In SIAM J. on Computing 23 (1), pp. 203-221, 1994

On Collision-Free Placements of Simplices and the Closest Pair of Lines in 3-Space[1]

Marco Pellegrini
Department of Computer Science
King's College
London WC2R 2LS, U.K.
(m.pellegrini@oak.cc.kcl.ac.uk)

Abstract

We consider the problem of detecting efficiently whether a query simplex is *collision-free* among polyhedral obstacles. If n is the number of vertices, edges and faces of the polyhedral obstacles, and m is the amount of storage we allocate for the data structure ($n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$), we can solve collision-free placements queries for simplices in time $O(n^{1+\epsilon}/m^{1/4})$ for any $\epsilon > 0$, where the constants depend on ϵ. In order to solve this problem we develop data structures to detect on-line intersections of query halfplanes with sets of lines and segments.

Next, we consider some nearest-neighbor problems for objects in 3-space. Given a set of n lines in 3-space, the shortest vertical segment between any pair of lines is found in randomized expected time $O(n^{8/5+\epsilon})$ for every $\epsilon > 0$. The longest connecting vertical segment is found in time $O(n^{4/3+\epsilon})$. The shortest connecting segment is found in time $O(n^{5/3+\epsilon})$.

Key words. Collision-free placements, Plücker coordinates of lines, arrangements, point location, half-space range searching, parametric search, proximity, 3-dimensional space.

AMS(MOS) subject classifications. 68U05.

1 Introduction

1.1 Collision-free placements of simplices

Detecting intersections of objects is a basic problem in computational geometry [PS85] [Ede87] [Meh84]. While intersection problems on the plane are more or less solved, for their

[1]A preliminary version of this paper appeared with the title "Incidence and nearest-neighbor problems for lines in 3-space" in the Proceedings of the 8th ACM Symposium on Computational Geometry, 1992. This research was initiated while visiting the International Computer Science Institute in Berkeley.

three dimensional counterpart fewer results are known. Intersection of *convex polyhedra* in three-space can be detected in time logarithmic in the total number n of vertices, faces and edges, using data structures of linear size [CD80, DK83, DK90]. Non-convex polygonal objects in 3-space are much harder from a computational point of view. If we have two *terrains* with the same vertical direction, Chazelle et al. [CEGS89a] solve the intersection-detection problem in time $O(n^{4/3+\epsilon})$. In [MS85] the case of one convex and one non-convex polyhedra is considered. If we have two non-convex polyhedra or any set of non-convex polyhedra we can solve the intersection-detection problem in time $O(n^{8/5+\epsilon})$ [Pel93].

In this paper we are interested in detecting the intersection of a query simplex among a set of (possibly intersecting) non-convex polyhedral obstacles. We allow preprocessing of the obstacles and we answer on-line intersection queries in sublinear time. For this problem, a simple-minded extension of the approach of [CD80, DK90] requires testing each obstacle with the query simplex independently. In the worst case we would have a linear number of simplex-obstacle pairs to consider. The intersection-detection results in [CEGS89a, Pel93] are inherently off-line and do not immediately imply a solution for our on-line query problem.

The solution of this intersection-detection problem allows us to determine efficiently whether a placement of a polyhedral object of constant complexity within polyhedral obstacles is collision-free. Finding collision-free placements for objects is a basic problem for applications in robotics. There are many results for several variations of the free-placement problem (e.g. finding a collision-free path, finding the placement of the largest copy, etc.) for a polygonal object amidst polygonal obstacles *in the plane* (see [LS87b, LS87a, GSS88] [KS88, HO89] [CK89] [FHS89, Tol91], see [SS89] for a survey).

In [Hal91] some motion planning problems for systems with three degrees of freedom are considered. In this case the robot is mapped as a point in a suitable 3-dimensional parametric space and the obstacles are mapped as surfaces within this space. Also, recent work of Aronov and Sharir [AS90, AS92] on the complexity of a cell in an arrangement of triangles in 3-space has applications to this kind of problem.

The main result of the paper is the following (Theorem 7): given a set of polyhedral obstacles with n vertices, edges and faces, we can build a data structure using m units of storage, with $n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$, such that for any query simplex we can determine in time $O(n^{1+\epsilon}/m^{1/4})$ whether the simplex is collision-free[2].

To our knowledge no previous sublinear algorithm for collision-free placement queries among several non convex polyhedra has been known.

In 3-dimensional space, free-placement problems have a high intrinsic combinatorial complexity, therefore we tackle the simple problem of building a data structure for free-placement queries. On the other hand, our data structure is powerful because *both* the placement (six degrees of freedom) *and* the shape (as long as it is of constant complexity) of the robot are part of the query.

We consider first a solution which, at the expense of large storage, answers the queries

[2]All the bounds presented hold for every $\epsilon > 0$ and the multiplicative constants depend on ϵ.

in logarithmic time. Applying the methods in [CSW90, Mat91, AM92a] (see also [AS91a]) we obtain a space/query-time trade-off.

In [Pel93, AS91a] algorithms for the following problem are presented: given a set of triangles, count (or report) the triangles intersected by any query ray or segment. One of the main building blocks for our collision-free data structure is a solution for the *dual problem*: given a set of segments in 3-space count (or report) the segments intersected by any query triangle. Dobkin and Edelsbrunner in [DE84] solve the problem of counting the number of segments intersected by a query *plane*; but their approach does not seem to extend to query *half-planes*. Our approach is based on reducing the problem to *halfspace range searching* and it attains the goal of a data structure for half-plane queries. Once we have the solution for half-planes it easily extended to triangle queries.

1.2 The closest pair of lines in 3-space

Arrangements of lines in 3-space have been studied recently in computational geometry (e.g. [CEGS89a] [Pel90] [Pel93] [PS92] [AS91a] [dBHO$^+$91] [CEG$^+$90]). Some of the problems concerning lines can be classified as "incidence" or "order" problems. For example, the *relative orientation* of lines in space [CEGS89a] is exploited in [CEGS89a, Pel93, dBHO$^+$91] to solve ray-shooting problems, and in this paper to answer collision-free queries. Problems that involve a metric on the set of lines in R^3 can be classified as "neighbor" problems

If a robot does not intersect any obstacle, the next natural question is to find the obstacle *closest to the robot*. Thus, we are led to consider "neighbor" problems in 3-space. Neighbor problems for sets of points have been extensively studied in computational geometry [Vai89, AESW91, PS85] also in connection with the well-studied *Voronoi Diagrams* (see [Aur91] for a survey). Much less is known about neighbor problems in higher dimensional spaces *for objects which are not points.*

Neighbor problems for polyhedral objects in 3-space such as lines, segments and polyhedra are important for applications not only in robotics, but also for 3D-VLSI and CAD. In these two areas we often need to enforce a minimum separation distance among 3-dimensional objects.

In Sections 4.2 and 4.3 we describe algorithms for solving a few neighbor problems on lines in 3-space. We are interested in finding the shortest segment connecting two lines in a set of lines, under a variety of constraints. We give algorithms to find the *shortest vertical segment* connecting two elements in a set of lines or segments which runs in $O(n^{8/5+\epsilon})$ expected time. Surprisingly, we prove an expected time bound for finding the *longest vertical segment* that is only $O(n^{4/3+\epsilon})$. We find the shortest segment connecting two lines in a set of n lines in $O(n^{5/3+\epsilon})$ expected time.

All the neighbor problems considered in this paper can be easily solved in $O(n^2)$ time. For some of the problems slightly sub-quadratic algorithms have been known. For the special case of the shortest vertical distance between two sets of edges of non-intersecting polyhedral terrains Chazelle et al. [CEGS89a] give an $O(n^{4/3+\epsilon})$ randomized expected time algorithm.

The problem of the minimum vertical separation for segments was solved in [CS90] in time $O(n^{1.99987})$. The method in [CS90] maps a segment into a point in 6-dimensional space. Using the representation in [CS90] and the more recent decomposition technique of [CEGS89b] it is possible to obtain a time bound of roughly $O(n^{2-1/9}) = O(n^{1.8889})$.

Independently, Guibas, Sharir and others [Gui91] have considered the problem of finding the closest pair of lines, obtaining an $O(n^{5/3+\epsilon})$ algorithm. More recently, Chazelle et al. [CEGS92] have improved the running time to $O(n^{8/5+\epsilon})$.

The main strategy used in this paper to solve closest pair problems is to build first a data structure to answer closest line queries. In a second stage we transform the method for solving on-line queries into an off-line method by batching the queries. Since the on-line data structures can have a *multilayer structure* [Meh84] we develop in Section 4.1 a general method called *nested batching technique* which is of independent interest (for more applications see [Pel92]).

The paper is organized as follows. In Section 2 we give an overview of the geometric techniques used to obtain the results. In Section 3 we discuss the problem of detecting efficiently whether a placement of a simplex is collision-free. Section 4 deals with several neighbor problems: finding the shortest and longest vertical segments, and finding the shortest segment in a set of lines.

2 Geometric and algorithmic preliminaries

In this section we survey some geometric and algorithmic results which lay the groundwork for the main results of the subsequent sections.

2.1 Arrangements

A finite set H of hyperplanes in R^d defines a decomposition of R^d into cells of various dimensions, which is called the *arrangement* $\mathcal{A}(H)$ of H [Ede87]. If $|H| = n$ the maximum number of cells in $\mathcal{A}(H)$ is $O(n^d)$ and the arrangement $\mathcal{A}(H)$ can be computed in optimal $O(n^d)$ time [EOS86]. One d-dimensional cell of $\mathcal{A}(H)$ is bounded by $O(n^{\lfloor d/2 \rfloor})$ cells of any dimension [Ede87].

Similarly, a set V of real semi-algebraic varieties in R^d decomposes R^d into cells of various dimensions, which is called the *arrangement* $\mathcal{A}(V)$ of V [CEGS89b]. In [CEGS89b], Chazelle at al. describe a method for building a point-location data structure for the arrangement of n algebraic varieties of fixed maximum degree in R^d. For $d \geq 3$, the data structure has size $O(n^{2d-3+\epsilon})$ and it can be build in $O(n^{2d-3+\epsilon})$ randomized expected time or $O(n^{2d+1})$ deterministic time. The query time is $O(\log n)$.

2.2 Random sampling and cuttings

Given a random sample R of a set of hyperlanes H, with $|R| = r \leq n$, let us consider the arrangement $\mathcal{A}(R)$. A triangulation $\triangle\mathcal{A}(R)$ is a subdivision of each cell of $\mathcal{A}(R)$ into simplices such that the vertices of each simplex are vertices of $\mathcal{A}(R)$. The number of simplices in $\triangle\mathcal{A}(R)$ is $O(r^d)$. The random sampling theory of Clarkson [Cla87] states that with probability at least 1/2 the interior of each simplex $s \in \triangle\mathcal{A}(R)$ does not meet more than $O(n/r \log r)$ hyperplanes of H. A set as $\triangle\mathcal{A}(R)$ is called a *cutting* for H.

Given H we can build a data structure which uses $Cn^{d+\epsilon}$ storage, for each $\epsilon > 0$, where the constant C depends on ϵ, such that a query point is located in $\mathcal{A}(H)$ in $O(\log n)$ time [Cla87]. This data structure is built in expected time $O(n^{d+\epsilon})$ [Cla87].

For $d = 2$ Matoušek [Mat90] gives a deterministic method that, for a parameter $r < n$ subdivides the plane into $O(r^2)$ triangles in time $O(nr)$ such that the interior of each triangle meets only n/r lines in H.

For a set V of n algebraic varieties of fixed degree in R^d, $d > 2$, Chazelle at al. [CEGS89b] prove that a R^d can be decomposed into $O(r^{2d-3}\beta(r))$ cells of constant descriptive complexity, where $\beta(.)$ is a sub-logarithmic function and each cell intersects no more than $O(n/r \log r)$ surfaces in V.

These results on cuttings are the basis of many divide-and-conquer algorithms in computational geometry (see [Aga91] for a survey) .

2.3 On-line and off-line point location

The algorithms of [Cla87] and [CEGS89b] solve, in logarithmic time, on-line point location problems in arrangements of hyperplanes or algebraic varieties of fixed degree. In several applications the queries are given off-line and we can batch them in order to speed up the overall computation.

In [EGS90], Edelsbrunner et al. give a method for batched point-location which is generalized is Section 4.1 to deal with a vast class of off-line pont location problems. Given n surfaces and m points, we compute properties of the points which depend only on the locations of these points with respect to the arrangement of the given surfaces in expected time $O(m^{a/(a+1)}n^{a/(a+1)} + m^{1+\epsilon} + n^{1+\epsilon})$, where a is a characteristic parameter of the problem.

To apply this technique an important requirement is the ability to map data and queries as point or surfaces, via dual transformations.

Geometric dual transformations [Ede87, EMP$^+$82] are pairs of functions, one function maps points to surfaces (hyperplanes) and the second function maps surfaces (hyperplanes) to points in R^d. Duality mappings preserve incidence and order relations.

2.4 Halfspace range searching

A problem intimately connected to the point-location problem is the half-space range searching problem. Given a set S of n points in R^d, build a data structure such that, for every query half-space h^+, the number of points in $S \cap h^+$ is computed efficiently. This problem is solved in [CSW90, Mat91] using *partition trees*. In a partition tree, each node is associated with a region in R^d such that only a fraction of the children intersect the hyperplane h supporting the query half-space. During the query we retrieve the number of points of S within the regions associated with children not intersected by h and we recurse the query on the children intersected by h. Partition trees are quite versatile and they can be used to set up multi-level data structures.

2.5 Multi-level data structures

Multilevel data structures are a basic paradigm in computational geometry [Meh84]. They are used to search for elements satisfying a complex property. Usually the complex property is split into elementary properties and each elementary property is tested at a specific level of the data structure. For example, in [CSW90, Mat91] sets of points are organized in multilevel partition trees to answer simplex range queries, where each level of the data structure tests the position of the data points with respect to the hyperplane spanning a facet of the simplex. We have this fundamental theorem in [CSW90]:

Theorem 1 ([CSW90]) *Simplex range searching in n points in R^d can be performed in $O(n^{1+\epsilon}/m^{1/d})$ query time, for every $\epsilon > 0$, using a data structure of size m (for any m between n and n^d) which can be computed in $O(m^{1+\epsilon})$ randomized expected time.*

Matoušek in [Mat91], using a different partition scheme, is able to make the preprocessing deterministic and to reduce the query time to $O(n \log^{O(1)} n/m^{1/d})$.

2.6 Parametric search

In some of the algorithms for nearest neighbor problems on lines we use Megiddo's parametric search technique [Meg83, Col87]. Roughly speaking, this technique transforms an algorithm to test a property into an algorithm to find the minimum value of a parameter for which the property is true, under quite general conditions.

More formally, suppose we have an algorithm P' to compute a predicate $P(i,t)$, which depends on an input i and a real parameter t. The algorithm P' only uses the parameter t to perform branching tests based on the evaluation of fixed degree polynomials which depend on i and t. Moreover, suppose that the predicate $P(i,t)$ is monotone in t, meaning that for increasing values of t it never switches from false to true.

Megiddo's parametric search technique is a method that transforms P' into an algorithm P'' for finding the minimum value of t for which the predicate $P(i,t)$ is true.

As an intermediate step of the transformation we need a parallel version of the program P', which we call $\mathcal{P}$. The parallel time is $T_{\mathcal{P}}$ and the number of processors is p. The new algorithm P'' simulates sequentially $\mathcal{P}$ without specifying the value of t. When each of the p processors is stopped at a branching step which requires the evaluation of a polynomial in t we compute all the roots of the polynomials at the branching steps and we sort them. Then we use the sequential algorithm P' to perform a binary search on the sequence of roots in order to find the interval where the minimum value t^* lies. Once we have this interval we can compute the sign of the branching polynomials at t^*. The algorithm branches accordingly to these values. The algorithm completes the simulation and determines a final interval whose left endpoint is exactly t^*. The total time of the simulation is given by $O(pT_{\mathcal{P}} + T_{P'}T_{\mathcal{P}} \log p)$.

Applications of Megiddo's parametric search to geometric problems are in [AASS90] [AM92b] [CEGS92].

2.7 Plücker Coordinates of lines

To solve problems on lines and polyhedra we use the *Plücker coordinates* of lines. Algorithmic uses of Plücker coordinates can be found in [CEGS89a] and in [Pel91, PS92]; a classical treatment of Plücker coordinates can be found in [Som51].

A point in real 3-dimensional space has Cartesian coordinates (x, y, z) and homogeneous coordinates (x_0, x_1, x_2, x_3). The relations between the two systems of coordinates are given by the following equations: $x = x_1/x_0$, $y = x_2/x_0$ and $z = x_3/x_0$. Two points $x = (x_0, x_1, x_2, x_3)$ and $y = (y_0, y_1, y_2, y_3)$ in 3-dimensional homogeneous coordinates define a line l in 3-space. The six quantities

$$\xi_{ij} = x_i y_j - x_j y_i \quad for\ ij = 01, 02, 03, 12, 23, 31$$

are called *Plücker coordinates* of the line l (oriented from x to y). They correspond to the two-by-two minors of the two-by-four matrix formed by the coordinates of the point x (on the first row) and y (on the second row).

The six parameters are not independent; they must satisfy the following equation (whose solution set constitutes the Plücker hypersurface or Klein quadric or Grassman manifold $\mathcal{F}_4^2$ [Sto89, Som51]):

$$\Pi \ : \ \xi_{01}\xi_{23} + \xi_{02}\xi_{31} + \xi_{03}\xi_{12} = 0 \tag{1}$$

The incidence relation between two lines l and l' can be expressed using the Plücker coordinates of l and l'. Let a_1, b_1 (resp. a_2, b_2) be two points on l (resp. l') oriented as l (resp. l'). The incidence between l and l' is expressed as the vanishing of the determinant of a four-by-four matrix whose rows are the coordinates of a_1, b_1, a_2, b_2 in this order from top to bottom.

$$\begin{vmatrix} a_{10} & a_{11} & a_{12} & a_{13} \\ b_{10} & b_{11} & b_{12} & b_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ b_{20} & b_{21} & b_{22} & b_{23} \end{vmatrix} = 0 \tag{2}$$

If we expand the determinant according to the two-by-two minors of the sub-matrix formed by the coordinates of the points a_1, b_1 and the minors of the sub-matrix formed by the points a_2, b_2, we obtain the following equation in which only Plücker coordinates are involved:

$$\xi_{01}\xi'_{23} + \xi_{02}\xi'_{31} + \xi_{03}\xi'_{12} + \xi'_{01}\xi_{23} + \xi'_{02}\xi_{31} + \xi'_{03}\xi_{12} = 0 \tag{3}$$

Let us introduce two mappings: $\pi : l \to \pi(l)$ maps a line in R^3 to an hyperplane in $\mathcal{P}^5$ (5-dimensional oriented projective space) whose plane coordinates are the Plücker coordinates of l appropriately reordered. $p : l \to p(l)$ maps a line in R^3 to a point in $\mathcal{P}^5$ whose coordinates are the Plücker coordinates of the line. The incidence relation between the two lines l, l' (expressed by Equation 3) can be reformulated as an incidence relation between points and hyperplanes in $\mathcal{P}^5$. Equation 3 can be rewritten in the form $\pi_l(p_{l'}) = 0$, which is equivalent to requiring point $p(l')$ to belong to hyperplane $\pi(l)$. Computations that are standard in real spaces can be done in oriented projective spaces using a method in [Sto89].

For any given pair of lines l and l' the sign of the quantity $\pi_l(p_{l'})$ is called the *relative orientation* of l and l', denoted by $l \diamond l'$.

2.8 Lines in Space

In a seminal paper [CEGS89a] Chazelle, Edelsbrunner, Guibas and Sharir use Plücker coordinates to obtain several results on problems involving polyhedral terrains in 3-space.

One of the problems discussed in [CEGS89a] is the following: given a set $\mathcal{L}$ of n lines in R^3, build a data structure so to answer efficiently whether a query line l is above every line in $\mathcal{L}$. They give a data structure of size $O(n^{2+\epsilon})$ to answer such queries in time $O(\log n)$. Since this data structure is the point of departure of the present article from previous work, we discuss it in full detail.

Given two non vertical lines l and l', l is above l' if there exists a vertical line meeting both lines, and the intersection with l is above the intersection with l'. The objective is to express the *above* relation in terms of relative orientation. Let us consider the line λ parallel to l and passing through the point $z_\infty = (0, 0, 1, 0)$. The line l' is *above* l if it is "between" l and λ, therefore we require that:

$$l' \diamond l = -l' \diamond \lambda \tag{4}$$

Thus, we need to check the consistency of two linear inequalities. Let $\mathcal{L} = \{l_1, .., l_n\}$ be a set of n *oriented lines*. And let λ_i be the line parallel to l_i and through the point z_∞. Given query line l, we first test if it is *consistently oriented* with respect to $\mathcal{L}$, that is if $l \diamond \lambda_i = -1$ for all i. For a line l which is consistently oriented the aboveness test reduces to test membership of its corresponding Plücker point in the intersection of n Plücker halfspaces.

Chazelle et al. also observe that to decide if a line l is consistently oriented it is sufficient to project l and $\mathcal{L}$ on the xy-plane and to test if the projection of l is clockwise with respect to the projections of all lines in $\mathcal{L}$. Thus, there are $2n$ ways to orient the lines in $\mathcal{L}$ consistently such that any l is consistently oriented exactly for one orientation of $\mathcal{L}$.

3 An incidence problem in 3-space

3.1 Redefining the *above* relation

Our first objective is to redefine the *above* relation between lines in a form more suitable for algorithmic applications. We do so by generalizing in a more algebraic context the steps of the algorithm in Section 2.8.

Let us fix once and for all an orthogonal reference frame in 3-space with unit vectors $(\vec{i}, \vec{j}, \vec{k})$ forming a positively oriented triple according to the skew rule. And let (x, y, z) be the coordinates of a generic point in this reference frame.

Given a line l, a vector $\vec{v}$, and a line l' in R^3 we say that l is *above* l' in direction $\vec{v}$ (namely $above(l, l', \vec{v})$) if moving l in direction $\vec{v}$ we eventually intersect l'.

When $\vec{v}$ is a vector parallel to the $-\vec{k}$-axis, we are in the situation described in [CEGS89a]. In [CEGS89a] the condition for a line to be above another line was given by the relative orientation of the two lines in Plücker space and by the relative orientations of the projections of the two lines on the xy-plane. Let us define oriented lines by ordered pairs of points in 3-space: $l = (a, b)$ and $l' = (a', b')$, then the relative orientation of the projections of l and l' on the xy-plane is equivalent to testing if the triple $(a - b, a' - b', \vec{v})$ is oriented as $(\vec{i}, \vec{j}, \vec{k})$. This, in turns, is equivalent to computing the sign of the following determinant:

$$tsp(l, l', \vec{v}) = sign \begin{vmatrix} a_x - b_x & a_y - b_y & a_z - b_z \\ a'_x - b'_x & a'_y - b'_y & a'_z - b'_z \\ 0 & 0 & 1 \end{vmatrix}$$

which is the triple scalar product of the vectors $a - b$, $a' - b'$ and $(0, 0, 1)$. We can generalize the notion of relative orientation by making the "vertical direction" part of the query. If the vertical direction is specified by a variable vector $\vec{v}$, the relative orientation is still the sign of the triple scalar product of $(a - b, a' - b', \vec{v})$:

$$tsp(l, l', \vec{v}) = sign \begin{vmatrix} a_x - b_x & a_y - b_y & a_z - b_z \\ a'_x - b'_x & a'_y - b'_y & a'_z - b'_z \\ v_x & v_y & v_z \end{vmatrix} \qquad (5)$$

Considering the signs $(+1, -1)$ as boolean values, we prove the following technical lemma:

Lemma 1 *$above(l, l', \vec{v}) = tsp(l, l', \vec{v})$ xor $(l \diamond l')$.*

Proof. i) If $\vec{v} = -\vec{k}$, then clearly $tsp(l, l', \vec{v}) = l' \diamond \lambda$, by the discussion in [CEGS89a]. Since l is above l' if and only if the two sides of equation 4 have different values the lemma is proved.

ii) If $\vec{v} \neq -\vec{k}$, then we can apply a rigid transformation to our reference frame which takes $-\vec{k}$ into $\vec{v}$. It is easy to check that such transformation does not change the value of $l \diamond l'$ and $tsp(l, l', \vec{v})$, therefore we are back to case i) and the lemma is proved. ∎

We say that a line l is above a set $\mathcal{L}$ in direction $\vec{v}$ if it is above every line in $\mathcal{L}$, namely $above(l, \mathcal{L}, \vec{v})$. From Lemma 1 we have:

$$above(l, \mathcal{L}, \vec{v}) = \bigwedge_{i=1,n} tsp(l, l_i, \vec{v}) \ xor \ (l \diamond l_i). \tag{6}$$

3.2 Halfplane intersection queries on lines

The determinant in (5) can be expanded in a bilinear form in terms of the two-by-two minors of the last two rows and the one-by-one minors of the first row. Let us define $\pi'(l) = (a_x - b_x, a_y - b_y, a_z - b_z)$ and $p'(l', \vec{v}) = (u_{yz}, u_{zx}, u_{xy})$, where $u_{ij} = (a'_i - b'_i)v_j - (a'_j - b'_j)v_i$.

We can think of $p'(l', \vec{v})$ as a point in oriented projective 2-space $\mathcal{P}^2$ and of $\pi'(l)$ as a hyperplane in the same space. The value of $tsp(l, l', \vec{v})$ is just the sign of the inner product of $\pi'(l)$ and $p'(l', \vec{v})$. The value of $tsp(l, l', \vec{v})$ is also the sign of the polynomial defining $\pi'(l)$ evaluated at $p'(l', \vec{v})$.

Theorem 2 *Given a set $\mathcal{L}$ of n lines, there is a data structure $D(\mathcal{L})$ of size $O(n^{2+\epsilon})$ that in time $O(\log n)$, for a query pair $(l', \vec{v})$, determines if l' is above $\mathcal{L}$ in direction $\vec{v}$.*

Proof. We number the lines in $\mathcal{L}$ from 1 to n obtaining a sequence of lines $L = [l_i]$. We construct the arrangement $\mathcal{A}'(\mathcal{L})$ of the hyperplanes in $\{\pi'(l) \mid l \in \mathcal{L}\}$. Consider a cell c of $\mathcal{A}'(\mathcal{L})$, and a point $q \in c$. We define as the *sign sequence* of the cell c the sequence $[sign(\pi'(l) \cdot q) | l \in L]$. We build for each region c a Plücker polytope using the negated sign sequence c as a guide for choosing the correct side of the Plücker hyperplanes.

Let $(l', \vec{v})$ be a query pair. First we locate the cell c in the arrangement $\mathcal{A}'(\mathcal{L})$ containing $p'(l', \vec{v})$. Then, we test for inclusion of the Plücker point $p(l')$ in the Plücker polyhedron associated with c.

In order to obtain a space-efficient data structure we use a a plane partition approach to construct the arrangement in $\mathcal{P}^2$. Using the method in [Mat91] we partition the plane into $O(r^2)$ regions such that each region is cut by no more than n/r hyperplanes $\pi'(l)$. For at most $O(n)$ hyperplanes π' not cutting through a region c we build a corresponding point-in-polytope query data structure in Plücker space at the expense of at most $O(n^{2+\epsilon})$ storage (see [CEGS89a, Pel90]). Then we recurse the construction in each region on the hyperplanes cutting through. The storage $S(n)$ needed to construct the data structure satisfies the following inequality:

$$S(n) \leq cr^2 S(n/r) + cr^2 O(n^{2+\epsilon}). \tag{7}$$

The solution is $S(n) = O(n^{2+\epsilon})$. By choosing $r = n^\nu$ for an appropriate value of ν, and by adding planar point location data structures, we can achieve $O(\log n)$ query time. Note that the data structure itself does not depend on any specific direction $\vec{v}$. ∎

Corollary 1 *Given a set $\mathcal{L}$ of n lines, there is a data structure $D(\mathcal{L})$ of size $O(n^{2+\epsilon})$ that in time $O(\log n)$, for a query halfplane h, determines if h intersects* 0 *(or all) lines in $\mathcal{L}$.*

Proof. If a line l is above all lines in $\mathcal{L}$ along a direction $\vec{v}$, then the half-plane defined by l and $-\vec{v}$ does not intersect any line in $\mathcal{L}$. Conversely, the halfplane defined by l and $\vec{v}$ intersects all lines in $\mathcal{L}$. Here we consider lines parallel to the halfplane h as intersecting h (at infinity). ∎

Corollary 2 *Given a set H of n halfplanes, there is a data structure $D(H)$ of size $O(n^{2+\epsilon})$ that in time $O(\log n)$, for a query line l, determines if l intersects 0 (or all) halfplanes in H.*

Proof. A pair $(l', \vec{v})$ determines a halfplane h in 3-space. We can think of $p'(l', \vec{v})$ as an hyperplane in $\mathcal{P}^2$ and $\pi'(l)$ as a point in $\mathcal{P}^2$. We build the data structure of Theorem 2 reversing the roles of hyperplanes and points in $\mathcal{P}^2$. For a query line l we can detect efficiently if any line l_i stored in the data structure hits l when moved in the associated direction $\vec{v}_i$. This is equivalent to detecting an intersection of l with all the half-planes stored in the data-structure. If a line l hits all the opposite halfplanes defined by l_i and $-\vec{v}_i$, then it misses all the halfplanes defined by l_i and $\vec{v}_i$. ∎

The data structure of Corollary 2 is a generalization of the data structure in [CEGS89a] which can detect intersections of lines and *vertical* halfplanes only. A data structure solving the same problem of Corollary 2 is in [AM92b].

3.3 Triangle intersection queries on segments

Let S be a set of segments in 3-space. We build a multilevel data structure to solve the following problem: count the number of segments in S intersected by a query triangle. The general strategy is the following. Let t be a query triangle and let $\text{aff}(t)$ be the plane spanning t. The first level of the data structure detects the segments of S which are intersected by $\text{aff}(t)$. Those segments meeting $\text{aff}(t)$ can be extended into full lines without introducing new intersections. The next three levels of the data structure are associated with the three edges of t. For each edge of t we consider the line spanning it and the half-plane of $\text{aff}(t)$ based on that line and containing t. We select the lines that meet this query halfplane. At the bottom of the multilevel data structures we count exactly the number of segments meeting t. Now we formalize the above strategy and we give bounds on the storage and query time.

Theorem 3 *Given a set $\mathcal{L}$ of n lines, there is a data structure $D(\mathcal{L})$ of size $O(n^{4+\epsilon})$ that in time $O(\log n)$, for a query halfplane h, counts the number of lines in $\mathcal{L}$ intersected by h.*

Proof. We follow the same general approach as in the data structure of Theorem 2. The main difference is that for each planar region we associate a point-location data structure for the whole zone of the Plücker surface in the arrangement of Plücker hyperplanes [Pel93]. This data structure requires $O(n^{4+\epsilon})$ storage and we can locate a point within the Plücker arrangement in time $O(\log n)$. We prestore in each cell the number of data lines that satisfy the *above* condition with respect to the line supporting the query half-plane. We solve the same problem recursively on each planar region relatively to the lines such that $\pi'(l)$ crosses the planar region. The total storage is $O(n^{4+\epsilon})$ and the total query time is $O(\log n)$. ∎

Theorem 4 *Given n segments there is a data structure of size $O(n^{4+\epsilon})$ that counts segments intersected by a query triangle in time $O(\log n)$.*

Proof. We dualize every segment in S into a double spatial wedge using a standard point-plane duality $\mathcal{D}$ [Ede87]. Detecting the segments intersected by the plane $\mathrm{aff}(t)$ is equivalent to locate the dual point $\mathcal{D}(\mathrm{aff}(t))$ in the arrangement $\mathcal{A}(\mathcal{D}(S))$ of the wedges.

When we have obtained the set of segments intersected by $\mathrm{aff}(t)$ as the disjoint union of sets stored in the first level of the data structure we can extend the segments into lines without introducing any new intersection.

We consider now t as the intersections of three half-planes in 3-space. In the next three levels of the data structure we store the lines supporting the segments and we build halfplane-line intersection data structures of Theorem 3. At the last level, we can count the number of segments intersecting t. The total storage is $O(n^{4+\epsilon})$ as follows from solving a sequence of inequalities similar to (7) and the query time is $O(\log n)$. ∎

We can also report the intersections in time $O(\log n + k)$, using $O(n^{4+\epsilon})$ storage, where k is the output size.

3.4 Linear storage and sublinear query time

In the previous section we have reduced the problem of counting the number of segments of S intersected by a query triangle to the problem of locating several query points within several arrangements of hyperplanes (Plücker hyperplanes in Plücker space or lines on a parametric plane). By doing so we compute efficiently, during the query, the relative position of the points with respect to the hyperplanes stored in the data structure.

As a matter of fact we could have interpreted the data structure as storing points and the query as halfspace range queries. The final result is the same (i.e. checking the sign of polynomials) but the preprocessing is quite different because the data structure can now be constructed using less storage at the expenses of an increased query time.

We can rebuild the whole multilevel data structure of section 3.3 using an *half-space range searching approach* [CSW90, AS91a, Mat91, AM92a] rather than the point-location approach. The half-space range query approaches in [CSW90, Mat91] use a point location data structure as a subroutine, and in section 3.3 we have developed such subroutine for the triangle query problem.

When solving the half-space range problem in Plücker space we have to take into account the presence of the Plücker hypersurface. We have the property that all Plücker points are on the Plücker hypersurface and all Plücker hyperplanes are tangent to the Plücker surface. Adapting the analysis of the data structure in [Mat91] to this special case and using a result in [APS93], Agarwal and Matoušek prove the following lemma:

Lemma 2 ([AM92a]) *Given n Plücker points it is possible to solve half-space range searching problems with Plücker hyperplanes using m storage, with $n^{1+\epsilon} < m < n^{4+\epsilon}$, $O(m)$ expected preprocessing time and $(n^{1+\epsilon}/m^{1/4})$ query time.*

As in [CSW90, Mat91] it is possible to trade storage and query time. Using Lemma 2 and the analysis in [AS91a], Agarwal and Matoušek obtain the following result for ray-shooting on a set of triangles in 3-space:

Theorem 5 ([AM92a]) *Given n triangles in 3-dimensional space and any $\epsilon > 0$, one can preprocess them in randomized expected time $O(m)$ into a data structure of size $O(m)$, for $n^{1+\epsilon} < m < n^{4+\epsilon}$, so that for a query ray ρ, the first triangle met by ρ can be computed in time $O(n^{1+\epsilon}/m^{1/4})$.*

Lemma 2 applied to the analysis of section 3.3 gives the following theorem:

Theorem 6 *Given a set of n segments we can build a data structure using m units of storage, with $n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$, such that for any query triangle t we can count in time $O(n^{1+\epsilon}/m^{1/4})$ the number of segments intersected by t.*

Theorem 6 is an essential ingredient in the proof of the main theorem of the paper:

Theorem 7 *Given a set of polyhedral obstacles with n edges, faces and vertices, we can build a data structure using m units of storage with $n^{1+\epsilon} \leq m \leq n^{4+\epsilon}$ such that for any query simplex s we can determine in time $O(n^{1+\epsilon}/m^{1/4})$ whether s is collision-free.*

Proof. Assume that we are given a set of polyhedral objects with a total of n edges and a simplex s in 3-space. We have to consider only four collision cases. For each collision case we describe a data structure that answers whether the query simplex s generates a collision of that type.

(i) *An object is completely contained in s.* We select one arbitrary point of each obstacle thus obtaining a set P of at most n points in R^3. We build a data structure D_1 for simplex range searching on P using the methods in [CSW90, Mat91], which satisfies the the stated bound. We use the simplex s to query D_1. If any point of P is in s then we have a clash of type (i).

(ii) *An edge of s meets a face of some object.* We take the facets of the obstacles and we triangulate them, obtaining a set T of $O(n)$ triangles in 3-space. We build a data structure D_2 for ray-shooting on T using the method in [AS91a] and its improvement in [AM92a]. We take for each vertex v of s the rays from v containing the edges of s incident to v. Using the data structure D_2 we determine the first triangle intersected by each ray. If for some ray the intersection point is closer to v than the second end-point of the segment spanned by this ray, then we have a collision of type (ii). We have a constant number of ray-shooting queries for each simplex s. The preprocessing and the query time are those stated Theorem 5.

(iii) *A face of s meets an edge of some object.* We take the edges of the obstacles obtaining a set S of $O(n)$ segments in 3-space. We build a data structure D_3 for answering triangle-intersection queries, using the method of Theorem 6. In turns we take the facets of s and we count the number of segments intersected by each facet. If we detect some intersection we have a collision of type (iii).

(iv) *The query simplex s is completely contained within a polyhedral obstacle.* This case is checked by using the data structure D_2 for ray-shooting queries. We take a ray ρ from an arbitrary point of s in an arbitrary direction. If no obstacle is hit the simplex is not contained in any obstacle. Otherwise, we obtain the first facet t hit by ρ. If locally the ray ρ hits t from the interior of the object containing t, then we have a collision of type (iv).

Collision cases (i), (ii), (iii) and (iv) cover all possible collisions between a simplex and a set of polyhedral obstacles. ∎

4 Neighbor problems for lines in space

To solve neighbor problems on a set of lines in 3-space we use the following general strategy. Given a set of lines $\mathcal{L}$ we build a data structure $D(\mathcal{L})$ that efficiently answers nearest-line queries for any query line l. The second step is to balance the preprocessing time and the query time by batching the queries, which we know in advance, over the data structure $D(\mathcal{L})$.

On an abstract level we can think that a query is a pair $(l, \mathcal{L})$, where l is any line and $\mathcal{L}$ a set of lines. Solving the query is equivalent to computing the truth value of a conjunction of several predicates which depend on l and on the elements of $\mathcal{L}$. The data structure $D(\mathcal{L})$ allows us to find the truth value of many predicates efficiently in an implicit way and therefore allows us to solve the query problem. Once we have the data structure $D(\mathcal{L})$, we can produce its batched version in a quite general way which we call *nested batching technique*. This technique is a generalization of an approach to batching computations in [EGS88]. This technique has been used in several recent papers (e.g. [Pel90, AS91b]) where the underlying predicate has only one conjunct. In [Pel93] the technique has been used for a formula with two conjuncts. Here we generalize it in an abstract setting for any constant number of conjuncts.

4.1 The nested batching technique

Let $\mathcal{A}$ and $\mathcal{B}$ be two classes of geometric objects and $A \subset \mathcal{A}$ and $B \subset \mathcal{B}$ two finite subsets. Our goal is to compute efficiently $|\mathcal{C}_k(A,B)| = |\{(a,b) \in A \times B | F(a,b)\}|$, where $F(a,b) = \wedge_{j=1,k} C_j$ is a conjunction of a constant number k of elementary conditions. Each elementary condition C_j is of the form $S_a^j(p_b^j) > 0$, where $S_a^j(.)$ is a polynomial whose structure depends on j and $\mathcal{A}$ and whose coefficients depend only on $a \in A$. Also, p_b^j is a tuple of real numbers depending only on $b \in B$. Moreover, we require that we can rewrite C_j in the dual form $S_b^j(p_a^j) > 0$, where $S_b^j(.)$ is a polynomial whose structure depends on j and $\mathcal{B}$, whose coefficients depend only on $b \in B$. Also, p_a^j is a tuple of real numbers depending only on $a \in A$. For technical reasons we assume that $C_0 = TRUE$. If $S_a^j(p_b^j) > 0$ we say that the *point* p_b^j is on the *positive side* of the *surface* $S_b^j(.) = 0$. From now on we will use the terms *point* and *surface* in this context. Note that, for a given j, an element $a \in A$ (resp. $b \in B$) is mapped to a point or to a surface. We call the point and the surface *dual* to one another.

The following conditions must be also satisfied:

(I) Given a family of surfaces $\{S_a^j(x) = 0 | a \in A\}$ there is a constant $d'(\mathcal{A},j)$ such that the space underlying the family of surfaces can be partitioned into $M'_j(r) = O(r^{d'(\mathcal{A},j)} \log^{O(1)} r)$ cells of constant descriptive complexity such that each cell meets at most $|A|/r \log r$ surfaces in the family. Moreover, we assume that this decomposition can be computed in expected time $O(|A| r^{O(1)})$.

(II) Given a family of surfaces $\{S_b^j(x) = 0 | b \in B\}$ there is a constant $d(\mathcal{B},j)$ such that the space underlying the family of surfaces can be partitioned into $M_j(r) = O(r^{d(\mathcal{B},j)} \log^{O(1)} r)$ cells of constant descriptive complexity such that each cell meets at most $|B|/r \log r$ surfaces in the family. Moreover, we assume that this decomposition can be computed in expected time $O(|B| r^{O(1)})$.

Conditions (I) and (II) can be rephrased requiring that there is a $1/r$-*cutting* of a certain size for the arrangement of surfaces (see [Aga91] for a survey on cuttings). To be more precise we require a cutting covering the portion of space where the query points are confined. If we do not have any condition on the points then the whole space must be covered. If the points are constrained to lie on a surface or within a convex polyhedron, only the surface or the polyhedron must be covered.

Let $d_j = \max\{d'(\mathcal{A},j), d(\mathcal{B},j)\}$ and let $d = \max_j d_j$. We will refer to d as the *characteristic dimension of the formula* F, while d_j is the *characteristic dimension of the elementary conjunct* C_j.

Let $T_j(m,n)$ be the time needed to determine the number of pairs $(a,b) \in A \times B$ satisfying the conditions associated with the elementary conjuncts form 1 to j, that is the time to compute $|\mathcal{C}_j(A,B)| = |\{(a,b) \in A \times B | \wedge_{i=1,j} C_i\}|$. We set $M(r) = \max_j\{M_j(r), M'_j(r)\}$. Just to give some examples,

for lines and points on the plane we obtain $M'_j(r) = M_j(r) = r^2$ simple regions.

For surfaces and points used in section 4.3 we obtain $M_j(r) = M'_j(r) = r^4\beta(r)$ simple regions, using the technique in [CEGS89b], where $\beta(r)$ is a sub-logarithmic function.

For Plücker points in an arrangement of Plücker hyperplanes we obtain $M_j(r) = M'_j(r) = r^4 \log r$, from a result in [APS93], because we need to cover only the Plücker hypersurface.

Lemma 3 *Given a set A and B as above, let $|A| = n$ and $|B| = m$. For every $0 \leq j \leq k$, we can build in time $O(n^{d+\epsilon})$ a data structure such that, for every $b \in B$, $|\mathcal{C}_j(\{b\}, A)|$ can be computed in time $O(\log^j n)$.*

Proof. Let $T_j(n)$ be the time needed to build the data structure up to level j. We prove the claim by induction on j. For $j = 0$, $T_0(n) = O(1)$ therefore the lemma is satisfied. Assume $j > 0$. Applying property (I) for a value of r constant from n and m, we compute a decomposition of the space into $M'_j(r)$ elementary cells each one intersected by no more than $n/r \log r$ surfaces in $\{S^j_a = 0 | a \in A\}$. For each elementary cell we select a point in its interior and we compute the set S^+_τ of surfaces not intersecting the region τ and in positive position with respect to τ. We associate the cardinality of S^+_τ with the region and we recurse this construction. Also, for each region τ and for all the elements $a \in A$ whose corresponding surface is in S^+_τ we build the data structure to compute $\mathcal{C}_{j-1}$, which can be built in time $T_{j-1}(n)$. The total time and space for this construction satisfies the recurrence:

$$\begin{aligned} &T_j(n) \leq M'_j(r)T_j(n/r \log r) + M'_j(r)T_{j-1}(n) + O(nr^{O(1)} + nM'_j(r)) \\ &T_j(O(1)) = O(1) \end{aligned}$$

Since $M'(r) \leq r^d \log^{O(1)} r$ and assuming by induction hypothesis that $T_{j-1} = O(n^{d+\epsilon})$, the solution to this recurrence is $T_j(n) = O(n^{d+\epsilon})$.

Given a query point p we locate by exhaustive search the region τ containing it. Than we recurse on the two data structures associated with τ. At the last level we collect the counters of the positive regions and we sum them up to form the final result. The query time $Q_j(n)$ satisfies this recurrence:

$$Q_j(n) \leq Q_j(n/r \log r) + Q_{j-1}(n) + O(M'(r))$$

Assuming by induction hypothesis that $Q_{j-1} = O(\log^{j-1} n)$, we obtain the solution is $Q_j(n) = O(\log^j(n))$. ∎

From Lemma 3 easily follows:

Lemma 4 *If $m \geq n^d$ then, for every $j \leq k$, $|\mathcal{C}_j(B, A)|$ can be computed in time $O(m^{1+\epsilon})$.*

Let us suppose now that $m \leq n^d$. We dualize points and surfaces at level j, and we compute the partition (II) for a constant value of r. For each cell τ of the partition we have a set of n_τ points contained it and a set of m_τ surfaces cutting through. By the property (II), $m_\tau \leq cm/r \log r$ for some constant c. For each cell τ the relative sign of the points in τ and the surfaces outside τ is similar for each point in τ. This implies that in time $O(m)$ we can determine for all points in τ and for most surfaces the truth value of the predicate associated with the j-th level. For the surfaces intersecting τ we proceed recursively. Let $T_j(m,n)$ be the time needed to compute $|\mathcal{C}_j(B,A)|$ From the above discussion $T_j(m,n)$ satisfies this recurrence:

$$T_j(m,n) \leq \sum_\tau T_j(m_\tau, n_\tau) + M(r)(m+n) + M(r)T_{j-1}(m, n_\tau) + O(mr^{O(1)}) \tag{8}$$

The following is the main theorem of this section:

Theorem 8 *For any ϵ and $\delta \leq \epsilon/d$:*

$$T_j(m,n) = D_j n^{d/(d+1)+\epsilon} m^{d/(d+1)} + B_j m^{1+\delta} + A_j n \log^{(1+j)} m$$

where D_j, B_j, A_j depend on ϵ and δ.

Proof. The proof is an induction on j and n inspired by similar argument is in [EGS88].

For $j = 0$, $C_0 = TRUE$ is satisfied by nm pairs. The product is computed in constant time. We consider $j > 0$ and we use the inductive hypothesis on T_{j-1}. Equation (8) becomes:

$$\begin{aligned} T_j(m,n) \leq\ & \sum_\tau T_j(m_\tau, n_\tau) + O(mr^{O(1)}) + \\ & M(r)(D_{j-1} n^{d/(d+1)+\epsilon} m^{d/(d+1)} + B_{j-1} m^{1+\delta} + A_{j-1} n \log^{1+(j-1)} m) + \\ & M(r)(m+n) \end{aligned} \tag{9}$$

For any fixed pair δ, ϵ, with $\delta \leq \epsilon/d$ we choose a value of r, constant from n and m, depending on ϵ and δ. The actual dependency of r on ϵ and δ will be defined along with the proof. If $m \geq n^d$ then, from Lemma 4, $T_j(m,n) \leq cm^{1+\delta}$ and the theorem is proved assuming $B_j \geq c$. Suppose $m \leq n^d$. In this case

$$m^{1+\delta} = m^{d/(d+1)} m^{1/(d+1)+\delta} \leq m^{d/(d+1)} n^{d/(d+1)+d\delta} \tag{10}$$

First notice that at each level of the recursion the term $M(r)n + M(r)A_{j-1} n \log^j m$ in (9) contributes $O(n \log^j m)$ on one level of the recursion; and there are at most $O(\log m)$ levels. This is a consequence of the fact that the objects in A (whose counter is n) are represented as points. The objects in B (which are counted in m) are represented as surfaces. The overall contribution of this term is $O(n \log^{j+1} m)$. It is sufficient to drop this term from the recursion (9) and prove that the modified inequality for a new unknown function $T'_j(m,n)$.

$$T'_j(m,n) \le \sum_\tau T'_j(m_\tau,n_\tau) + M(r)m + M(r)(D_{j-1}m^{d/(d+1)}n^{d/(d+1)+\epsilon} + B_{j-1}m^{1+\delta}) + O(mr^{O(1)}) \tag{11}$$

satisfies the bound

$$T'_j(m,n) = D_j m^{d/(d+1)} n^{d/(d+1)+\epsilon} + B_j m^{1+\delta}.$$

Choosing B_j and D_j large enough so that the bound holds for small values of m and n, we apply the inductive argument on $T'_j(m_\tau,n_\tau)$.

$$\begin{aligned} T'_j(m,n) \;\; &\le \;\; \Sigma_{i=1}^{M(r)}[D_j m_i^{d/(d+1)} n_i^{d/(d+1)+\epsilon} + B_j m_i^{1+\delta}] \\ &\quad +M(r)m + M(r)D_{j-1}m^{d/(d+1)}n^{d/(d+1)+\epsilon} + M(r)B_{j-1}m^{1+\delta} + O(mr^{O(1)}) \end{aligned}$$

But $\Sigma_{i=1,M(r)} m_i^{1+\delta} \le [M(r)c\log^{1+\delta} r/r^{1+\delta}]m^{1+\delta}$. We can sum up terms in m and $m^{1+\delta}$, set $E = B_j M(r)c\log^{1+\delta} r/r^{1+\delta} + M(r) + M(r)B_{j-1} + O(r^{O(1)})$, and use inequality 10. We obtain

$$T'_j(m,n) \le [D_j \Sigma_{i=1}^{M} m_i^{d/(d+1)} n_i^{d/(d+1)+\epsilon} + E m^{d/(d+1)} n^{d/(d+1)+d\delta}]$$

Now we eliminate the summation using first the bound on m_i and then the Hölder-Minkowsky inequality [Mit70].

$$\begin{aligned} &\Sigma_{i=1}^{M(r)} m_i^{d/(d+1)} n_i^{d/(d+1)+\epsilon} \le \\ &[(c'\log r/r)m]^{d/(d+1)} \Sigma_{i=1}^{M(r)} n_i^{d/(d+1)+\epsilon} \le \\ &[c'\log r/r]^{d/(d+1)} m^{d/(d+1)} M(r)^{1/(d+1)-\epsilon} n^{d/(d+1)+\epsilon} \end{aligned}$$

Assuming $\epsilon \ge d\delta$, we obtain, for some constant c''

$$T'_j(m,n) \le [D_j c'' \log^{d/(d+1)} r/r^{d/(d+1)} M(r)^{1/(d+1)-\epsilon} + E] m^{d/(d+1)} n^{d/(d+1)+\epsilon} \tag{12}$$

Recalling that $M(r) = r^d \log^{O(1)} r$ the coefficient becomes:

$$D_j c'' \log^{d/(d+1)} r/r^{d/(d+1)} M(r)^{1/(d+1)-\epsilon} + E = D_j c'' \log^{d/(d+1)} r \log^{O(1)/(d+1)-\epsilon} r(1/r)^{d\epsilon} + E$$

If we choose r sufficiently large so that the constant in the equation (12) is less than $[D_j/2 + E]$, and if we choose $D_j = 2E$, we obtain

$$T'_j(m,n) \le D_j m^{d/(d+1)} n^{d/(d+1)+\epsilon}$$

which completes the proof. ∎

It is also easy to show the following corollary:

Corollary 3 *It is possible to report the set $\mathcal{C}_k(A, B)$ in time $T_k(m, n) + |\mathcal{C}_k(A, B)|$.*

Corollary 4 *Under the assumption that the cuttings at (I) and (II) for a constant value of r can be computed in parallel using polylogarithmic time and linearly many processors, then $|\mathcal{C}_k(A, B)|$ can be computed in parallel using $O(T_k(m, n))$ processors and polylogarithmic parallel time.*

Proof. The algorithms of Lemma 3 and of Theorem 8 are based on taking a sample of the surfaces and computing a cutting. For each cell in the cutting we find the points within the region, the surfaces intersecting the region and the surfaces on the positive side of the region. Corresponding to every phase of the sequential algorithm we allocate a processor for each pair point-region and for each pair surface-region. In constant time we can determine the input to the next phase. The number of processors we use is bounded by the number of pairs, which in turns is bounded by the sequential time. The parallel time is bounded by the number of phases of the sequential algorithm. This fact, with the hypothesis that we can compute cuttings in polylogarithmic time, gives us a total polylogarithmic query time. ∎

If we use Clarkson's random sampling techniques [Cla87] to compute the cuttings at (I) and (II) then it is easy to check that constant parallel time and a linear number of processors is sufficient for the construction.

4.2 Finding the shortest vertical segment

We apply the general nested batching technique to the problem of finding the minimum length vertical segment whose end-points are on two distinct lines of $\mathcal{L}$.

Lemma 5 *Given a non-vertical line l and a set of non-vertical lines $\mathcal{L}$, the line in $\mathcal{L}$ with the shortest vertical connecting segment to l is the first line hit by l when translated downward or upward.*

Proof. It follows from elementary geometry. ∎

Our approach to the problem is to find for each line $l \in \mathcal{L}$ the first lines in $\mathcal{L}$ intersected when translating l upward and downward. If l meets more than one line simultaneously we record only one line. Clearly we produce no more than a $O(n)$ pairs of lines which we check for the minimum vertical segment in time $O(n)$. In order to find the set of pairs of lines we use the following theorem from [CEGS89a].

Theorem 9 ([CEGS89a]) *Given a set $\mathcal{L}$ of n lines in R^3 and any $\epsilon > 0$, we can preprocess $\mathcal{L}$ into a data structure whose storage is $O(n^{2+\epsilon})$, so that for any query line l we can decide whether l is above all lines in $\mathcal{L}$ in $O(\log n)$ time and, if so, find the line $l' \in \mathcal{L}$ hit first by l when translated downward.*

The difference between Theorem 9 and the result discussed in Section 2.8 is a subtle but important one. The data structure is based on locating Plücker points of the query line in a given Plücker polytope formed by the hyperplanes associated with the data lines in $\mathcal{L}$. The point location data structure is built by taking a random sample of the hyperplanes and by triangulating the interior of the resulting polytope. In order to obtain the result of section 2.8 any triangulation would do. For the result of Theorem 9 Chazelle et al. carefully devise a triangulation so that, for every "triangle" produced, any Plücker point in the "triangle" would hit the same line in the random sample when translated downward. During the point location query we collect at most a logarithmic number of candidate data lines. Then in extra logarithmic time we select the data line immediately below the query line.

First we establish the following theorem:

Theorem 10 *Given a set $\mathcal{L}$ of n lines in R^3 and any $\epsilon > 0$, we can preprocess $\mathcal{L}$ into a data structure whose storage is $O(n^{4+\epsilon})$, so that for any query line l we can find in $O(\log^2 n)$ time the line $l' \in \mathcal{L}$ hit first by l when translated downward (or upward).*

Proof. Using a data structure similar to that of Theorem 3, we can determine for a query line the lines in $\mathcal{L}$ above and below it. For these lines, which are represented as a disjoint union of canonical sets we apply the result of Theorem 9. We obtain a data structure that for every query line l gives us the line in $\mathcal{L}$ immediately above and the line immediately below. The storage $S(n)$ used by the data structure satisfies the following inequality:

$$S(n) \leq r^4 S(n/r \log r) + r^4 n^{2+\epsilon}$$

from which we have $S(n) = O(n^{4+\epsilon})$. ∎

4.2.1 The batching technique and the shortest vertical segment

Theorem 11 *Given n lines in 3-space it is possible to find the shortest vertical connecting segment in $O(n^{8/5+\epsilon})$ randomized expected time for any $\epsilon > 0$, where the multiplicative constant depends on ϵ.*

Proof. We divide the set of lines $\mathcal{L}$ into two sets L_1 and L_2 of roughly equal size. We solve the problem recursively for the two sets and then we consider the bichromatic problem (i.e. finding the minimum distance between a line in L_1 and a line in L_2). The total complexity is dominated by the solution for the bichromatic case.

We now apply the nested batching technique of Section 4.1 to the data structure of Theorem 10 (Corollary 3). L_1 is used as our data set of lines, L_2 as the query set for the algorithm of Theorem 10.

At the first level of the data structure we have to compute the predicate $tsp(l, l', \vec{v})$, where $\vec{v}$ is the vertical direction. The characteristic dimension of this test is $d_1 = 1$ (the vector $\vec{v}$ is fixed). At the second level we locate Plücker points in an arrangement of Plücker hyperplanes. From results in [APS93] we have characteristic dimension for this test $d_2 = 4$. At the last level we use the special cutting of Theorem 9 with characteristic dimension $d_3 = 2$. At the last level, by locating the Plücker points within the regions of the cutting we are able to find also the closest data line.

By applying the nested batching method to the data structure of Theorem 10 we would like to obtain for each line in L_1 the line in L_2 with the shortest vertical segment. Unfortunately the batched nested method does not quite produce such set of pairs.

At the second level of the algorithm we have sets of Plücker hyperplanes $H(M_\sigma)$ and sets of Plücker points $P(N_\sigma)$ such that the relative orientation is known and uniform (i.e. all lines in M_σ are above all lines in N_σ). By batching the algorithm of Theorem 9 on M_σ as data and N_σ as points we obtain a set $\mathcal{Q}$ of pairs of lines, where each line in N_σ is associated to the line in M_σ immediately below. But, in order to use the nested batching technique, we must sometimes flip the roles of data and queries. Reversing the role of M_σ and N_σ we obtain a set of pairs $\mathcal{Q}'$. The two sets $\mathcal{Q}$ and $\mathcal{Q}'$ can be very different one from the other. On the other hand if $(l, l') \in \mathcal{Q}$ is the pair of lines with minimum connecting vertical segment, then (l', l) is an element of $\mathcal{Q}'$ and it is the pair of lines with minimum connecting vertical segment in $\mathcal{Q}'$. Reversing the role of the two sets of lines we do not loose the information about the minimum connecting segment and the algorithm produces the correct answer. The time bound follows from Theorem 8. ∎

We now extend the result of Theorem 11 to segments. It is enough to solve the bichromatic version of the problem where we are given two sets of segments R (red) and B (blue). The main idea is to find a set Σ of pairs of sets (R_σ, B_σ) with the following properties:

a) For every σ, $R_\sigma \subseteq R$ and $B_\sigma \subseteq B$.

b) Every pair of red, blue segments whose xy-projections intersect is represented in one and only one pair σ.

c) The xy-projection of every segment in R_σ intersects the xy-projection of every segment in B_σ.

For each pair σ we can safely extend the segments into full lines without introducing new vertical segments. We use the result of Theorem 11 on each pair and pick the minimum among the partial results.

In order to find such set Σ we need a few auxiliary definitions. Given a segment s on the plane, let l_s be the line supporting s, $rhp(s)$ the right endpoint and $lhp(s)$ the left endpoint. $Slope(l)$ is the slope of line l.

Lemma 6 *Two segments s_1 and s_2 on the plane intersect if and only if $l(s_1)$ meets s_2 and $l(s_2)$ meets s_1.*

Lemma 7 *A line l meets a segment s in either of these two cases:*

A) $(Slope(l) < Slope(l_s)) \wedge (lhp(s)\ below\ l) \wedge (rhp(s)\ above\ l)$

B) $(Slope(l) > Slope(l_s)) \wedge (lhp(s)\ above\ l) \wedge (rhp(s)\ below\ l)$

Proof. Follows from elementary geometry. ∎

Corollary 5 *Given n segments in 3-space it is possible to find the shortest vertical connecting segment in $O(n^{8/5+\epsilon})$ randomized expected time for any $\epsilon > 0$, where the multiplicative constant depends on ϵ.*

Proof. We build a multilevel data structure using the nested batching technique in order to find pairs of red-blue segments whose xy-projection satisfies the properties of Lemma 6 and Lemma 7. At the first level we compare the slopes of the segments. This gives us a test of characteristic dimension $d = 1$. Once we have groups of segments whose relative slope is uniform we can test at the next levels the above/below relations using standard point/line duality in dimension $d = 2$. At the last level we obtain pairs of sets of segments (blue and red) such that all red segments meet all blue segments. We can extend the corresponding red-blue segments in 3-space into full lines without introducing any new and potentially dangerous candidates for the minimum vertical connecting segment test. At the last level we therefore add a data structure similar to those of Theorem 11. The characteristic dimension of the whole data structure is therefore dominated by $d = 4$. The time bound derives from the application of the nested batching technique of Theorem 8. ∎

4.2.2 Finding the longest vertical segment

Suppose we want to find the *longest* among the vertical segments connecting two lines in $\mathcal{L}$. Although this problem has a formulation similar to finding the *shortest* vertical segment, we obtain an asymptotically faster algorithm.

Theorem 12 *Given a set $\mathcal{L}$ of n lines in R^3, the maximum length vertical segment connecting two lines in $\mathcal{L}$ can be found in $O(n^{4/3+\epsilon})$ randomized expected time.*

Proof. We partition $\mathcal{L}$ into two subsets L_1, and L_2 of equal size and we solve the problem recursively in each set. It is thus enough to solve the bichromatic case. Suppose without loss of generality the pair (l_1, l_2), with $l_1 \in L_1$ and $l_2 \in L_2$, achieves the maximum and moreover l_1 is above l_2. Let δ be the value of the maximum. If we shift every line in L_1 downward of a distance δ, we obtain a new set L_1'. From the fact that δ is the maximum distance follows that every line in L_1' is below every line in L_2. The two sets L_1' and L_2 satisfy the towering property (see [CEGS89a]) which can be tested in $O(n^{4/3+\epsilon})$ randomized expected time. This algorithm can be seen as a special instance of the nested batching technique of Section 4.1. In this case we have a first level to test the *tsp* condition with characteristic dimension $d_1 = 1$ and a second level to test the relative orientation with characteristic dimension $d_2 = 2$.

The algorithm for testing the towering property can be used as the oracle of Megiddo's parametric search method. A parallel version of the oracle can be devised using the inherent tree-structure of the nested batching scheme of section 4.1, as shown in Corollary 4. We obtain logarithmic parallel time using no more than $O(n^{4/3+\epsilon})$ processors. The parametric search approach allows us to find the value of δ which is the maximum bichromatic distance. ∎

The result of Theorem 12 extends, with the same time bound, to sets of segments, since the characteristic dimension of each level is no more than $d = 2$, as follows from Lemma 6 and 7 and from the algorithm to test the towering property [CEGS89b].

4.3 Finding the shortest segment

Given a set $\mathcal{L}$ of n lines in R^3 the problem is to find the shortest segment connecting two lines. A more formal statement of the problem is the following: compute $Min(\mathcal{L})$, where

$$Min(\mathcal{L}) = \min_{l',l\in\mathcal{L},l\neq l'} \min_{p\in l,q\in l'} |\bar{pq}|.$$

Computing $Min(\mathcal{L})$ fits nicely into the general schema of Megiddo's parametric search. Given a set $\mathcal{L}$ of n lines in R^3 we partition $\mathcal{L}$ into two disjoint sets R (red) and B (blue) of roughly $n/2$ lines each. If $T(n)$ is the time of the algorithm for finding the closest pair we have the following recursive inequality:

$$T(n) \leq 2T(n/2) + T'(n/2, n/2)$$

where $T'(n,m)$ is the time sufficient to solve the bichromatic version of the problem (i.e. find the shortest segment connecting a "red" line with a "blue" line).

Let us denote as $P'(R,B,t)$ the program that, given a set R of red lines and a set B of blue lines, counts the number of red lines at distance less than t from any blue line. Clearly $P'(R,B,t)$ is monotone in t. Using a method in [Pel93] we can easily check if $P'(R,B,0) = 0$. Finding the shortest bichromatic distance is equivalent to finding the minimum value of t such that $P'(R,B,t) > 0$.

4.3.1 The oracle

In this subsection we give the algorithm underlying the program $P'(R,B,t)$. The problem of counting pairs of lines within distance t can be seen as the problem of counting the number of intersections among the set R of lines and the set B' of cylinders of radius t whose axes are lines in B.

Lemma 8 *Given a set of n lines and a set of m cylinders of the same radius, there is an algorithm that counts all line-cylinder intersections in time*

$$Am^{5/6}n^{5/6+\epsilon} + Bm^{1+\epsilon} + Cn\log^2 m,$$

where the constants A,B and C depend on ϵ.

Proof. A line l in 3-space is representable in a canonical way as the intersection of a plane parallel to the x-axis and of a plane parallel to the y-axis[3].

$$y = az + b$$
$$x = cz + d$$

The four numbers (a, b, c, d) uniquely represent a line in 3-space. Thus we can map a line l to a point $p(l)$ in real parametric space R^4. A cylinder c of fixed radius t is representable in a canonical way as a point $p(c)$ in real parametric space R^4 (as a matter of fact we need only to map the axis of the cylinder).

Given a line l we define as the dual surface of l the set $S(l)$ of all the points $p(c)$ where c is a cylinder of radius t tangent to l. Clearly $S(l)$ is an algebraic variety (of co-dimension 1) in R^4. In particular there is a polynomial $q_l(.)$ such that $S(l) = \{x \in R^4 | q_l(x) = 0\}$. Symmetrically we define as the dual surface of the cylinder c the set $S(c)$ of the points $p(l)$ where l is tangent to c. Again $S(c)$ is an algebraic variety of co-dimension 1. And we know there exists a polynomial $r_c(.)$ such that $S(c) = \{x \in R^4 | r_c(x) = 0\}$. The relative position of the point $p(l)$ and surface $S(c)$ determines whether l meets c or not.

Using the data structure in [CEGS89b] we can locate the points $p(l)$ in the arrangement of algebraic surfaces $S(c)$. The size is $O(n^{2d-3+\epsilon})$. The query time is $O(\log n)$. We apply the batching argument developed in Section 4.1 for locating m points in an arrangement of n surfaces. The time bound follows from the batching technique. ∎

Theorem 13 *Given n lines in 3-space, it is possible to find the shortest connecting segment in $O(n^{5/3+\epsilon})$ randomized expected time for any $\epsilon > 0$, where the multiplicative constant depends on ϵ.*

Proof. The algorithm of Lemma 8 (based on the data structure in [CEGS89b]) and on the nested batching technique has a tree structure of logarithmic depth. It is relatively easy to transform the sequential algorithm into a parallel algorithm using corollary 4, as required by Megiddo's technique. ∎

The oracle used in Theorem 13 is a counting oracle which counts the number of intersections. In order to solve the closest line problem is sufficient for the oracle to test whether the number of incidences is greater than 0. It is likely that such an algorithm can attain a better asymptotic bound than the counting oracle. Such improved bound would imply a faster algorithm for finding the closest line. The improved time bound of [CEGS92] is based on a similar observation.

[3]We cannot represent horizontal lines but, with a suitable space transformation, we can make sure that we do not have any.

5 Conclusions and open problems

Solving problems on lines in 3-space is often a necessary ingredient for solving problems on sets of polyhedra in 3-space. In this paper we solved an on-line intersection query problem for lines and half-planes in 3-space, which is an essential ingredient of a fast method for collision-free placement queries amidst polyhedral obstacles in 3-space. In the same spirit, we gave algorithms for solving a few neighbor problems on lines in 3-space. Here are a few open questions: Is it possible to extend the solution of neighbor problems of lines to segments and polyhedra? How can we combine neighbor information and collision free placements to plan a collision-free movement of a polyhedral object in a polyhedral environment? Is it possible to have a dynamic (under insertions and deletions) version of the data structure of Theorem 7? An initial result in the direction of solving this last question is in [AM91].

6 Acknowledgments

I wish to thank to Subhash Suri for proposing the problem of finding the closest pair of lines. Thanks to Raimund Seidel, Micha Sharir and Leonidas Guibas for useful discussions. Thanks also to Pankaj Agarwal for his comments on an early version of the paper and to Professor Julian Ullmann for his comments.

References

[AASS90] P.K. Agarwal, B. Aronov, M. Sharir, and S. Suri. Selecting distances in the plane. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 321–331, 1990.

[AESW91] P.K. Agarwal, H. Edelsbrunner, O. Schwarzkopf, and E. Welzl. Euclidean minimum spanning tree and bichromatic closest pair. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 203–210, 1991.

[Aga91] P.K. Agarwal. Geometric partitioning and its applications. Technical Report CS-1991-27, Dept. of Computer Sci. Duke University, July 1991.

[AM91] P.K. Agarwal and J. Motoušek. Dynamic half-space range reporting and its applications. Technical Report CS-1991-43, Computer Science Dept. Duke University, December 1991.

[AM92a] P.K. Agarwal and J. Matoušek. Range searching with semialgebraic sets. In *Proc. of the 17th Symp. on Mathematical Foundations of Computer Science*, number 629 in Lecture Notes in Computer Science, pages 1–13, 1992.

[AM92b] P.K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*, pages 517–526, 1992.

[APS93] B. Aronov, M. Pellegrini, and M. Sharir. On the zone of an algebraic surface in a hyperplane arrangement. *Discrete & Computational Geometry*, 9(2):177–188, 1993.

[AS90] B. Aronov and M. Sharir. Triangles in space or building (and analyzing) castles in the air. *Combinatorica*, 10(2):137–173, 1990.

[AS91a] P. K. Agarwal and M. Sharir. Applications of a new space partitioning technique. In *Proceedings of the 1991 Workshop on Algorithms and Data Structures*, number 519 in Lecture Notes in Computer Science, pages 379–391. Springer Verlag, 1991.

[AS91b] P. K. Agarwal and M. Sharir. Counting circular arc intersections. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 10–20, 1991.

[AS92] B. Aronov and M. Sharir. Castles in the air revisited. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 146–256, 1992.

[Aur91] F. Aurenhammer. Voronoi Diagrams - A Survey of a Fundamental Geometric Data Structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[CD80] B. Chazelle and D.P. Dobkin. Detection is easier than computation. In *Proceedings of the 21th IEEE Symposium on Foundations of Computer Science*, pages 146–153, 1980.

[CEG+90] B. Chazelle, H. Edelsbrunner, L. Guibas, R. Pollack, R. Seidel, M. Sharir, and J. Snoeyink. Counting and cutting circles of lines and rods in space. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science*, 1990.

[CEGS89a] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Lines in space: combinatorics, algorithms and applications. In *Proc. of the 21st ACM Symposium on Theory of Computing*, pages 382–393, 1989.

[CEGS89b] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. A singly exponential stratification scheme for real semi-algebraic varieties and its applications. In *Proceedings of the 16th International Colloquium on Automata, languages and Programming*, number 372 in Lecture Notes in Computer Science, pages 179–193. Springer Verlag, 1989.

[CEGS92] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Diameter, width, closest line pair and parametric search. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 120–129, 1992.

[CK89] L.P. Chew and K. Kedem. Placing the largest similar copy of a convex polygon among polygonal obstacles. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 167–174, 1989.

[Cla87] K.L. Clarkson. New applications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2:195–222, 1987.

[Col87] R. Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. of ACM*, 34:200–208, 1987.

[CS90] B. Chazelle and M. Sharir. An algorithm for generalized point location and its applications. *J. of Symbolic Computation*, (10):281–309, 1990.

[CSW90] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 23–33, 1990.

[dBHO+91] M. de Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. van Kreveld. Efficient ray-shooting and hidden surface removal. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 21–30, 1991.

[DE84] D. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. In *25th FOCS*, pages 387–392, 1984.

[DK83] D.P. Dobkin and D.G. Kirkpatrick. Fast detection of polyhedral intersection. *Theoretical Computer Science*, 27:241–253, 1983.

[DK90] D. Dobkin and D. Kirkpatrick. Determining the separation of preprocessed polyhedra: a unified approach. In *Proc. of the 17th International Colloqium on Automata, Languages and Programming*, pages 400–413, 1990.

[Ede87] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*. Springer Verlag, 1987.

[EGS88] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity of many faces in arrangements of lines and segments. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 44–55, 1988.

[EGS90] H. Edelsbrunner, L. Guibas, and M. Sharir. The complexity and construction of many faces in arrangements of lines and segments. *Discrete & Computational Geometry*, 5:161–196, 1990.

[EMP+82] H. Edelsbrunner, H. Mauer, F. Preparata, E. Welzl, and D. Wood. Stabbing line segments. *BIT*, (22):274–281, 1982.

[EOS86] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computing*, 15:341–363, 1986.

[FHS89] J. Friedman, J. Hershberger, and J. Snoeyink. Compliant motion in a simple polygon. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 175–186, 1989.

[GSS88] L.J. Guibas, M. Sharir, and S. Sifrony. On the general motion planning problem with two degrees of freedom. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 289–298, 1988.

[Gui91] L. Guibas. Personal communicationion. November 1991.

[Hal91] D. Halperin. On the complexity of a single cell in certain arrangements of surfaces in 3-space. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 314–323, 1991.

[HO89] D. Halperin and M. Overmars. Efficient Motion Planning of an L-shaped Object. In *Proceedings of the 5th ACM Symposium on Computational Geometry*, pages 156–166, 1989.

[KS88] K. Kedem and M. Sharir. An automatic motion planning system for a convex polygonal mobile robot in 2-d polygonal space. In *Proceedings of the 4th ACM Symposium on Computational Geometry*, pages 329–340, 1988.

[LS87a] D. Leven and M. Sharir. An efficient simple motion planning algorithm for a ladder moving in two-dimensional space amidst polygonal barriers. *J. of Algorithms*, 8:192–215, 1987.

[LS87b] D. Leven and M. Sharir. Planning a purely translational motion for a convex object in two-dimensional space using generalized voronoi diagrams. *Discrete & Computational Geometry*, 2:9–31, 1987.

[Mat90] J. Matoušek. Cutting hyperplane arrangements. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 1–9, 1990.

[Mat91] J. Matoušek. Efficient partition trees. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 1–9, 1991.

[Meg83] N. Megiddo. Applying parallel computation algorithms in the design of sequential algorithms. *J. of ACM*, 30:852–865, 1983.

[Meh84] K. Mehlhorn. *Multidimensional Searching and Computational Geometry.* Springer Verlag, 1984.

[Mit70] D.S. Mitrinovic. *Analytic inequalities.* Springer Verlag, 1970.

[MS85] K. Mehlhorn and K. Simon. Intersecting two polyhedra one of which is convex. In *Proc. of Fundamentals of Computation Theory*, number 199 in Lecture Notes in Computer Science, pages 534–542. Springer Verlag, 1985.

[Pel90] M. Pellegrini. Stabbing and ray shooting in 3-dimensional space. In *Proceedings of the 6th ACM Symposium on Computational Geometry*, pages 177–186, 1990.

[Pel91] M. Pellegrini. *Combinatorial and Algorithmic Analysis of Stabbing and Visibility Problems in 3-Dimensional Space.* PhD thesis, New York University–Courant Institute of Mathematical Sciences, February 1991. Courant Institute, Robotics Lab. Report Number 241.

[Pel92] M. Pellegrini. A new algorithm for counting circular arc intersections. Technical Report TR-92-010, International Computer Science Institute, February 1992.

[Pel93] M. Pellegrini. Ray shooting on triangles in 3-space. *Algorithmica*, 9:471–494, 1993.

[PS85] F.P. Preparata and M.I. Shamos. *Computational Geometry: an Introduction.* Springer Verlag, 1985.

[PS92] M. Pellegrini and P. Shor. Finding stabbing lines in 3-space. *Discrete & Computational Geometry*, 8:191–208, 1992.

[Som51] D. M. H. Sommerville. *Analytical geometry of three dimensions.* Cambridge University Press, Cambdrige, England, 1951.

[SS89] J.T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. In D. Kapur and J.L. Mundy, editors, *Geometric Reasoning*, pages 157–169. The MIT press, 1989.

[Sto89] J. Stolfi. Primitives for computational geometry. Technical Report 36, Digital Systems Research Center, 130 Lytton Avenue, Palo Alto, California, 1989.

[Tol91] S. Toledo. Extremal polygon containment problems. In *Proceedings of the 7th ACM Symposium on Computational Geometry*, pages 176–185, 1991.

[Vai89] P.M. Vaidya. An $O(n \log n)$ algorithm for the all-nearest-neighbors problem. *Discrete & Computational Geometry*, 4:101–115, 1989.