage.*

Proof. Let $Q(n)$ denote the query time in a two-level cutting tree for a set of n lines. The associated structures are one-level cutting trees, so the query time for the associated structures is $O(\log n)$ by Lemma 3. Hence, $Q(n)$ satisfies the recurrence:

$$Q(n) = \begin{cases} O(1) & \text{if n = 1,} \\ O(r^2) + O(\log n) + Q(n/r) & \text{if n > 1.} \end{cases}$$

This recurrence solves to $Q(n) = O(\log^2 n)$ for any constant $r > 1$.

Let $\varepsilon > 0$ be given. According to Lemma 3 we can construct the associated structures of the children of the root such that each of them uses $O(n^{2+\varepsilon})$ storage. Hence, the amount of storage used by the cutting tree, $M(n)$, satisfies

$$M(n) = \begin{cases} O(1) & \text{if n = 1,} \\ \sum_v [O(n^{2+\varepsilon} + M(n_v)] & \text{if n > 1,} \end{cases}$$

where we sum over all children v of the root of the tree. The number of children of the root is $O(r^2)$, and $n_v \leq n/r$ for each child v. It follows that, if r is large enough constant, the recurrence solves to $M(n) = O(n^{2+\varepsilon})$. □

Lemma 5 *The $k-$nearest points from slab(s) to segment s can be found in $O(k + \log^3 n)$ query time after preprocessing $O(n^{2+\varepsilon})$ time and space.*

Proof. ... □

It is possible to get rid of the $O(n^{\varepsilon})$ factor in the storage of cutting tree [12] and the fact that the data structure in [2] needs $O(n^2/\log n)$ space. So the total storage is $O(n^2)$ and preprocessing time is the same $O(n^{2+\varepsilon})$. But for the new structure the query time will be $O(k + \log^7 n)$.

References

[1] P. Mitra, B.B. Chaudhuri
Efficiently computing the closest point to a query line
Pattern Recognition Letters 19 (1998) 1027-1035

[2] Subhas C. Nandy, Sandip Das, Partha P. Goswami
An efficient k nearest neighbors searching algorithm for a query line
Theoritical Computer Science 299 (2003) 273-288

[3] D.T. Lee, Y.T. Ching
The power of geometric duality revisited
Inform. Process. Lett. 21 (1985) 117-122

[4] H. Edelsbrunner
Algorithms in Computational Geometry
Springer, Berlin 1987

[5] H. Edelsbrunner, L.J. Guibas, J. Stolfi
Optimal point location in monotone subdivision
SIAM J. Comput. 15 (1986) 317-340

[6] M. de Berg, M. van Kreveld, M.Overmars, O. Schwartzkopf
Computational Geometry: Algorithms and Applications
Springer-Verlag, 1997

[7] K. Mulmuley
Computational Geometry: An Introduction through Randomized Algorithms
Prentice-Hall, Englewood Chiffs, NJ

[8] H. Everett, J.-M. Robert, M. van Kreveld
An optimal algorithm for the $(\leq k)$-levels, with applications to separation and transversal problems
Internat. J. Comput. Geom. Appl., 6 (1996), pp. 247-261

[9] B. Chazelle
Cutting hyperplanes for divide-and-conquer
Discrete Comput. Geom., 9(2):145-158, 1993

[10] D. T. Lee
On k-nearest neighbor Voronoi diagrams in the plane
IEEE Trans. Comput., C-31:478-487, 1982

[11] Pankaj. K. Agarwal, M. de Berg, J. Matoušek, and O. Schwartzkopf
Constructing levels in arrangements and higher order Voronoi diagrams
In Proc. 10th Annu. ACM Sympos. Comput. Geom., pages 67-75, 1994

[12] J. Matoušek
Range searching with efficient hierarchical cuttings
Discrete Comput. Geom., 10(2):157-182, 1993