Implicit point location in arrangements of line segments, with an application to motion planning

P.K. Agarwal, M. van Kreveld

RUU-CS-92-15
April 1992

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Implicit point location in arrangements of line segments, with an application to motion planning

P.K. Agarwal, M. van Kreveld

Technical Report RUU-CS-92-15
April 1992

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Implicit Point Location in Arrangements of Line Segments, with an Application to Motion Planning*

Pankaj K. Agarwal† Marc van Kreveld‡

Abstract

Let S be a set of n (possibly intersecting) line segments in the plane. We show that the arrangement of S can be stored implicitly into a data structure of size $O(n \log^2 n)$ so that the following query can be answered in time $O(n^{1/2} \log^2 n)$: Given two query points, determine whether they lie in the same face of the arrangement of S and, if so, return a path between them that lies within the face. This version of the *implicit point location* problem is motivated by the following motion planning problem: Given a polygonal robot R with m vertices and a planar region bounded by polygonal obstacles with n vertices in total, preprocess them into a data structure so that, given initial and final positions of R, one can quickly determine whether there exists a continuous collision-free translational motion of R from the initial to the final position. We show that such a query can be answered in time $O((mn)^{1/2} \log^2 mn)$ using $O(mn \log^2 mn)$ storage.

*Part of this work was done while the second author was visiting the first author at Duke University on a grant of the Netherlands Organization for Scientific Research (N.W.O.). The research of the first author was supported by National Science Foundation Grant CCR-91-06514. The research of the second author was supported by the ESPRIT Basic Research Action No. 3075 (project ALCOM).

†Computer Science Department, Duke University, Durham, NC 27706, U.S.A.

‡Department of Computer Science, Utrecht University, P.O.Box 80.089, 3508 TB Utrecht, the Netherlands.

1 Introduction

Point location in a subdivision in $\mathbb{R}^d$ (i.e., preprocess a given subdivision so that the face of the subdivision containing a query point can be found efficiently) is one of the fundamental problems in computational geometry. In two dimensions, the problem has been studied for a long time and several optimal algorithms are known. A planar subdivision with n edges can be preprocessed in time $O(n \log n)$ into a linear size data structure so that a query can be answered in time $O(\log n)$ [11, 26]. If all the faces of the subdivision are simply connected, the preprocessing time can be reduced to $O(n)$ [5]. The problem becomes considerably more difficult in higher dimensions. Even in three dimensions no efficient algorithm was known until very recently [22, 25], and, in higher dimensions, efficient solutions are known only in some special cases, e.g., when the subdivision is formed by an arrangement of hyperplanes [7] or of algebraic surfaces [6], or when the subdivision is a convex polyhedron [27].

All of the above algorithms assume that the explicit representation of the subdivision is given, and most of them (more or less) store the entire subdivision in the data structure. For example, in the plane one assumes that all vertices, edges and faces of the planar subdivision are given explicitly. In several applications, however, the subdivision is induced by certain geometric objects, which may increase the complexity of the subdivision substantially. In these cases, the subdivision may be represented more compactly by defining it implicitly in terms of the underlying objects rather than describing it explicitly. It is an interesting question whether we really need to store the entire subdivision explicitly, or we can answer a query efficiently without storing the entire subdivision. For example, a planar subdivision induced by an arrangement of n lines can be described implicitly by giving the equations of the lines. But if we represent the entire arrangement explicitly, we need to specify $\Omega(n^2)$ features. The question in this case is whether we can answer a query using a data structure of size $O(n)$ instead of $O(n^2)$.

Motivated by various applications, researchers have studied the implicit point location problem in the last few years [1, 10, 12]. If the subdivision is stored implicitly, then the question is what information is required about the face that contains the query point. There are several possibilities depending on the application. In certain applications, it suffices to determine whether the query point lies inside any of the geometric objects, or to return an object that contains the query point. In some of these cases, the range searching structures can be modified to answer a point location query, e.g. [1, 12, 20]. On the other hand, in some applications one wants to report all the edges of the face containing the query point. Edelsbrunner et al. [10] have shown (see also [2]) that a set $\mathcal{L}$ of lines in the plane can be preprocessed into a data structure of size $O(n \log^2 n)$ so that the face in the arrangement of $\mathcal{L}$, containing a

query point, can be reported in time $O(n^{1/2}\log^3 n + k)$, where k is the number of edges in the face. They also showed that a set $\mathcal{S}$ of n segments in the plane can be processed into a data structure of size $O(n^{4/3}\log^{O(1)} n)$ so that a point location query can be answered in time $O(n^{1/3}\log^3 n + k)$. But their approach fails to give a data structure of roughly linear size for segments. It was an open question whether one could develop a (close to) linear size data structure for point location queries in sets of line segments, so that a query can be answered in roughly $O(\sqrt{n})$ time.

In this paper we develop a data structure of size $O(n\log^2 n)$ to preprocess a set of segments so that the following point location query can be answered in time $O(n^{1/2}\log^2 n)$: Given two points, determine whether they lie in the same face of the arrangement of $\mathcal{S}$. This query problem may be somewhat easier than the one considered in [10], because their solution can be modified to solve this problem. But we are not aware of any close-to-linear size data structure for point location in an arrangement of line segments. Another advantage of our algorithm is that it admits a storage/query time tradeoff. For a given parameter $n\log^2 n \leq s \leq n^2$, we can construct a data structure of size $O(s)$ so that a query can be answered in time $O(\frac{n}{\sqrt{s}}\log^2\frac{n}{\sqrt{s}} + \log n)$.

Our data structure is motivated by the following motion planning problem. We are given a set $\mathcal{O}$ of polygonal obstacles with a total of n vertices and a polygonal robot R (not necessarily simply connected) with m vertices. We want to preprocess $\mathcal{O}$ and R so that for given initial and final positions σ_I and σ_F of R, respectively, we can determine whether R can be translated from σ_I to σ_F without hitting any obstacle. This problem can be reduced to determining whether two points lie in the same face of a planar subdivision formed by $O(mn)$ segments; see Section 6 for details.

One possible solution to the above motion planning problem is to compute the entire arrangement of these segments and preprocess it for answering point location queries. Then we can answer a query in $O(\log mn)$ time, but the storage required is $\Omega(m^2n^2)$ in the worst case. On the other hand, we can answer a query in time $O(mn\log mn)$ and $O(mn)$ storage using the algorithm of Leven and Sharir [16]. So the challenge is whether we can answer a query efficiently (in sublinear time) if we allow only roughly $O(mn)$ space. If R is a convex polygon, then a query can be answered in $O(\log^2(m+n))$ time using $O(m+n)$ space [15, 28]. But no efficient data structure was known for nonconvex polygons. Using our point location structure, we can preprocess R and $\mathcal{O}$ into an $O(mn\log^2 mn)$ size data structure, so that the above motion planning query can be answered in time $O((mn)^{1/2}\log^2 mn)$. It can also return a (translational) collision-free path between σ_I and σ_F, if there exists one. Furthermore, one can reduce the query time by allowing more space.

This paper is organized as follows. In Section 2 we describe certain properties of

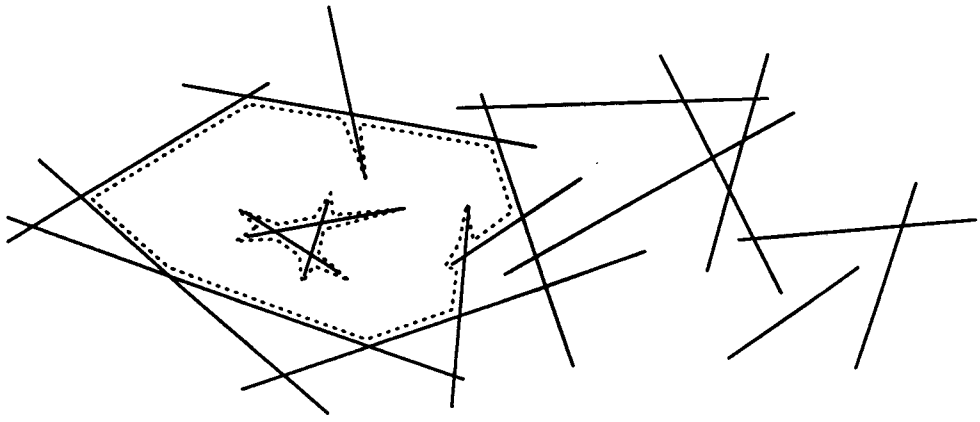

Figure 1: An arrangement of line segments with convex and nonconvex faces.

nonconvex faces of arrangements of segments and the basic ingredients of our solution. Section 3 presents the data structure. In Section 4 we show how to adapt our solution to finding a path between two query points, if they lie in the same face. We describe the space/query-time tradeoff in Section 5 and discuss the application to the motion planning problem in Section 6.

2 Arrangements of Line Segments

Let $\mathcal{S}$ be a set of n line segments in the plane. The arrangement $\mathcal{A}(\mathcal{S})$ consists of convex faces and of nonconvex faces; some of the nonconvex faces may not even be simply connected. It is easy to verify that each nonconvex face contains an endpoint of some segment of $\mathcal{S}$, so there are at most $2n$ nonconvex faces in $\mathcal{A}(\mathcal{S})$. Let $\mathcal{N}$ denote the collection of nonconvex faces in $\mathcal{A}(\mathcal{S})$. Aronov et al. [3] showed that the maximum complexity of m distinct faces in an arrangement of n segments is $O(n^{2/3}m^{2/3}+n\alpha(n)+m\log n)$, where $\alpha(n)$ is the extremely slowly growing functional inverse of Ackermann's function. This bound immediately implies that the total complexity of all nonconvex faces is $O(n^{4/3})$. Moreover, since every reflex vertex is an endpoint of some segment in $\mathcal{S}$, there are only $O(n)$ reflex vertices in $\mathcal{A}(\mathcal{S})$.

Due to technical reasons, we consider each segment two-sided, i.e., we expand each edge into an infinitely thin rectangle. As a result, if both sides of an edge of the arrangement appear in the same face, the two sides are treated as distinct edges and if a face has a vertex of degree $d > 2$, then $d-1$ copies of this vertex are created

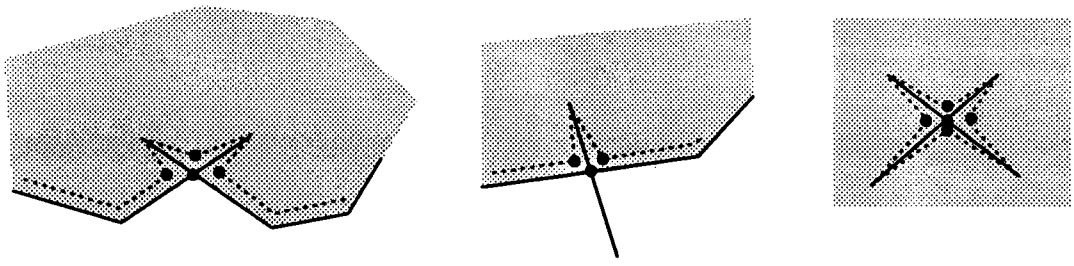

Figure 2: Expanding each edge and copying the vertices.

to ensure that each vertex has degree two (see Figure 2). Let c be a nonconvex face of $\mathcal{A}(\mathcal{S})$. The boundary of c may consist of several connected components. Since we consider the edges two-sided and copy the vertices of degree greater than 2, each component of the boundary of c is a simple cycle (see Figure 2). Let $V = (v_1, \ldots, v_k)$ be such a cycle, and let $W_V = (w_1, \ldots, w_j)$ be the set of reflex vertices of V plus the vertices of V whose y-coordinates are locally minimum or maximum, ordered in the clockwise direction. See the left face of Figure 3 (for the sake of clarity we have not expanded the edges of the face).

It is easily seen that there is at least one reflex vertex between two consecutive locally maximum vertices of V, and the same holds for locally minimum vertices. Hence, the number of vertices in W is proportional to the number of endpoints appearing in V. We thus have

$$\sum_{c \in \mathcal{N}} \sum_{V \text{cycle of } c} |V| = O(n^{4/3}) \qquad \text{and} \qquad \sum_{c \in \mathcal{N}} \sum_{V \text{cycle of } c} |W_V| = O(n). \qquad (1)$$

Without loss of generality, assume that $w_1 = v_1$. For each $1 \leq i \leq j$, let V_i denote the portion of V between w_i and w_{i+1} in the clockwise direction. By construction, each V_i is a y-monotone polygonal chain which, from w_i to w_{i+1}, only makes turns in clockwise direction. Let $\tilde{V}_i$ be the (geodesic) shortest path between w_i and w_{i+1} homotopic to V_i, that is, $\tilde{V}_i$ can be continuously deformed to V_j so that the path never leaves the face c and never crosses any segment of $\mathcal{S}$ (see the right face of Figure 3). We define the *reduced cycle* for V to be the cycle $\tilde{V}$ formed by concatenating $\tilde{V}_1, \tilde{V}_2, \ldots, \tilde{V}_j$. Notice that $\tilde{V}$ is not necessarily a simple cycle. Let M_i denote the polygon formed by V_i and $\tilde{V}_i$, which we will refer to as a *moon polygon*. M_i consists of a *convex* chain V_i and a *concave* chain $\tilde{V}_i$. These two chains meet only at the topmost and the bottommost vertices of M_i. Since $\tilde{V}_i$ is homotopic to V_i, the interior of M_i does not intersect any segment of $\mathcal{S}$, and furthermore, M_i is a y-monotone polygon.

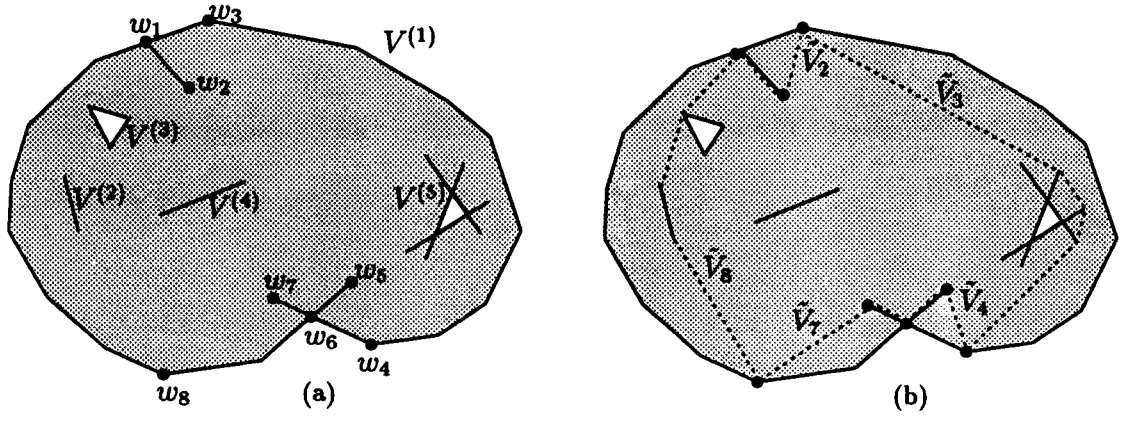

Figure 3: (a) a nonconvex face c with five connected components, $W_{V^{(1)}}$ are indicated; (b) the shortest paths of $\tilde{V}^{(1)}$ are shown dotted.

Note that some of the moon polygons may be degenerate in the sense that their interiors may be empty (see e.g. the moon polygon formed by V_5 and $\tilde{V}_5$ in Figure 3). We will discard such polygons. Let $\mathcal{M}_c$ denote the set of moon polygons of c with nonempty interiors.

Lemma 2.1 $\sum_{c \in \mathcal{N}} \sum_{V \text{ cycle of } c} |\tilde{V}| = O(n)$.

Proof: Let $\tilde{V}$ be a reduced formed by concatenating $\tilde{V}_1, \tilde{V}_2, \ldots, \tilde{V}_j$ where $j = |W_V|$. Then $|\tilde{V}| = \sum_{i=1}^{j} |\tilde{V}_i| - j$; the second term is due to the fact that, for each i, the common endpoint of $\tilde{V}_i$ and $\tilde{V}_{i+1}$ should be counted only once. We call a vertex $p \in \tilde{V}_i$ an *interior* vertex if $p \notin \{w_i, w_{i+1}\}$. Then $|\tilde{V}_i|$ is 2 plus the number of interior vertices in $\tilde{V}_i$. Thus

$$|\tilde{V}| = |W_V| + \sum_{i=1}^{j} (\#\text{ interior vertices in } \tilde{V}_i).$$

We claim that a vertex can appear as an interior vertex of at most one $\tilde{V}_i$. Suppose, on the contrary, there is a vertex p that appears as an interior vertex of two reduced chains $\tilde{V}_l$ and $\tilde{V}_i$. Since the interiors of M_i and M_l are disjoint, either V_i or V_l, say V_i, passes through p. But the only intersection points of $\tilde{V}_i$ and $\tilde{V}_{i+1}$ are w_i and w_{i+1}, which contradicts the assumption that p is an interior vertex of $\tilde{V}_i$. Since each interior

vertex is a reflex vertex, the total number of interior vertices over all reduced cycles is $O(n)$, therefore

$$\sum_{c\in\mathcal{N}} \sum_{V\text{cycle of } c} |\tilde{V}| = O(n) + \sum_{c\in\mathcal{N}} \sum_{V\text{cycle of } c} |W_V| = O(n)$$

where the last equality follows from (1). □

Let $V^{(1)},\ldots,V^{(u)}$ be the cycles of the boundary of c. For any cycle, we define its interior to be the (open) bounded region enclosed by it, and the exterior is the (open) unbounded region. If the face c is unbounded, then c lies in the exterior of all $V^{(1)},\ldots,V^{(u)}$, and we define B_c to be the portion of c lying in the exterior of all the reduced cycles $\tilde{V}^{(1)},\ldots\tilde{V}^{(h)}$. If c is bounded, then we can assume that c lies in the interior of $V^{(1)}$ and in the exterior of all other cycles. We now define B_c to be the portion of c that lies in the interior of $\tilde{V}^{(1)}$ and in the exterior of the other reduced cycles. In order to ensure that B_c is connected and does not contain any segment of $\mathcal{S}$ in its interior, we add the reduced cycles of c to the boundary of B_c. We will refer to B_c as the *body polygon* of c. Thus every nonconvex face c is partitioned into one body polygon B_c and a set $\mathcal{M}_c$ of moon polygons. Body polygons are not necessarily simple. See Figure 4 for the body polygon and moon polygons of the face of Figure 3. Let $\mathcal{B} = \{B_c \mid c \in \mathcal{N}\}$, $\mathcal{M} = \bigcup_{c\in\mathcal{N}} \mathcal{M}_c$, and let $\mathcal{E}$ be the set of edges in the concave chains of moon polygons. Since the edges of B_c and the segments of $\mathcal{E}$ are the edges of the reduced cycles, we have

$$\sum_{c\in\mathcal{N}} |B_c| = O(n), \quad |\mathcal{E}| = O(n) \quad \text{and} \quad \sum_{M\in\mathcal{M}} |M| = O(n^{4/3}). \tag{2}$$

3 The Data Structure

Recall that the query problem we are aiming to solve is: 'Given two query points p and q, do they lie in the same face of $\mathcal{A}(\mathcal{S})$?' If p and q lie in the same *convex* face of $\mathcal{A}(\mathcal{S})$, then the segment $\overline{pq}$ does not intersect any segment of $\mathcal{S}$. Conversely, if $\overline{pq}$ does not intersect any segment of $\mathcal{S}$, then p and q lie in the same (not necessarily convex) face of $\mathcal{A}(\mathcal{S})$. Consequently, we can determine whether p and q lie in the same convex face of $\mathcal{A}(\mathcal{S})$ by preprocessing $\mathcal{S}$ for segment intersection detection queries, i.e., given a query segment e determine whether e intersects any segment of $\mathcal{S}$. Alternatively, we can answer a segment intersection query by preprocessing $\mathcal{S}$ into a ray shooting structure as follows. Determine the first intersection point of $\mathcal{S}$ and the ray emanating

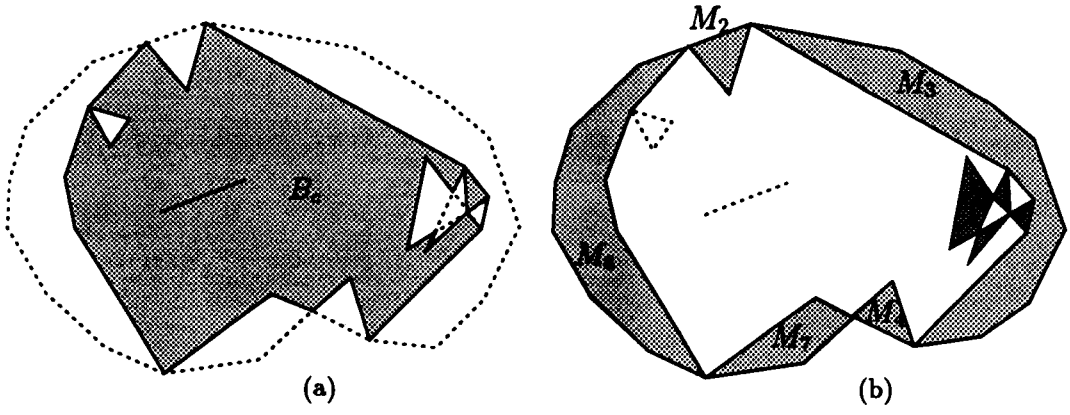

Figure 4: (a) The body polygon, and (b) the nonempty moon polygons of the face of 3; M_2, M_3, M_4, M_7, M_8 are the nonempty moon polygons of $V^{(1)}$, and the dark shaded polygons are nonempty moon polygons of $V^{(5)}$.

from p in direction $\vec{pq}$. If it lies beyond q, then $\overline{pq}$ does not intersect any segment of $\mathcal{S}$.

Next, we describe another structure that can determine whether p and q lie in the same nonconvex face of $\mathcal{A}(\mathcal{S})$. If a point p lies in a nonconvex face c, then it lies either in B_c or in one of the moon polygons of M_c. Therefore, to determine the face of $\mathcal{N}$ containing p (if any), we will do point location in $\mathcal{B}$, the set of all body polygons, and in $\mathcal{M}$, the set of all nonempty moon polygons. Since the polygons in $\mathcal{B}$ are pairwise disjoint and the total complexity of $\mathcal{B}$ is $O(n)$, we can apply any efficient point location structure to preprocess $\mathcal{B}$, see [11, 26]. But the complexity of $\mathcal{M}$ is $O(n^{4/3})$, so we cannot preprocess it explicitly if we allow only close-to-linear storage. The following lemma suggests how to preprocess $\mathcal{M}$ implicitly.

Let e be an edge of a moon polygon M. We distinguish between two sides of e; the one that lies in the interior (exterior) of M is denoted by e^+ (resp. e^-). If we think of e as expanded into a very thin rectangle and of e^+, e^- as denoting the sides of the rectangles that lie inside and outside M, respectively, then a ray ρ hits e from the *inside* if it first intersects e^+ and then e^-.

Lemma 3.1 *A point p lies inside a moon polygon M if and only if one of the two horizontal rays emanating from p intersects an edge of a concave chain of M from the inside before intersecting any other segment of $\mathcal{S} \cup \mathcal{E}$.*

Proof: Follows immediately from the facts that the interior of M does not intersect any segment of $\mathcal{S}$, and that M is a y-monotone polygon. □

In view of the above lemma, we can determine the moon polygon that contains a query point by answering ray shooting queries. We will preprocess $\mathcal{S}$ and $\mathcal{E}$ separately. Since we query $\mathcal{E}$ with only horizontal rays, we need a structure that, given a query point, can determine the segments that lie immediately to its left and to its right. Recall that the segments in $\mathcal{E}$ are nonintersecting except at their endpoint, so we can use a persistent data structure to answer ray shooting queries [26].

In summary, we construct a data structure that consists of the following three substructures.

1. A data structure Ψ_1 for efficient ray shooting in the set $\mathcal{S}$ of n line segments in the plane [1, 8].

2. A data structure Ψ_2 for efficient (horizontal) ray shooting in the set $\mathcal{E}$ of $O(n)$ edges of the nonempty moon polygons [26]. For each edge e of $\mathcal{E}$ we also distinguish which of its sides is e^+, and we store with e in which nonconvex face it lies.

3. A data structure Ψ_3 for efficient point location in the planar subdivision induced by $\mathcal{B}$.

3.1 Answering a query

In this subsection we describe how the above data structure is used to decide whether two query points p and q lie in the same face of $\mathcal{A}(\mathcal{S})$. A query is answered in three steps.

Let ρ be the ray emanating from p in the direction $\vec{pq}$. We query Ψ_1 with ρ and determine the first intersection point of ρ and $\mathcal{S}$, if there exists one. If ρ does not intersect $\mathcal{S}$ or the first intersection point lies beyond q, then p and q lie in the same face of $\mathcal{A}(\mathcal{S})$ and we are done. Otherwise, we can conclude that p and q do not lie in the same convex face of $\mathcal{A}(\mathcal{S})$.

Next, we determine in $O(\log n)$ time by a point location query in Ψ_3 whether p lies in any body polygon. If $p \in B_c$, then we can conclude that p lies in the nonconvex face c. Otherwise, p lies either in a moon polygon or in a convex face. Let h^- and h^+ be the leftward and rightward directed horizontal rays emanating from p. By Lemma 3.1, p lies in a moon polygon if and only if one of h^- and h^+ intersects a

segment of $\mathcal{E}$ from the inside before intersecting any other segment of $\mathcal{S} \cup \mathcal{E}$. Thus by querying Ψ_1 and Ψ_2 with both h^- and h^+ (four ray shooting queries), we can determine whether p lies in a moon polygon. If the answer is 'yes', the edge $e \in \mathcal{E}$ that is hit by the ray also gives us the nonconvex face c that contains p. If p does not lie in a moon polygon, then p lies in some convex face of $\mathcal{A}(\mathcal{S})$. Since we already know that p and q do not lie in the same convex face, we can conclude that p and q do not lie in the same face of $\mathcal{A}(\mathcal{S})$.

Finally, if p lies in a nonconvex face c, we repeat the above steps for q and determine whether q lies in a nonconvex face of $\mathcal{A}(\mathcal{S})$. If the answer is 'yes', then the label c' of the face that contains q is returned, so we can check whether it is the same as the label of the one containing p. Otherwise, p and q do not lie in the same face of $\mathcal{A}(\mathcal{S})$. This finishes the description of the query answering procedure.

In Section 4 we show how to adapt the data structure and the query algorithm so that if p and q lie in the same face, then we can also report a path between them that does not cross any segment of $\mathcal{S}$

3.2 The preprocessing

Next, we show how to construct the above data structure in $O(n^{4/3}\log^2 n)$ time. Ψ_1 stores $\mathcal{S}$ using $O(n\log^2 n)$ space, so that a ray shooting query can be answered in time $O(n^{1/2}\log^2 n)$. Ψ_1 can be constructed using the algorithm of Agarwal [1] or of Cheng and Janardan [8]. Both of these algorithms are based on a data structure called 'spanning paths with low stabbing number' (i.e., a spanning path of n points such that every line intersects at most $O(\sqrt{n})$ edges of the path). The preprocessing time of these algorithms is bounded by the time required for constructing such a spanning path. The best known algorithm for constructing such a spanning path of n points in the plane is $O(n^{4/3}\log^2 n)$ [18, 20].[1]

The structures Ψ_2 and Ψ_3 require additional work. First, we compute the set of nonconvex faces in $\mathcal{A}(\mathcal{S})$. Since each nonconvex face contains at least one endpoint, these $O(n)$ faces can be computed in time $O(n^{4/3}\log^2 n)$, using the algorithm described in [2].[2]

Let c be a nonconvex face. For each cycle V of its boundary, we determine the subset $W_V = \{w_1, \ldots, w_j\}$ of reflex vertices plus the vertices whose y-coordinates are

[1] In [18], Matoušek gave an algorithm for computing a spanning path with low stabbing whose time complexity was $O(n^{3/2}\log^2 n)$. But its running time can be improved to $O(n^{4/3}\log^2 n)$ using a result described in [20].

[2] The algorithm described in [2] is slightly worse than the stated bound, but in combination with the more recent results of Matoušek [19] the bound follows.

locally minimum or maximum. Next, for each $1 \leq i \leq j$, we apply the algorithm of de Berg et al. [4] or of Hershberger and Snoeyink [14] to compute the (geodesic) shortest path $\tilde{V}_i$ between w_i and w_{i+1} homotopic to V_i (relative to c). Roughly speaking, both of these algorithm triangulate the nonconvex face, traverse all triangles adjacent to V_i, and maintain the shortest path $\tilde{V}_i$. This gives the reduced cycle $\tilde{V}$ for V. The total time spent in computing shortest paths over all cycles of c is proportional to the number of edges in c, because each triangle is traversed only a constant number of times; see [4, 14] for a proof.

After having computed reduced cycles for all components of the boundary of c, the body polygon B_c and the set of (nonempty) moon polygons M_c can be easily obtained. Let M be a nonempty moon polygon of $\mathcal{M}_c$. For each edge e of its concave chain, we determine which side of e is e^+. We also record the information that e belongs to the face c.

Repeating these steps for all nonconvex faces, we obtain the sets $\mathcal{B}$ and $\mathcal{E}$. We preprocess the planar subdivision induced by $\mathcal{B}$ into a point location data structure, Ψ_3, using the algorithm of Sarnak and Tarjan [26]. Finally, we preprocess $\mathcal{E}$ into a ray shooting data structure Ψ_2. Since $|\mathcal{E}| = O(n)$, Ψ_2 requires $O(n \log^2 n)$ space and $O(n^{4/3} \log^2 n)$ prepocessing time. Putting everything together, we can conclude

Lemma 3.2 *Given a set $\mathcal{S}$ of n segments in the plane, we can preprocess it in time $O(n^{4/3} \log^2 n)$ into a data structure of size $O(n \log^2 n)$ so that we can determine in $O(n^{1/2} \log^2 n)$ whether two query points lie in the same face of $\mathcal{A}(\mathcal{S})$.*

4 Reporting a Path

The data structure and the query procedure described above can be easily modified to compute—in case p and q lie in the same face c—a path Π_{pq} between them that does not intersect any segment of $\mathcal{S}$ properly.

For the sake of simplicity, we assume that all the faces of $\mathcal{N}$ are bounded. This assumption does not restrict the problem, because we can enclose the entire arrangement by a big rectangle and the points lying outside the rectangle can be handled easily. In order to return a path, we further preprocess each body polygon B_c. We triangulate B_c using an standard algorithm, e.g., using a sweep-line algorithm [24]. Let B_c^* denote the resulting subdivision. Ψ_3 can be modified so that it returns the triangle containing a query point if it lies in one of the body polygons. We view B_c^* as a connected planar graph whose nodes are the vertices of B_c and whose edges are the edges of B_c^*. We compute a spanning tree T of B_c^*, choose an arbitrary node to

be the root t, and direct all edges of T towards the root t, see Figure 5.

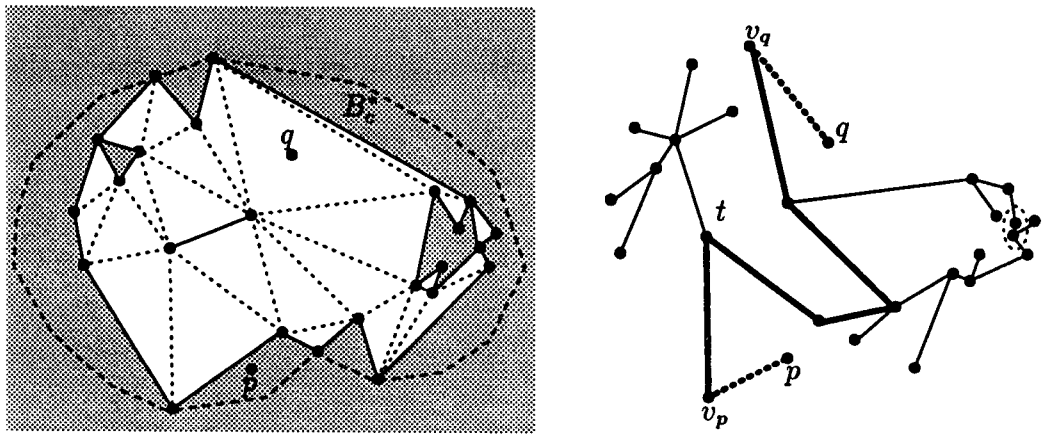

Figure 5: (a) The triangulated subdivision B_c^* of the body polygon of Figure 4 and two query points, (b) solid lines denote the spanning tree T, and bold lines denote the path $\Pi_{p,q}$; the vertices inside the dotted oval are two copies of a single vertex of c.

The query algorithm is adapted as follows. If the segment $\overline{pq}$ lies in c, we return the segment itself as the desired path Π_{pq}. Otherwise, we do the following. If p lies in a body polygon B_c for some nonconvex cell c, Ψ_3 returns the triangle Δ of B_c that contains p. In this case we define v_p to be a vertex of Δ, say, the topmost vertex. If, on the other hand, p lies in a moon polygon, then Ψ_2 returns the edge e (of the concave chain) that one of the two horizontal rays, emanating from p, intersects from the inside. Now we define v_p to be one of the endpoints of e, say, the lower endpoint. The point v_p can be computed by spending $O(1)$ additional time. Similarly, we compute v_q for the point q. Observe that both v_p and v_q are vertices of B_c and thus nodes of T. $\Pi_{p,q}$, the path between p and q that we return, consists of four parts: the segment $\overline{pv_p}$, the path from v_p to the root t in T, the path from t to v_q in T, and the segment $\overline{v_q q}$. The path from v_p (resp. v_q) to t can be computed by following the edges of T from v_p to t. By construction, neither $\overline{pv_p}$, $\overline{v_q q}$, nor any of the edges of T cross any segment of $\mathcal{S}$. Furthermore, since each edge of c is considered two sided and the vertices of degree > 2 are split, it is easily seen that the we do not cross any segment of $\mathcal{S}$ as we cross a node of T. Thus, $\Pi_{p,q}$ does not cross any segment of $\mathcal{S}$. Note that $\Pi_{p,q}$ may touch the segments of $\mathcal{S}$.

The additional preprocessing required for reporting a path involves triangulating

the body polygons and computing T. Since $\sum_c |B_c| = O(n)$, these operations can be performed in $O(n \log n)$ time using any standard algorithm for these problems, see e.g. [21, 24]. We thus have

Theorem 4.1 *For a set $\mathcal{S}$ of n line segments in the plane, there exists a data structure of size $O(n \log^2 n)$, such that it takes $O(n^{1/2} \log^2 n)$ time to decide whether two query points lie in the same face of the arrangement $\mathcal{A}(\mathcal{S})$. If two query points lie in the same face, a path between them that consists of k segments can be reported in $O(k)$ additional time. The preprocessing time of the structure is $O(n^{4/3} \log^2 n)$.*

Remark 4.2: The above data structure can be modified to report a path that does not touch any segment of $\mathcal{S}$. One possibility is compute the Voronoi diagram of B_c and compute a spanning tree of the Voronoi diagram. This approach, called 'retraction' has been used earlier in motion planning algorithms, e.g., see [23] for details.

5 Space/Query-Time Tradeoff

We now describe how to reduce the query time by allowing more space. The basic approach is similar to the one used in [1] for answering ray shooting queries, therefore we will only describe the main idea.

Let $r \leq n$ be some fixed parameter. A $\frac{1}{r}$-cutting of $\mathcal{S}$ is the set of the triangles of Ξ with pairwise disjoint interiors, such that it covers the entire plane, and that the interior of each triangle intersects at most $\frac{n}{r}$ segments of $\mathcal{S}$. It is known that there exists a $\frac{1}{r}$-cutting of size $O(r^2)$ [19]. Let Ξ be such a $\frac{1}{r}$-cutting. For each triangle $\Delta \in \Xi$, let $\mathcal{S}_\Delta \subseteq \mathcal{S}$ denote the set of segments that intersect the interior of Δ. For the sake of simplicity, we assume that the segments of $\mathcal{S}_\Delta$ are clipped to within Δ. Let f_Δ denote the unbounded face of $\mathcal{A}(\mathcal{S}_\Delta)$.

If a face c of $\mathcal{A}(\mathcal{S})$ does not intersect the boundary of any triangle in Ξ, then it lies completely inside one triangle Δ, in which case c is a face of $\mathcal{A}(\mathcal{S}_\Delta)$. On the other hand, if c intersects the boundary of some triangle, then c is a connected component of $\bigcup_{\Delta \in \Xi} \Delta \cap f_\Delta$. Let C denote the set of faces in $\mathcal{A}(\mathcal{S})$ that intersect the boundary of some triangle. C can be constructed by first computing f_Δ for each triangle Δ, and then gluing them together. By a result of Guibas et al. [13], f_Δ has $O(\frac{n}{r}\alpha(\frac{n}{r}))$ edges, so the total complexity of C is $O(r^2) \times O(\frac{n}{r}\alpha(\frac{n}{r})) = O(nr\alpha(\frac{n}{r}))$.

The overall data structure now consists of the following parts:

1. A point location structure for Ξ, where Ξ is a $\frac{1}{r}$-cutting of $\mathcal{S}$.

2. A point location structure for C, where C is the set of faces of $\mathcal{A}(\mathcal{S})$ that intersect the boundary of some triangle $\Delta \in \Xi$.

3. A structure, as described in Section 3, on $\mathcal{S}_\Delta$ for each $\Delta \in \Xi$.

The cutting Ξ and the subsets $\mathcal{S}_\Delta$, for each $\Delta \in \Xi$, can be computed in $O(nr)$ time by the algorithm of Matoušek [19], and Ξ can be preprocessed for point location queries in $O(r^2 \log r)$ time [11, 26]. Next, using the algorithm of Guibas et al. [13], f_Δ can be computed in time $O(\frac{n}{r}\alpha(\frac{n}{r})\log^2 \frac{n}{r})$. Therefore, C can be computed and preprocessed for point location in time $O(nr\alpha(n)\log^2 \frac{n}{r})$. Finally, by Theorem 4.1, the total time spent in preprocessing $\mathcal{S}_\Delta$ over all $\Delta \in \Xi$ is $O((\frac{n}{r})^{4/3}\log^2 \frac{n}{r}) \times O(r^2) = O(n^{4/3}r^{2/3}\log^2 \frac{n}{r})$. If we assume that $r \leq n/\alpha^3(n)$ (which is the case, see below), then the total preprocessing time is $O(n^{4/3}r^{2/3}\log^2 \frac{n}{r})$. A similar analysis shows that the total space required by the structure is $O(nr\log^2 \frac{n}{r})$.

Next, we explain how to answer a point location query. Let p and q be two query points, and let c_p and c_q denote the face of $\mathcal{A}(\mathcal{S})$ containing p and q, respectively. We first determine, in $O(\log n)$ time, by locating p in C whether $c_p \in C$. If the answer is 'yes', we locate q in C to determine if $c_p = c_q$. The points p and q lie in the same face of $\mathcal{A}(\mathcal{S})$ if and only if $c_p = c_q$.

If $c_p \notin C$, then c_p lies completely inside a single triangle Δ, and it is a bounded face of $\mathcal{A}(\mathcal{S}_\Delta)$. We therefore locate p and q in Ξ and determine the triangles containing them. If they are different, then obviously c_p and c_q are different. If, on the other hand, p and q lie in the same triangle Δ, we query the structure constructed on $\mathcal{S}_\Delta$ with p and q in order to determine whether they lie in the same face of $\mathcal{A}(\mathcal{S}_\Delta)$.

The correctness of the algorithm is obvious. As for the query time, the first two steps require only $O(\log n)$ query time, and, by Theorem 4.1, the last step requires $O\left(\left(\frac{n}{r}\right)^{1/2}\log^2 \frac{n}{r}\right)$ time. Thus, choosing $r = \frac{s}{n\log^2(n/\sqrt{s})}$, we obtain:

Theorem 5.1 *Given a set $\mathcal{S}$ of n segments in the plane and a parameter $n\log^2 n \leq s \leq n^2$, one can preprocess $\mathcal{S}$ in time $O(s^{2/3}n^{2/3}(\log^{2/3}\frac{n}{\sqrt{s}}+1))$ into a data structure of size $O(s)$, so that given two points p and q, we can determine in time $O(\frac{n}{\sqrt{s}}\log^2 \frac{n}{\sqrt{s}} + \log n)$ whether they lie in the same face of $\mathcal{A}(\mathcal{S})$.*

6 Application to Motion Planning

We now apply Theorem 4.1 to the following version of the motion planning problem: Let R be a polygonal body (not necessarily simply connected) with m ver-

tices, free to translate (but not to rotate) in a planar region bounded by a collection $\mathcal{O} = \{O_1, \ldots, O_k\}$ of polygonal obstacles with n vertices in total. We want to preprocess them into a data structure, so that, given initial and final positions[3] σ_I and σ_F, respectively, of R, we can quickly determine whether there exists a (purely translational) continuous collision free motion of R from σ_I to σ_F and, if so, return a path.

In the more general problem, R is also allowed to rotate, but then the problem becomes much harder. In most pragmatic applications, however, it is sufficient to find a purely translational motion, or a translational motion with at most one rotation (particularly when the environment is not cluttered with obstacles). This simplified version was first considered by Lozano-Pérez and Wesley [17] (see also [13, 16]). They observed that one can replace the problem by that of a collision-free path for a single point between σ_I and σ_F amidst *expanded obstacles* $K_i = O_i - R$, $i = 1, \ldots, k$, where $A - R$ denotes the Minkowski difference, which is defined as $\{p - q \mid p \in A, q \in R\}$; see [17] for details.

Hence R can be translated from σ_I to σ_F without hitting an obstacle if and only if σ_I and σ_F lie in the same connected component of $K^c = \left(\bigcup_{i=1}^k K_i\right)^c$. It is easy to check that K is bounded by a collection Γ of $O(mn)$ segments, each of which is of the form $e - p$, where e is an edge of some obstacle and p is a vertex of R, or vice versa. Each connected component of K^c is a face of $\mathcal{A}(\Gamma)$. However, each face of $\mathcal{A}(\Gamma)$ is not necessarily a component of K^c. The above motion planning problem reduces to determining whether σ_I and σ_F lie in the same face of $\mathcal{A}(\Gamma)$, which is a component of K^c.

A possible solution is to preprocess the faces of $\mathcal{A}(\Gamma)$ that lie in K^c for planar point location. If R is a convex polygon, then the complexity of these faces is $O(mn)$, so one can store them explicitly and can answer a query in time $O(\log mn)$. (In fact, as mentioned in the introduction, one can do even better.) But if R is nonconvex, then K^c can have as many as $\Omega(m^2n^2)$ edges in the worst case, so it is space consuming to store K^c explicitly. However, by Theorem 4.1, we can store $\mathcal{A}(\Gamma)$ implicitly in a data structure of size $O(mn \log^2 mn)$, so that a query can be answered in time $O(\sqrt{mn} \log^2 mn)$. We also have to ensure that the face of $\mathcal{A}(\mathcal{S})$ containing σ_I and σ_F is a component of K^c. Using a result of [1], we can preprocess Γ into a data structure of size $O(mn \log mn)$, so that we can determine in time $O(\sqrt{mn} \log mn)$ time whether a given placement of R is free with respect to $\mathcal{O}$. Hence, putting everything together, we obtain

[3]Here we assume that we have a standard placement of R, and that the origin coincides with a point p of (this placement of) R. We describe a placement of R by specifying the coordinates of the reference point p.

Theorem 6.1 *Let $\mathcal{O}$ be a set of obstacles in the plane, consisting of n vertices in total, and let R be a robot modeled by a simple m-gon. There exists an $O(mn \log^2 mn)$ size data structure which can be constructed in $O((mn)^{4/3} \log^2 mn)$ time, such that it takes $O(\sqrt{mn} \log^2 mn)$ time to decide whether the robot can be translated from a given position to another without colliding with any obstacle.*

7 Conclusions

The main result of this paper is to show that the arrangement of a set of n segments can be stored implicitly in a data structure of size $O(n \log^2 n)$, so that one can determine in time $O(\sqrt{n} \log^2 n)$ time whether the two query points lie in the same face of the arrangement. We applied it to obtain an efficient solution for a translational motion planning problem. We conclude by mentioning two open problems:

1. Can the preprocessing time be improved to roughly linear (within a polylogarithmic factor)?
2. Can the above data structure be dynamized efficiently?

Acknowledgement

The authors thank Mark Overmars for a suggestion that simplified our data structure and its description.

References

[1] Agarwal, P. K., Ray shooting and other applications of spanning trees with low stabbing number, *SIAM J. Comput.* **21** (1992), in press.

[2] Agarwal, P. K., Partitioning arrangements of lines II: Applications, *Discr. & Comp. Geometry* **5** (1990), pp. 533–573.

[3] Aronov, B., H. Edelsbrunner, L. Guibas and M. Sharir, Improved bounds on the complexity of many faces in arrangements of segments, *Combinatorica*, to appear.

[4] de Berg, M., H. Everett, and H. Wagener, Translation queries for sets of polygons, Tech. Rept. RUU-CS-91-30, Dept. of Comp. Science, Utrecht University, the Netherlands, 1991.

[5] Chazelle, B., Triangulating a simple polygon in linear time, *Discr. & Comp. Geometry* **6** (1991), pp. 485–524.

[6] Chazelle, B., H. Edelsbrunner, L. Guibas, M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Proc. 16th Int. Coll. Aut. Lang. Prog.*, Lect. Notes in Comp. Science **372** (1989), pp. 179–193.

[7] Chazelle, B., and J. Friedman, Point location among hyperplanes, manuscript, 1991.

[8] Cheng, S., and R. Janardan, Space-efficient ray-shooting and intersection searching: Algorithms, dynamization, and applications, *Proc. 2nd ACM-SIAM Symp. on Discrete Algorithms*, (1991), pp. 7–16.

[9] Clarkson, K. L., H. Edelsbrunner, L. J. Guibas, M. Sharir, and E. Welzl, Combinatorial complexity bounds for arrangements of curves and surfaces, *Discr. & Comp. Geometry* **5** (1990), pp. 99–162.

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

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

[12] Guibas, L., M. Overmars, and M. Sharir, Ray shooting, implicit point location, and related queries in arrangements of segments, Tech. Rept. 433, New York University, New York, 1989.

[13] Guibas, L., M. Sharir, and S. Sifrony, On the general motion planning problem with two degrees of freedom, *Discr. & Comp. Geometry* **4** (1989), pp. 491–522.

[14] Hershberger, J. and J. Snoeyink, Computing minimum length paths of a given homotopy class, *Proc. 2nd Workshop on Algo. Data Struct.* (1991), pp. 331–342.

[15] Kao, T., and D. Mount, An algorithm for computing compacted Voronoi diagrams defined by convex distance functions, *Proc. 3rd Canadian Conf. on Comp. Geometry* (1991), pp. 104–109.

[16] Leven, D., and M. Sharir, Planning a purely translational motion of a convex object in two-dimensional space using generalized Voronoi diagrams, *Discr. & Comp. Geometry* **2** (1987), pp. 9–31.

[17] Lozano-Pérez, T., and M. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Comm. ACM* **22** (1979), pp. 560–570.

[18] Matoušek, J., More on cutting arrangements and spanning trees with low stabbing number, Tech. Rept. B–90–2, Fachbereich Mathematik, Freie Universitaät, Berlin, 1990.

[19] Matoušek, J., Approximations and optimal geometric divide-and-conquer, *Proc. 23rd Ann. ACM Symp. Theory of Computing* (1991), pp. 506–511.

[20] Matoušek, J., Efficient partition trees, *Proc. 7th Ann. ACM Symp. on Comp. Geometry* (1991), pp. 1–9.

[21] Mehlhorn, K., *Data Structures and Algorithms 2: Graph Algorithms and NP-Completeness*, Springer-Verlag, Berlin, 1984.

[22] Mulmuley, K., Randomized multidimensional search trees: Further results in dynamic sampling, *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science* (1991), pp. 216–227.

[23] O'Dúnlaing, C.. and C. Yap, A "retraction" method for planning the motion of a disc, *J. Algorithms* **6** (1986), pp. 104–111.

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

[25] Preparata, F., R. Tamassia, Efficient point location in a convex spatial cell complex, *SIAM J. Comput.* **21** (1992), in press.

[26] Sarnak, N., and R. E. Tarjan, Planar point location using persistent search trees, *Comm. ACM* **29** (1986), pp. 669–679.

[27] Schwarzkopf, O., Ray shooting in convex polytopes, *Proc. 8th Ann. ACM Symp. on Comp. Geometry* (1992), to appear.

[28] Sifrony, S., A real nearly linear algorithm for translating a convex polygon, Tech. Rept. 476, Dept. Computer Science, New York University, 1989.