On Point Location and Motion Planning among Simplices*

Marco Pellegrini†

May 20, 1996

Abstract

Let U be a set of n possibly intersecting $(d-1)$-simplices in d-space, for $d \geq 2$; and let $\mathcal{A}(U)$ be the arrangement of U. Let $K = |\mathcal{A}(U)|$ be the number of faces of any dimension in the arrangement of U. A data structure is described that uses storage $O(n^{d-1+\epsilon} + K)$ and is built *deterministically* in time $O(n^{d-1+\epsilon} + K \log n)$, where $\epsilon > 0$ is an arbitrarily small constant, such that the face of $\mathcal{A}(U)$ containing a query point is located in time $O(\log^3 n)$. If two query points are in the same cell of $\mathcal{A}(U)$ a collision-free path connecting them is produced. This result is obtained exploiting powerful and so far overlooked properties of sparse nets introduced by Chazelle [Cha93]. If the $(d-1)$-simplices in U have pairwise disjoint interiors and $d \geq 3$, improved bounds are obtained. A data structure is described that uses $O(n^{d-1})$ storage and is built deterministically in time $O(n^{d-1})$ such that point location queries are solved in time $O(\log n)$. Also, as a by-product, this method gives the first optimal worst-case algorithm for triangulating a non-simple polyhedron in 3-space.

Key words. arrangements of simplices, point location, sparse nets, motion planning, triangulations

AMS subject classifications. 68P05, 68Q25, 68Q40, 68U05

1 Introduction

1.1 Point Location: definition and previous results

Point location is a central problem in computational geometry [PS85, Ede87, Meh84] and a continuous stream of research papers have been published on this topic from the early days of computational geometry till today. Let U be a set of n possibly intersecting $(d-1)$-simplices in E^d. For example we consider segments on the plane, triangles in 3-space, tetrahedra in 4-space, etc.. Such a set U decomposes E^d into open cells of dimension d and into relatively open cells of dimension k, with $0 \leq k < d$. This collection of cells is called the *arrangement* $\mathcal{A}(U)$ induced by U in E^d. We

*A preliminary version of this work appeared in the *Proceedings of the 26th ACM Symposium on Theory of Computing.*

†Research supported by C.N.R. within the 1993-1994 Senior Visiting Scientist program at ICSI.

Affiliation: Istituto di Matematica Computazionale del C.N.R., via S. Maria 46, Pisa, Italy.
e-mail: pellegrini@iei.pi.cnr.it

associate a unique identifier to each cell in $\mathcal{A}(U)$ from a universe of distinct labels. Let K be the cardinality of $\mathcal{A}(U)$, also referred to as the combinatorial complexity of the arrangement of U. The point location problem consists in preprocessing U into a data structure so that subsequently the following query can be answered efficiently:

(A) Given a query point q find any cell in $\mathcal{A}(U)$ whose relative interior contains q.

We consider in this paper also two variations of the point location problems for which slightly different bounds are known.

(B) Given a query point q, determine whether q is incident to any element of U (incidence query).

(C) Given a query point q, determine the element of U immediately below q in a fixed direction (this variation is also called vertical ray-shooting problem).

Several results are known when restrictions are placed on the input set U and on the dimension d. We assume that the dimension d is a small constant and the multiplicative constants in the big-Oh notation depend on d.

Hyperplanes. When U is a set of hyperplanes H in E^d, for $d \geq 2$, Clarkson [Cla87] solves point location queries in time $O(\log n)$ using a data structure of size[1] $O(n^{d+\epsilon})$ which is constructed in *expected* time $O(n^{d+\epsilon})$. Chazelle and Friedman [CF92] solve the vertical ray-shooting problem in $O(\log n)$ time, using a data structure of size $O(n^d)$, but with an high pre-processing time. Chazelle [Cha93] solves point location queries in time $O(\log n)$ using a data structure of size $O(n^d)$ built deterministically in time $O(n^d)$. Chazelle [Cha93] solves incidence queries in $O(\log n)$ time using a data structure of size $O(n^d/(\log n)^{d-1})$ which is built deterministically in time $O(n^d/(\log n)^{d-1})$. Matoušek [Mat93] solves vertical ray-shooting queries within the same bounds.

Planar Maps. The point location problem has been studied extensively for the case when U is a set of segments in E^2 forming a planar map (e.g. [LP77, Pre81, LT80]). Even a partial list of all the important results on this problem is beyond the scope of this paper and for a more complete discussion of point location algorithms for planar maps we refer the reader to the book of Preparata and Shamos [PS85] and Edelsbrunner [Ede87]. Point-location queries (variations (A),(B) and (C)) are answered in time $O(\log n)$, using $O(n)$ storage and $O(n \log n)$ preprocessing time (see e.g. [Kir83], [EGS86], [Col86] and [ST86]). The techniques used in [Kir83, EGS86] are essentially based on properties of planar maps that do not hold for generalizations of such maps to 3 and higher dimensional spaces. The results in [Col86, ST86] are based on the sweeping paradigm. These algorithms sweep a line l on the plane and maintain dynamically (or semi-dynamically) the intersection of l with the cells in $\mathcal{A}(U)$. For the plane the problem is to maintain dynamically a 1-dimensional arrangement, which is relatively easy. Preparata and Tamassia [PT88] give an efficient method for dynamic maintenance of planar maps, which is used in [PT89] in connection with a plane sweeping in 3-space. The sweeping paradigm fails in dimension 4 since we do not have yet an efficient method for dynamic maintenance of 3-dimensional maps.

[1] Here and throughout the paper we denote with ϵ an arbitrarily small positive real number. The multiplicative constant in the big-Oh notation may depend on ϵ and it goes to infinity as ϵ tends to 0.

Pairwise Interior-Disjoint Simplices in $d \geq 3$. If U is a set of n interior-disjoint triangles in E^3 forming a *convex arrangement*, the method of [PT89, GT91] solves point location queries in time $O(\log^2 n)$ using $O(n \log n)$ storage and preprocessing time. If the set U is still formed by interior-disjoint triangles in E^3 but the arrangement of U is not composed of convex cells, Mulmuley [Mul91] gives a method that solves point location queries in time $O(\log^2 n)$, using $O(n^2)$ storage and $O(n^2 \log n)$ preprocessing time.

Intersecting Simplices. When U is a set of hyperplanes in general position or a set of interior-disjoint simplices, the size of the arrangement of U is bounded tightly by a function of n. Therefore the upper bounds are expressed as a function of one variable n. When we allow the simplices in U to intersect each other we have that K is in the range $[n, n^d]$. Therefore we are interested in algorithms whose time and storage bounds depend explicitly on K. If U is a set of n possibly intersecting segments in the plane, we can use the optimal algorithm of Chazelle and Edelsbrunner [CE92] to construct $\mathcal{A}(U)$ in time $O(n \log n + K)$. Since $\mathcal{A}(U)$ is a planar map of size $O(K)$ we can use the point location methods for planar maps and solve point location problems in time $O(\log n)$ using $O(K)$ storage and $O(n \log n + K)$ preprocessing time.

Recently and independently de Berg, Guibas and Halperin [dBGH94] have extended the sweeping plane technique in [Mul91] to sets of intersecting triangles in 3-space. They build a decomposition of $\mathcal{A}(U)$ into $O(n^{2+\epsilon} + K)$ convex cells using $O(n^{2+\epsilon} + K \log n)$ time. Point location queries are answered in time $O(\log^2 n)$. Since the approach is based on a sweeping plane it suffers of the limitations mentioned before for extensions to higher dimensions.

1.2 New results on point location

Intersecting simplices. The main contribution of this paper is a method for solving point location queries in an arrangement of n possibly intersecting simplices in E^d, for $d \geq 2$. Our time and storage bounds depend explicitly on the number of cells in the arrangement of simplices $K = |\mathcal{A}(U)|$. We answer point location queries in time $O(\log^3 n)$, using a data structure of size $O(n^{d-1+\epsilon} + K)$ which is built deterministically in time $O(n^{d-1+\epsilon} + K \log n)$. We answer incidence queries in time $O(\log^2 n)$ and vertical ray-shooting queries in time $O(\log^3 n)$, using $O(n^{d-1+\epsilon} + K)$ preprocessing time and storage.

Non-intersecting simplices. If the simplices in U are pairwise interior-disjoint, we obtain improved time and storage bounds for $d \geq 3$. We give a deterministic algorithm that builds a data structure of size $O(n^{d-1})$ in time $O(n^{d-1})$. Point location, vertical ray-shooting and incidence queries are answered using this data structure in time $O(\log n)$.

1.3 Motion planning among simplices

The results in this paper are relevant for motion planning problems. For background material on motion planning we refer the reader to [SS90, Lat91]. The motion planning problem in its simplest (purely geometrical) form is defined in [SS89] as follows:

> Let B be a robot system consisting of a collection of rigid subparts having a total of k degrees of freedom, and suppose that B is free to move in two- or three-dimensional

space V amidst a collection of obstacles whose geometry is known to the robot system. The *motion planning problem* for B is: Given an initial position Z_1 and a desired final position Z_2 of B, determine whether there is a continuous obstacle-avoiding motion of B from Z_1 to Z_2, and if so plan such a motion.

Since B has k degrees of freedom a position of B is representable as a point in E^k. The subset of E^k representing positions in which B does not collide with obstacles in V is called the *space of free positions* $FP \subset E^k$. Thus the motion planning problem is reduced to the problem of finding a path connecting Z_1 and Z_2 in FP. If FP is a semi-algebraic set in E^k, Schwartz and Sharir [SS83] solve the motion planning problem in time polynomial in the number of algebraic constraints defining FP and their maximal degree and doubly exponential in k. Canny [Can87] gives an algorithm with running time roughly $O(n^k)$, where n is the number of algebraic constraints.

Point-Robots. If the robot B is a point in d-dimensional space and the obstacles are simplices, we have the simplest d-dimensional motion planning problem. Aronov and Sharir [AS92] give an $O(n^{d-1}\log n)$ upper bound on the complexity of one cell in an arrangement of n $(d-1)$-simplices in E^d. For $d = 3$, this bound and method in [AS90] result in an algorithm for computing one cell of $\mathcal{A}(U)$ in $O(n^{2+\epsilon})$ expected time.

A recent result of de Berg, Matoušek and Schwarzkopf proves that any two points connected by a collision free path among n possibly intersecting *convex* obstacles in E^d can also be connected by a piece-wise linear path composed of $O(n^{(d-1)\lfloor d/2+1\rfloor})$ segments. Algorithms to compute such paths are provided only for $d = 2$.

1.4 New results in motion planning

The technique we develop for solving point-location queries among simplices in E^d gives us with little extra effort a method for planning motions of point-robots among obstacles represented by simplices in E^d. Our result is relevant when the initial position is not fixed but it is rather part of the query, as it is often the case during the design and simulation of robotic systems. In this case it is more convenient to compute the whole arrangement of simplices once during preprocessing rather than computing single cells on-line. Moreover, since motion planning problems for more general robots can be solved by reduction to motion planning for point-robots in higher dimensional space, we can use our method to compute the free space FP whenever the expanded obstacles in configuration space are simplices. The free space FP is in general only a subcomplex of $\mathcal{A}(U)$ and in the worst case the two sets can have complexity differing by orders of magnitude. On the other hand, in more practical situations the arrangement $\mathcal{A}(U)$ is likely to be sparse when the obstacles are far from each other in terms of the size of the robot B. Our algorithm is able to take advantage of the sparsity of $\mathcal{A}(U)$ and in such cases it provides a more efficient algorithm than general roughly $O(n^k)$ solutions [Can87] or solutions based on extending the simplices into hyperplanes.

For example, we use our results to find the isotopy classes of lines induced by a set of polygons in 3-space which are on a family of parallel planes. We map lines in E^3 as points in E^4 and edges in E^3 as simplices in E^4. The arrangement $\mathcal{A}(U)$ of these simplices represents the isotopy classes of lines. Thus using $O(n^{3+\epsilon} + K)$ storage we can find in time $O(\log^3 n)$ the isotopy class of a query line and produce collision-free motions of a line. In this context K is related to the number of lines that meet four edges of the input polygons. This particular motion planning problem arises in controlling a radiation beam for radio-surgery of the brain. The model of the human brain as a collection of parallel polygons can be obtained in a natural way by using a CAT scanner.

1.5 Triangulations

Often point location methods are based on decomposing the free space E^d/U into convex parts. Such decompositions are referred to generically as *triangulations.* The problem of building triangulations is important in its own right with a wide range applications. Most of the known results are for $d = 2$ (see e.g. [Cha87]) and fewer for higher dimensional spaces (i.e. $d \geq 3$). In this paper we consider the problem in $d \geq 3$ when U is a set of pairwise interior-disjoint $(d-1)$-simplices. This class of input is important because includes boundaries of general polyhedra in E^d.

A concept similar to that of a triangulation is that of a *Binary Space Partition.* Given a set U of n interior-disjoint triangles in 3-space, a Binary Space Partition of E^3 induced by U (denoted $BSP(U)$) is a hierarchical partition of E^3 into *convex cells* associated with the nodes of a binary tree. The root is associated with the whole space E^3 and the regions associated with the children of node v form a convex partition of the region associated with v. The leaves are associated with regions whose interior does not meet any triangle in U. Paterson and Yao show in [PY90] how to obtain a BSP's of size $O(n^2)$ in time $O(n^3)$ for $d = 3$. In higher dimensional space the method in [PY90] builds a BSP of size $O(n^{d-1})$ in time $O(n^{d+1})$.

For the special case when U is the boundary of a polyhedron in E^3 and U is connected, Chazelle [Cha84] gives a triangulation of size $O(n^2)$ which is built in time $O(n^2)$. Both bounds are worst case optimal. The method in [Cha84] relies on the connectivity of the boundary and therefore this method does not extend naturally to polyhedra with disconnected boundary. Some triangulations for special polyhedral sets in E^3 are discussed in [Ber93].

1.6 New results on triangulations

In this paper we show a method for building a triangulation induced by pairwise interior-disjoint $(d-1)$-simplices in d-dimensional space ($d \geq 3$). We obtain a *deterministic* algorithm that builds a triangulation of size $O(n^{d-1})$ in time $O(n^{d-1})$. To our knowledge no better algorithm is known for dimension $d \geq 4$ for the case when U is a set of pairwise interior-disjoint simplices even forming a connected boundary. The size and time bounds of our algorithm are worst case optimal for $d = 3$, as follows from an $\Omega(n^2)$ lower bound in [Cha84]. For $d = 3$, our result is the first worst case optimal $\Theta(n^2)$ algorithm for triangulating a non-simple polyhedron in 3-space.

1.7 On cuttings and sparse nets

The main tools used in this paper are *cuttings* and *sparse nets.* Given n hyperplanes in E^d, a $(1/r)$-cutting is a collection of simplices with disjoint interiors which together cover E^d and such that the interior of any simplex is cut by at most n/r hyperplanes. Chazelle describes a deterministic algorithm to construct $(1/r)$-cuttings of size $O(r^d)$ in time $O(nr^{d-1})$ [Cha93]. This algorithm is the basis of improved methods for locating points in arrangements of hyperplanes and for many other geometric problems. Matoušek uses the algorithm in [Cha93] in order to improve time and storage bounds on the simplex range searching problem [Mat92].

The algorithm for constructing $(1/r)$-cutting relies on properties of the so called *sparse nets* which are special subsets of an input set H of hyperplanes. Intuitively, a sparse net R of a set H for a region s of E^d is a subset of H such that the number of vertices in the arrangement of H within s is well approximated by the number of vertices of the arrangement of R within s.

In [Cha93, Mat92] this property of sparse nets is applied to objects (hyperplanes) that produce "globally dense" arrangements (i.e. every d hyperplanes in general position meet in one point), by exploiting the "local sparsity" of the arrangements. Previous randomized (e.g. [Cla87]) and deterministic (e.g. [Mat91]) techniques are already quite efficient for globally dense arrangement and therefore we gain polylogarithmic or small polynomial factors by using the cuttings described in [Cha93]. In this paper we show that sparse nets are very useful computational tools for objects like $(d-1)$-simplices that produce "globally sparse" arrangements (i.e. not every d $(d-1)$-simplices meet, and in general there can be between 0 and $O(n^d)$ points meeting d $(d-1)$-simplices).

1.8 The method

We will show three data structures to solve incidence queries, vertical ray shooting queries and point location queries. Each data structure uses the previous one. We tackle first the problem of solving incidence queries. The general strategy is a classical geometric divide-and-conquer. We select a subset $N \subset U$ and we use N to partition E^d into convex cells of constant descriptive complexity (also referred to as *elementary* cells). We compute the simplices of U intersecting each elementary cell and we apply the same method recursively within the elementary cell. At each recursive call the size of the problem is reduced and eventually we obtain elementary cells which do not intersect any simplex in U. At this point we stop the construction.

The elementary cells generated during the preprocessing are organized in a search tree, which has the flavour of a multi-dimensional extension of the *hereditary segment tree* [CEGS89, dBGH94], or of a multi-dimensional extension of search trees in the *trapezoid method* for planar point location [Pre81, PS85].

We answer the incidence query by visiting the tree until either we detect an incidence of the query point with a simplex in U, or we reach a leaf and we conclude that there is no incidence.

The main contribution of this paper is in the analysis techniques that allow us to derive the input sensitive bounds on the storage and time bounds. The difficult task is to make sure that we obtain at each level of the search tree a fine tuned bound on the number of elementary cells. If we use a coarse bound on the number of elementary cells we will end up easily with total storage $O(n^d)$, which falls short of our goal of a bound roughly $O(K + o(n^d))$. This tight control on the size of the construction is attained exploiting the properties of the *sparse nets* [Cha93].

Chazelle uses the sparse nets in order to produce hierarchical cuttings for sets of hyperplanes. These cuttings than are the main ingredient of his method for solving point location queries in arrangements of hyperplanes [Cha93]. Here we follow the same general approach of [Cha93] with two main differences. Algorithmically, we construct elementary cells in d-space and also elementary cells in $(d-1)$-space, which are then "lifted" to d-space. In the analysis, we use finer arguments on the number of elementary cells so to obtain a final bound that depends explicitly on K.

Once we have the data structure for incidence queries we add auxiliary data structures that enable us to solve vertical ray-shooting queries, within the same asymptotic bounds for preprocessing and storage.

We solve incidence queries and vertical ray-shooting queries using the fact that these are *decomposable* queries, that is, knowing the solution for sets of simplices U_1 and U_2 we can easily find the solution for $U_1 \cup U_2$. This does not hold for point location queries among simplices. In fact, if we partition the input set U we may loose essential connectivity information on the set $E^d \setminus U$. When the input is formed by hyperplanes, two points are in the same cell of the arrangement if and only if the two points are on the same side of each hyperplane. For simplices this simple characterization does not hold any more and we need more powerful methods to identify the cells in $\mathcal{A}(U)$.

Our method to overcome these difficulties consists in building a sequence of sets of graphs which encode the topological information. This sequence is indexed by the dimension of the subspaces in which our arrangement is contained, starting from dimension two up to dimension d. In this construction ray-shooting operations are an essential ingredient needed to "lift" the topological information from lower dimensional to higher dimensional arrangements. At the end of the preprocessing phase we have built the connectivity graph for $\mathcal{A}(U)$ and each face is properly labelled. Given a query point p, we use a sequence of d vertical ray-shooting queries to find a vertex of $\mathcal{A}(U)$ in the same face

When the set U is formed of pairwise disjoint simplices we use a more direct line of attack that aims at generating a hierarchical triangulation of the free space $E^d \setminus U$. The point location procedure is a natural product of the hierarchical triangulation.

The paper is organized as follows. Section 2 recalls the main properties of the sparse nets [Cha93]. In Section 3 we give and analyze the algorithms for solving incidence queries and vertical ray-shooting queries. In Section 4 we give and analyze the algorithm for solving point-location queries. In Section 5 we give the algorithm to construct a triangulation induced by disjoint simplices, as a corollary we obtain a method for solving point location queries for this particular type of input set.

2 Sparse Nets and Hierarchical Cuttings

In this section we give the basic definitions and lemmas from [Cha93] which form the background to the results shown in the subsequent sections. Let H be a set of n hyperplanes in Euclidean d-dimensional space E^d. We assume that H is in *general position*, meaning that exactly d hyperplanes meet in a common point. Let $R \subseteq H$ be a subset of $\rho \leq n$ hyperplanes. For a segment e, let R_e (resp H_e) be the number of hyperplanes in R (resp. H) intersecting e but not containing e. For a d-simplex s, let $v(R, s)$ (resp $v(H, s)$) be the number of vertices of the arrangement created by R (resp. H) contained in the relative interior of s. We denote by $H(s)$ the subset of hyperplanes of H intersecting s. All definitions and lemmas in this Section extend to improper simplices (i.e. with some vertices at infinity). Let r be a positive integer number.

Definition 1 *R is a $(1/r)$-approximation for H if, for any segment e:*

$$\left|\frac{R_e}{|R|} - \frac{H_e}{|H|}\right| < \frac{1}{r}.$$

Definition 2 *R is a $(1/r)$-net for H if for any segment e, $H_e > n/r$ implies $R_e > 0$.*

Definition 3 *R is a sparse $(1/r)$-net for (H, s) if for any segment e, $H_e > n/r$ implies $R_e > 0$; and the following inequality holds:*

$$v(R, s) \leq 4(\frac{|R|}{|H|})^d v(H, s).$$

Definition 4 *A $(1/r)$-cutting for H is a partition of E^d into interior-disjoint simplices such that any simplex meets at most n/r hyperplanes of H. The number of simplices in the partition is called the size of the cutting.*

Let us suppose that we are given a set H of n hyperplanes in E^d and a collection Θ of d-simplices which partitions of E^d. Let s be one of these d-simplices in Θ, let r_0 be a constant and $i > 0$ an integer. Assume that for any s in Θ we have $|H(s)| \leq n/r_0^{i-1}$. In these conditions Θ is a $(1/r_0^{i-1})$-cutting for H. The objective is to refine locally each simplex s so to obtain a $1/r_0^i$-cutting for H. If it happens that $|H(s)| \leq n/r_0^i$, then there is no work to be done, since s satisfies the condition for being an element of a $(1/r_0^i)$-cutting.

So we consider the case when $n/r_0^i \leq H(s) \leq n/r_0^{i-1}$. Computing a sparse net for $(H(s), s)$ directly is time-consuming, therefore, following Chazelle [Cha93], we generate first an approximation $A(s)$ of $H(s)$, then we build a strong net for $(A(s), s)$, whose triangulation is the desired refinement for s. We define $\rho_0 = r_0^i|H(s)|/n$ and $\rho = \rho_0 \log \rho_0$. We can achieve our aim with a sparse net of size ρ. The following lemmas summarize important properties of the construction in [Cha93]:

Lemma 1 ([Cha93]) *Let $A(s)$ be a $(1/(2d\rho_0))$-approximation of $H(s)$ and $R(s)$ a sparse $(1/(2d\rho_0))$-net for $A(s)$ in s, then the following inequality holds:*

$$v(R, s) \leq 4(\rho/|H(s)|)^d v(H, s) + 4\rho^d/\rho_0.$$

Lemma 2 ([Cha93]) *A canonical triangulation of the sparse net $R(s)$ of Lemma 1 has size $O(\rho^{d-1} + v(R, s))$. which is*

Observation: combining the bounds of Lemma 1 and Lemma 2 we obtain a bound on the number of simplices in the refinement of s that is

$$O(\rho_0^{d-1} \log^d \rho_0 + (\frac{\rho^d}{H(s)^d})v(H, s)).$$

Lemma 3 ([Cha93]) *The approximation $A(s)$ and the sparse net $R(s)$ of Lemma 1 are computed in time $O(|H(s)|)$.*

Lemma 4 ([Cha93]) *The number of hyperplanes in H intersecting any simplex in the canonical triangulation of Lemma 2 is less than or equal to n/r_0^i.*

The collection of all the simplices obtained as a result of canonical triangulations of the sparse nets $R(s)$ of Lemma 1 within s, together with those simplices $s \in \Theta$ for which no work was done, for all $s \in \Theta$, is a $(1/r_0^i)$-cutting for H.

3 Incidence and vertical ray-shooting queries

3.1 Preliminary definitions

We fix once and for all a vertical direction in E^d. The following definitions are essential for the algorithm:

Definition 5 *A d-cell in E^d is a d-dimensional simplex in E^d or an infinite prism whose axis is vertical and whose cross section is a $(d-1)$-simplex.*

Definition 6 *A $(d-1)$-simplex t partially covers a d-cell s if t intersects s and the vertical projection of t does not completely contain the vertical projection of s.*

If $(d-1)$-simplex t partially covers a d-cell s then the vertical projection of some $(d-2)$-face of t will intersect the vertical projection of s.

Definition 7 *A $(d-1)$-simplex t completely covers a d-cell s if t intersects s and the projection of t contains the projection of s.*

The concept of partially covering and totally covering $(d-1)$-simplices can be seen as an extension to higher dimensions of the concept of "short" and "long" segment in hereditary segment trees. In this paper, d-cells will be called also *elementary cells*.

3.2 A data structure for incidence queries

We are given a set U of n possibly intersecting $(d-1)$-simplices in E^d. We build a sequence of sets of d-cells $C_0, .., C_l$ where $l = \log_{r_0} n$ and r_0 is a suitable constant. The base of the construction is C_0 and the construction precedes inductively from C_{i-1} to C_i. The set C_i is a collection of d-cells with auxiliary information $(s, P(s), Q(s))$ where s is d-cell in E^d, $P(s)$ a subset of simplices in U *partially covering* s, and $Q(s)$ is a subset of simplices in U *covering* s. In each set C_i the union of the elementary cells is E^d. The invariants maintained over the sets C_i are:

(I) $P(s)$ empty or $|P(s)| \leq n/r_0^i$,

(II) $|Q(s)| \leq r_0 n/r_0^i$.

3.3 The incremental step of the algorithm

The first set is $C_0 = \{(E^d, U, \emptyset)\}$, which satisfies invariants I and II. The algorithm to construct C_k from C_{k-1}, for $k > 0$, works in two main phases.

(1) For any elementary cell s in C_{k-1}, if $|Q(s)| \leq r_0 n / r_0^k$, then we add to C_k the triple $(s, \emptyset, Q(s))$. Otherwise, by induction hypothesis, we have that $r_0 n / r_0^k < |Q(s)| \leq r_0 n / r_0^{k-1}$ and we are in the conditions for the application of Lemma 1. We build a sparse net of Lemma 1 in s on $Q(s)$. We triangulate the sparse net thus obtaining a set of d-cells $(\sigma, \emptyset, Q(\sigma))$, where $Q(\sigma)$ is the subset of elements of $Q(s)$ totally covering σ.

We choose the parameter of the net $\rho_0'(s) = r_0^{k-1}|Q(s)|/n$ and thus obtain, for each new cell σ, $Q|(\sigma)| \leq r_0 n / r_0^k$ as follows from Lemma 4. Note that, for every s in C_{k-1} we have $\rho_0'(s) \leq r_0$, which is an important property exploited in the analysis.

(2) For any s in C_{k-1}, if $P(s)$ is empty we skip phase (2). If $|P(s)| \leq n/r_0^k$, then we add to C_k the triple $(s, P(s), \emptyset)$. Otherwise, by induction hypothesis we have $n/r_0^k < |P(s)| < n/r_0^{k-1}$. We project vertically the (d-2)-faces of simplices in $P(s)$ onto a $(d-1)$-dimensional subspace, and extend the projections into full hyperplanes in this subspace. We denote this set of hyperplanes in $(d-1)$-space with $P'(s)$. We project vertically s and decompose the projection into (a constant number of) $(d-1)$-simplices. For any such region s', we make a sparse net of Lemma 1 for the hyperplanes $P'(s)$ in s'. We triangulate the sparse net and obtain a set of $(d-1)$-cells. Extend the $(d-1)$-cells vertically in d-space within s obtaining infinite prisms $(\eta, P(\eta), Q(\eta))$, where $P(\eta)$ is the subset of simplices in $P(s)$ partially covering η, and $Q(\eta)$ is the subset of simplices in $P(s)$ totally covering η.

We choose the parameter of the nets $\rho_0''(s) = r_0^k|P(s)|/n$ so as to obtain $|P(\eta)| \leq n/r_0^k$ from lemma 4 and $|Q(\eta)| \leq |P(s)| \leq r_0 n/r_0^k$ from the induction hypothesis. Note that, for every s in C_{k-1} we have $\rho_0''(s) \leq r_0$.

The collection of all the elementary cells generated in the two phases is C_k. The elementary cells in C_k satisfy invariants I and II. The d-cells produced in the two phases are organized in a two-level search tree. The search trees on d-cells built in phase (1) (resp. (2)) will be called Q-trees (resp. P-trees). The search tree built on the sequence $C_0, ...C_l$ is used as sketched in the introduction to test whether a query point q is incident to any simplex in U. The main issues are the time spent in the construction and the size of the search tree.

3.4 Analysis of the algorithm

To simplify the notation we set $\rho_0(s) = \max\{\rho_0'(s)\rho_0''(s)\}$. We denote with K the total number of vertices in the arrangement of the set of simplices U which are incident to at least d of the $(d-1)$ simplices in U; clearly we have $K \leq \binom{n}{d}$. We denote with $c_1, c_2, ...$ absolute multiplicative constants which may depend on d. We obtain a bound on $|C_k|$ by summing the number of new d-cells generated in one iteration of the algorithm for all the d-cells in C_{k-1}. We use Lemma 2 to bound the number of d-cells obtained at each iteration of the algorithm.

$$|C_k| \leq \sum_{s \in C_{k-1}} c_1[\rho_0^{d-1}(s) \log^{d-1} \rho_0(s) + v(R(s), s)] + \sum_{s \in C_{k-1}} c_2[\rho_0^{d-2}(s) \log^{d-2} \rho_0(s) + v(R'(s), s')] \quad (1)$$

We apply the observation after Lemma 2, the observation that always $\rho(s) \leq r_0$ and the induction hypothesis on the size of $Q(s)$ and $P(s)$, thus obtaining a new expression bounding $|C_k|$:

$$\sum_{s \in C_{k-1}} c_3[r_0^{d-1} \log^d r_0 + (r_0^k \log r_0/n)^d v(Q(s), s)] + \sum_{s \in C_{k-1}} c_4[r_0^{d-2} \log^{d-1} r_0 + (r_0^k \log r_0/n)^{d-1} v(P'(s), s')]. \quad (2)$$

We rearrange the summations and subsume smaller terms under larger ones:

$$\sum_{s \in C_{k-1}} c_5[r_0^{d-1} \log^d r_0 + (r_0^k \log r_0/n)^d v(Q(s), s) + (r_0^k \log r_0/n)^{d-1} v(P'(s), s'))]. \quad (3)$$

Finally, noticing that $\sum_{s \in C_{k-1}} v(P'(s), s') = O(n^{d-1})$, we obtain the following recursive inequality in the variable k:

$$|C_k| \leq c_6[r_0^{d-1}(\log^d r_0)|C_{k-1}| + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)]. \quad (4)$$

In the next lemma we solve the recursion 4 and we find a bound on $|C_k|$ in terms of n and K.

Lemma 5 *$|C_k| \leq D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}$, where F and D are constants with respect to k, n and K.*

Proof. Let us define a value $\tilde{k}$ such that:

$$(K/\binom{n}{d}) r_0^{kd} = r_0^{k(d-1+\epsilon)} \quad (5)$$

That is:

$$\tilde{k} = (1/(1-\epsilon)) \log_{r_0}(\binom{n}{d}/K)).$$

Let $\bar{k} = \lfloor \tilde{k} \rfloor$. Note that for $k \leq \bar{k}$

$$(K/\binom{n}{d}) r_0^{kd} \leq r_0^{k(d-1+\epsilon)} \quad (6)$$

while for $k > \bar{k}$

$$(K/\binom{n}{d}) r_0^{kd} \geq r_0^{k(d-1+\epsilon)}. \quad (7)$$

We will show that for $k \leq \bar{k}$ equation 4 has solution $|C_k| \leq D r_0^{k(d-1+\epsilon)}$ for a constant D. We will show that for $k > \bar{k}$ equation 4 has solution $|C_k| \leq F(K/n^d) r_0^{kd}$ for a constant F. Notice

that the equation 4 is a recursive equation on a function of k, thus the term K/n^d is treated like a constant during the solution of the recurrence.
Case (1). $0 \le k \le \bar{k}$. We prove the bound $|C_k| \le D r_0^{k(d-1+\epsilon)}$ by induction. For $k=0$ the bound is trivially true since $C_0 = (E^d, U, \emptyset)$ and $|C_0| = 1$. Otherwise we inductively assume the bound holds for C_{k-1}:

$$\begin{aligned} |C_k| &\le c_6[r_0^{d-1+\epsilon}(\log^d r_0) D(1/r_0^{\epsilon}) r_0^{(k-1)(d-1+\epsilon)} + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)] \\ &\le c_6[(\log^d r_0) D(1/r_0^{\epsilon}) r_0^{k(d-1+\epsilon)} + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)]. \end{aligned}$$

Since $k \le \bar{k}$ we have $(K/n^d) r_0^{kd} = O(r_0^{k(d-1+\epsilon)})$, thus:

$$|C_k| \le c_7[(\log^d r_0) D(1/r_0^{\epsilon}) + (\log^d r_0) + (\log^{d-1} r_0)] r_0^{k(d-1+\epsilon)}. \tag{8}$$

Choosing r_0 in dependence on ϵ, and D large enough we can make sure that:

$$c_7[(\log^d r_0) D(1/r_0^{\epsilon}) + (\log^d r_0) + (\log^{d-1} r_0)] \le D$$

and therefore we prove the bound.
Case (2). $k > \bar{k}$. We prove by induction a bound $|C_k| \le F(K/n^d) r_0^{kd}$. The base case is for $k = \bar{k}+1$. By using the bound of case (1) for $C_{\bar{k}}$ within equation 4 we obtain easily a bound:

$$|C_{\bar{k}+1}| \le c_6[(\log^d r_0) D r_0^{d-1} r_0^{\bar{k}(d-1+\epsilon)} + r_0^{\bar{k}d}(\log^d r_0)(K/n^d) + r_0^{\bar{k}(d-1)}(\log^{d-1} r_0)] \le F(K/n^d) r_0^{d(\bar{k}+1)}$$

for a constant F depending on r_0,D and ϵ. Let us assume the bound holds inductively:

$$\begin{aligned} |C_k| &\le c_6[r_0^{d-1}(\log^d r_0)|C_{k-1}| + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)] \\ &\le c_6[r_0^{d}(\log^d r_0) F(1/r_0)(K/n^d) r_0^{(k-1)d} + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)]. \end{aligned}$$

Since $k > \bar{k}$ inequality 7 holds and we have $r_0^{k(d-1)} < r_0^{k(d-1+\epsilon)} = O((K/n^d) r_0^{kd})$, therefore:

$$|C_k| \le c_8[(\log^d r_0) F(1/r_0) + (\log^d r_0) + (\log^{d-1} r_0)](K/n^d) r_0^{kd} \tag{9}$$

Choosing F and r_0 large enough we have:

$$c_8[(\log^d r_0) F(1/r_0) + (\log^d r_0) + (\log^{d-1} r_0)] \le F$$

and the bound is proved. ∎

3.4.1 Storage and preprocessing time bounds

We continue the construction until k reaches the value $l = \log_{r_0} n$. The total number of elementary cells is:

$$\sum_{k=0}^{l} |C_k| \leq \sum_{k=0}^{l} [Dr_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}] \tag{10}$$

This is a summation of geometric series of ratio $r_0^{d-1+\epsilon}$ and r_0^d, which is proportional to the last term of the series. Thus we have a bound $O(Dr_0^{l(d-1+\epsilon)} + 2F(K/n^d)r_0^{ld}) = O(n^{d-1+\epsilon} + K)$ on the size of the search tree. The time spent on each d-cell in the search tree is linear in the number of $(d-1)$-simplices intersecting it by Lemma 3. So we have a bound:

$$\sum_{k=0}^{l} (nr_0/r_0^k)|C_k| \leq \sum_{k=0}^{l} (nr_0/r_0^k)[Dr_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}] \tag{11}$$

By simple manipulations and using the previous observation on the sum of geometric series we obtain an upper bound $O((nr_0)[Dr_0^{l(d-2+\epsilon)} + F(K/n^d)r_0^{l(d-1)}])$ which is $O(n^{d-1+\epsilon} + K)$.

3.4.2 Query time for incidence queries

We organize the sets $C_0, C_1, .., C_l$ as levels of a two-layer search tree where the P-tree is the primary structure and the Q-tree is the secondary one. Given the query point q we search exhaustively an elementary cell s at the root of our search data structure which contains q in its closure. Then we continue the search recursively in the P-tree and the Q-tree rooted at s. On the Q-trees the search visits a single path of logarithmic depth. Therefore we have a total cost $O(\log n)$ for searching a Q-tree. The query on a P-tree is solved recursively. If $T(h)$ is the time needed to solve the query starting from a node at height h, we have that:

$$\begin{aligned} T(h) &\leq O(\log n) + T(h-1) \\ T(1) &= O(1) \end{aligned}$$

Thus $T(\log n) = O(\log^2 n)$.

Since the total number of cells of any dimension in the arrangement $\mathcal{A}(U)$ is $O(n^{d-1} + K)$ we have the following theorem:

Theorem 1 *Given a set U of n possibly intersecting $(d-1)$-simplices in d-space we can build a data structure of size $O(n^{d-1+\epsilon} + K)$ deterministically in time $O(n^{d-1+\epsilon} + K)$, where K is the complexity of the arrangement $\mathcal{A}(U)$. Afterwards we can determine in time $O(\log^2 n)$ whether a query point is incident to a simplex in U.*

3.5 Vertical ray-shooting in arrangement of simplices

In order to solve the vertical ray-shooting problem we add auxiliary data structures to the search tree built previously. Let us discuss Q-trees first. We consider cells $s \in C_{k-1}$ and $\sigma \in C_k$ such that $\sigma \subset s$ and σ is obtained by phase (2) of the algorithm. We determine during preprocessing the subset of $(d-1)$-simplices in $Q(s)/Q(\sigma)$ which lies below σ. We can extend the simplices of this subset into full hyperplanes and build a data structure for general ray-shooting in convex polytopes of size roughly $O(|Q(s)|^{\lfloor d/2 \rfloor})$ [Sch92][2]. This data structure is associated with σ.

Let us consider now the P-trees. We consider cells $s \in C_{k-1}$ and $\sigma \in C_k$ such that $\sigma \subset s$ and σ is obtained by phase (1) of the algorithm. Note that both s and σ in this case are vertical infinite cylinders. From this it follows that if a query point q is in σ then the simplex in $P(s)$ immediately below q is either the simplex in $P(\sigma)$ immediately below q or the simplex in $Q(\sigma)$ immediately below q. Vertical ray shooting queries are solved by recursive calls on P-trees and by using the auxiliary data structures on Q-trees. On Q-trees the search path visits $O(\log n)$ nodes and at each node $O(\log n)$ time is spent querying the data structure in [Sch92].

The query time for the recursion on the Q-trees rooted at a simplex s has an additional logarithmic factor thus we obtain a time $O(\log^2 n)$ for one partial query. The query time satisfies the same equation as before with an additional $O(\log^2 n)$ term due to the queries on the auxiliary data structures. If $T(h)$ is the time needed to solve the query starting from a node at height h, we have that:

$$\begin{aligned} T(h) &\leq O(\log^2 n) + T(h-1) \\ T(1) &= O(1) \end{aligned}$$

Thus $T(\log n) = O(\log^3 n)$. The time needed to construct the data structure is bounded by:

$$\begin{aligned} \sum_{k=0}^{l} (nr_0^2/r_0^k)^{\lfloor d/2 \rfloor} |C_k| &\leq \sum_{k=0}^{l} (nr_0^2/r_0^k)^{\lfloor d/2 \rfloor} [D r_0^{k(d-1+\epsilon)} + F(K/n^d) r_0^{kd}] \\ &\leq 2(nr_0^2)^{\lfloor d/2 \rfloor} [D r_0^{l(d-1-\lfloor d/2 \rfloor+\epsilon)} + F(K/n^d) r_0^{l(d-\lfloor d/2 \rfloor)}] \\ &\leq 2r_0^d (Dn^{d-1+\epsilon} + FK) = O(n^{d-1+\epsilon} + K) \end{aligned}$$

The above discussion proves the following theorem:

Theorem 2 *Given a set U of n possibly intersecting $(d-1)$-simplices in d-space we can build a data structure of size $O(n^{d-1+\epsilon} + K)$ deterministically in time $O(n^{d-1+\epsilon} + K)$, where K is the complexity of the arrangement $\mathcal{A}(U)$. Afterwards we can solve vertical ray-shooting queries in time $O(\log^3 n)$*

[2] We could use the method in [AM92] but doing so we would add an extra logarithmic factor to the query time

4 Point Location among simplices

In this section we tackle the general point location problem by using the solution to the vertical ray-shooting problem as a subroutine. We encode the facial structure of $\mathcal{A}(U)$ in a sequence of sets of graphs. The final graph will have nodes for each face of $\mathcal{A}(U)$ and each node will have a label indicating the d-dimensional cell having this face at its boundary. For a query point p we retrieve at query time the labels associated with the faces containing it.

In the following discussion we will have to deal with the fact that a face is incident to more than one cell and therefore we must consider the role of the several "sides" of a face. The main idea is to walk along the boundary of cells and describe such boundary as an incidence graph of lower dimensional cells. We use ray-shooting in order to join "islands" of the boundary of a cell that cannot be reached by the simple walking procedure. If we think of this problem on the plane with a set of segments, it is clear that we can describe the boundary of a cell by walking clockwise on the connected parts of its boundary. What follows is a generalization of this walking and jumping idea to higher dimensional spaces.

4.1 Definitions

First we give additional definitions that lay the groundwork for the main result. The terminology and the concepts used are consistent with those introduced in [AS92].

A set U of n $(d-1)$-simplices in E^d decomposes E^d into open cells of dimension d (also called d-faces) and into relatively open cells of dimension k, for $0 \leq k < d$. Let $\mathcal{A}(U)$ be the collection of cells of any dimension induced by U. We assume that the simplices in U are in general position, meaning that any k of them (for $k = 2, .., d$) intersect, if at all, in a polytope of dimension at most $d-k$. This intersection is called a $(d-k)$-*flap* in [AS92]. We assume that $d+1$ simplices in U do not have a point in common. We assume also that the intersection of any collection of k open faces of dimensions $i_1, ..., i_k$ is either empty or is a polytope of dimension $d - \sum_j (d - i_j)$. A perturbation argument shows that the maximum size of the quantities of interest is attained by a set U in general position. We distinguish *outer k-faces* which are contained in the relative boundary of a simplex in U and *inner k-faces* which are contained exactly in the interior of $d-k$ simplices and avoid their relative boundary. We distinguish between different sides of an (inner or outer) face. Let f be an inner k-face contained in the relative interior of $d-k$ simplices. The hyperplanes spanning these simplices divide space into 2^{d-k} open regions. A *side* of f is a pair (f, r) where r is one of these regions. If f is a d-cell then it has only one side, namely (f, E^d). Notice that a $(d-1)$-face has two sides. Similar definitions hold for the outer k-faces with the difference that outer k-faces are contained in fewer than $d-k$ simplices.

Definition 8 *A side (f, r) is a k-border of a d-cell C if either $f = C$ and $k = d$, or, f is k-face on the boundary of C and some open neighbourhood of f in $r \cup f$ is contained in $C \cup f$.*

Intuitively the concept of k-border captures the fact that f is on the boundary of a d-cell C and that C is on the same side of f as r.

Definition 9 *For $0 \leq k \leq i < d$ a (k,i)-border of C is a pair $((f,R),(g,Q))$ of borders of C of dimensions k and i such that $f \subset \bar{g}$ and $R \subset Q$.*

Let $G(U)$ be a graph whose nodes are k-borders of $\mathcal{A}(U)$, for all values of k. Two nodes (f,R) and (g,Q) are connected by an edge in $G(U)$ if $((f,R),(g,Q))$ is a (k,i)-border. We aim at generating a labelling of the nodes of $G(U)$ such that all the k-borders of a same d-cell C have the same label and this label is different from the label of k-borders of d-cell C', if $C \neq C'$.

4.2 The pre-processing algorithm

The construction of the labelled graph $G(U)$ for U in E^d is done in phases which are denoted with an index $j = 2,..,d$. At the end of phase j we have a set of labelled incidence graphs where each graph corresponds to an arrangement of simplices in dimension j. We use this information to build a set of graphs of arrangements in dimension $j+1$. The output of the last phase for $j = d$ represents our labelled graph $G(U)$. To start our construction let us consider dimension $j = 2$ which is our base case. We compute the collection $\mathcal{B}_2 = \binom{U}{d-2}$ of all of $(d-2)$-subsets of U. Let β be an element of $\mathcal{B}_2$ and $q(\beta)$ be the 2-flap which is the intersection of the simplices in β. If $q(\beta)$ is not empty, we build the planar arrangement of the segments $U_\beta = \{s \cap q(\beta) | s \in U/\beta\}$ on $q(\beta)$. Using the method of [CE92] we can construct explicitly such in arrangement in time $O(n \log n + K_\beta)$, where K_β is the number of vertices in $\mathcal{A}(U_\beta)$. Within the same time we can compute the graph $G(U_\beta)$ and label its k-borders with the name of the bordering cell. Moreover we can add standard planar point location data structures so to be able to locate a 0-border of the 2-cell containing a query point.

We have established the result for $j = 2$, now we assume $j > 2$ and we assume inductively that we have computed the relevant data structure for $(j-1)$. More precisely, we have the labelled graphs $G(U_\beta)$ for all $\beta \in \mathcal{B}_{j-1} = \binom{U}{d-(j-1)}$, and also the point-location/vertical ray-shooting data structures for all sets U_β. We shall use this information to construct the graphs $G(U_\gamma)$, where $\gamma \in \Gamma = \mathcal{B}_j = \binom{U}{d-j}$. Consider a particular $\gamma \in \Gamma$, we aim to construct the incidence graph over the set of simplices $U_\gamma = \{s \cap q(\gamma) | s \in U/\gamma\}$ on $q(\gamma)$. The relative boundary of $q(\gamma)$ is formed by $j-1$ simplices and we can assume inductively that we know the arrangement induced by U on the facets of $q(\gamma)$ and the corresponding graphs incidence G.

An inner k-face f in $\mathcal{A}(U)$ is contained in $d-k$ simplices of U. We can split these $d-k$ simplices into two groups: $d-j$ of them define a flat $q(\gamma)$ to which f belongs; the $j-k$ remaining simplices define the affine span of f within $q(\gamma)$. We denote with $\Gamma(f) = \{\gamma \in \Gamma | f \subset q(\gamma)\}$, the subset of elements of Γ containing f in their corresponding j-flap. Similarly we define $\mathcal{B}(f) = \{\beta \in \mathcal{B}_{j-1} | f \subset q(\beta)\}$. From the previous observation $|\Gamma(f)| = \binom{d-k}{d-j}$ and $|\mathcal{B}(f)| = \binom{d-k}{d-(j-1)}$. Fixing a $\gamma \in \Gamma(f)$ is equivalent to fixing $d-j$ simplices incident to f, thus we can choose any one of the remaining $j-k$ simplices incident to f to define a set β in $\mathcal{B}(f)$. We have therefore the following set:

$$\mathcal{B}(f,\gamma) = \{\beta \in \mathcal{B}(f) | q(\beta) \subset q(\gamma)\}$$

representing the β's which contribute to γ. The cardinality of this set is $|\mathcal{B}(f,\gamma)| = j-k$.

The main idea is to use the regions of the k-borders in graphs $G(\mathcal{B})$ to build the regions of k-borders in graphs in $G(\Gamma)$. First we must establish with a counting argument that indeed we have enough nodes from lower dimensional graphs to carry out the construction. One face f in $\mathcal{A}(U_\gamma)$ corresponds to 2^{j-k} k-borders in $G(U_\gamma)$ and each such k-border has associated a region delimited by $(j-k)$ facets. These facets are regions of k-borders of f in the set of graphs $G(U_\beta)$ for $\beta \in \mathcal{B}(f,\gamma)$. In order to construct $G(U_\gamma)$ for all $\gamma \in \Gamma$ we proceed in the following way: we duplicate every node in every G_β for $\beta \in \mathcal{B}$. This operation reflects the fact that every $(j-1)$-cell has 2 sides in the new space $q(\gamma)$. Consequently every label is split into a positive and a negative label. The total number of k-borders of a face f in graphs $G(U_\beta)$ for $\beta \in \mathcal{B}(f,\gamma)$ becomes:

$$2[2^{j-1-k}(j-k)] = (j-k)[2^{j-k}].$$

Now the algorithm is easy: we build the regions of k-borders of f in $G(U_\gamma)$ by taking the regions of k-borders of f in $G(U_\beta)$ for $\beta \in \mathcal{B}(f,\gamma)$ as *facets*. We associate to the nodes in $G(U_\gamma)$ an initial labelling using the $(j-k)$ labels of the facets of the region of each k-border in $G(U_\gamma)$. We recall that the labels we are using are just names of $(j-1)$-cells. At this stage we have an initial labelled graph $G^i(U_\gamma)$ with a labelling consistent with the information we obtain from lower dimensional graphs. On the other hand this initial labelling is such that a single cell in $\mathcal{A}(U_\gamma)$ does not have yet a unique identifier. The next phase of the algorithm finds a unique identifier for every j-cell of $\mathcal{A}(U_\gamma)$ and updates correspondingly the labels of the k-borders in $G(U_\gamma)$. Let C be a j-cell in $\mathcal{A}(U_\gamma)$.

1. **Relabelling Step I**. We build a label graph $LG(\gamma)$ whose nodes are nodes of $G(U_\gamma)$. We place an edge between two nodes of $LG(\gamma)$ if and only if the two initial labels of the nodes have not empty intersection. Then we run a standard connected component algorithm on $LG(\gamma)$ [AHU74]. During the visit of the graph $LG(\gamma)$ we also compute for each connected component of the graph the lowest point in the vertical direction of the space $q(\gamma)$. At the end of this phase every path-connected component of the boundary of C has a unique label.

2. **Relabelling Step II**. Now we proceed to identify the several connected components of the boundary of a j-cell C on $q(\gamma)$. We build in $q(\gamma)$ for U_γ the vertical ray-shooting data structure described in Section 3. We build a a graph of boundary components $BCG(\gamma)$ whose nodes are the connected components of the graph $LG(\gamma)$. We connect with an arc two nodes u and u' in $BCG(\gamma)$ if, shooting in $q(\gamma)$ from the lowest point of u, we reach a point on u'.

   Afterwards we find the connected components of $BCG(\gamma)$ using standard graph-searching algorithms. The next lemma states that the final labelling obtained from the connected components of the graph $BCG(\gamma)$ satisfies the uniqueness property.

Lemma 6 *The identifiers of the connected components of $BCG(\gamma)$ are unique labels for the j-cells in $\mathcal{A}(U_\gamma)$.*

Proof. In order to simplify the argument we assume that all the simplices in U are bounded and we consider a solid compact d-ball B containing U in its interior. Let C be a non-empty j-cell of $\mathcal{A}(U_\gamma) \cap B$ and $B_1, ..., B_k$ the connected components of the boundary of C. Cell C is bounded and in particular at least one boundary component is a regular boundary as defined in [Ale56, page 41] i.e. a set which separates E^d into two domains. Moreover, one of these two domains is bounded and the other is unbounded. The procedure for linking nodes in the graph $BCG(\gamma)$ produces a partial order among the nodes $B_1, .., B_k$. We show that there cannot be two bottom elements in this partial order.

If B_i is not a regular boundary, then the vertical ray from the lowest point of B_i is within C and will intersect another boundary component. Therefore B_i cannot be a bottom element of the partial order. Let us assume by contradiction that we have two regular boundaries B_1 and B_2, with associated bounded domains D_1 and D_2, such that B_1 and B_2 are both bottom elements of the partial order. If this is the case then the vertical *downward* ray from the lowest point of B_1 (resp. B_2) is in the unbounded domain defined by B_1 (resp. B_2). Therefore, we have that $C \subseteq D_1$ and $C \subseteq D_2$. Now we have two main cases to consider since the other cases lead easily to contradictions. Case 1. $D_2 \subset D_1$, from which $C \subseteq D_2 \subset D_1$. But then B_1 cannot be on the boundary of C unless it is a subset of B_2. Since B_1 and B_2 are disjoint we have a contradiction. Case 2. $D_2 \subset E^d \setminus D_1$. In this case $D_1 \cap D_2 = \emptyset$ and therefore $C = \emptyset$, which is a contradiction. From the above discussion it follows that all the boundary components of a cell C are included in the same component of the graph $BCG(\gamma)$.

Now we assume that B_1 is a boundary component of C_1 and B_2 is a boundary component of C_2 and (B_1, B_2) is an arc in $BCG(U_\gamma)$. If so is the case there is a vertical segment which connects B_1 and B_2 which does not intersect any simplex in U and which at the extremes belongs to C_1 and C_2. It follows that C_1 and C_2 are path connected and therefore they are the same j-cell of $\mathcal{A}(U_\gamma)$. We have proved that if the nodes of $BCG(\gamma)$ are the boundary connected components of cells in $\mathcal{A}(U_\gamma)$ then the connected components of $BCG(\gamma)$ are in 1-1 correspondence with j-cells of $\mathcal{A}(U_\gamma)$.

Now, let p_1 and p_2 be two points on a connected component B of the boundary of a j-cell C. Then there is a continuous path contained in B and in C that has p_1 and p_2 as extremes. We can reduce this path so that it visits each $(j-1)$-face of B at most once. Let $F_1, ..., F_k$ be the sequence of $(j-1)$-faces of a path connecting p_1 and p_2, and let f_i be the $(j-2)$-face shared by F_i and F_{i+1}. Assuming by induction on the dimension that we have a unique label for the faces F_i, we have that the initial labelling of the $(j-2)$-border f_i of C is (F_i, F_{i+1}). Therefore if we build the graph $LG(\gamma)$ there will be a path connecting F_1 and F_k in $LG(\gamma)$.

On the other hand, if p_1 on face F_1 and p_2 on face F_2 belong to different boundary components B_1 and B_2, then any path between B_1 and B_2 will have at least on point which does not belong to any $(j-1)$-border in the graphs $G(\mathcal{B})$. Therefore the faces F_1 and F_2 are not connected by any path in $LG(\gamma)$. ∎

Once we have defined the labels, these are propagated from the graph $BCG(\gamma)$ to the graph $LG(\gamma)$ and to $G(\gamma)$ and recursively to the lower-dimensional graphs which contribute to $G(\gamma)$. We stop the construction when $j = d$ and the final object is a single graph $G(U)$ with a proper labelling together with a vertical ray-shooting data structure for U.

4.3 Point location procedure

Using the data structure $VRS(U)$ we can find for a query point p the simplex $s \in U$ immediately below p and also the landing point p' on s. We repeat the vertical query procedure on s. Eventually we reach a vertex of $\mathcal{A}(U)$ whose label is the (unique) label of the d-dimensional cell containing p.

4.4 Time and storage analysis

The number of sets in $\mathcal{B}_j$ is $O(n^{d-j})$. For $j = 2$ which is the base case, we spend time $O(n \log n + K_\beta)$ for each set β. So the total cost of the base step is:

$$\sum_{\beta \in \mathcal{B}_2} O(n \log n + K_\beta) = O(n^{d-1} \log n + K).$$

The graphs $G(\mathcal{B}_j)$ for $j > 2$ have total size:

$$\sum_{\beta \in \mathcal{B}_j} O(n^{j-1} + K_\beta) = O(n^{d-1} + K).$$

We keep a dictionary of faces such that for each face we can retrieve efficiently the corresponding borders and the graphs $G(\mathcal{B}_j)$ having these borders as nodes. We can use standard data structure with logarithmic cost per operation. So we bound the time to construct $G(\Gamma)$ with $O(n^{d-1} \log n + K \log n)$. The additional vertical ray-shooting data structures are set up at cost:

$$\sum_{\beta \in \mathcal{B}_j} O(n^{j-1+\epsilon} + K_\beta) = O(n^{d-1+\epsilon} + K).$$

The labelling steps I and II are based on Depth First Search [AHU74] and take overall time proportional to the size of $G(\mathcal{B})$. The ray shooting operations in labelling step II take $O(\log^3 n)$. The number of vertical ray-shooting queries is $O(n)$ for each $\beta \in \mathcal{B}_j$, because we have always at most n components. So the total cost is $O(n^{d-j+1} \log^3 n) = O(n^{d-1} \log^3 n)$. The graphs $BCG(\beta)$ has only $O(n)$ nodes, therefore finding the components takes total time $O(n^{d-j+1}) = O(n^{d-1})$ [AHU74]. The correctness of the algorithm has been established in the previous section. The analysis of the storage and preprocessing time gives us the theorem:

Theorem 3 *Given a set U of n simplices in E^d, let K be the number of cells of any dimension in $\mathcal{A}(U)$. For every $\epsilon > 0$, it is possible to build a point location data structure of size $O(n^{d-1+\epsilon} + K)$ in time $O(n^{d-1+\epsilon} + K \log n)$, where the constant of proportionality depend on ϵ, such that a unique identifier for the cell containing a query point p can be found in time $O(\log^3 n)$.*

We obtain the following corollaries which exploit the point-location data structure to solve motion planning problems.

Corollary 1 *Using the data structure of Theorem 3, given two points p_1 and p_2 we can determine in time $O(\log^3 n)$ whether there is a collision-free path from p_1 to p_2.*

Since we can store spanning trees of the graphs built in the several phases of the algorithms we can find an explicit representation of a path connecting two query points. The length of the path is $O(n^{d-1} \log n)$ as follows from the bound in [AS92].

5 A method for building triangulations

In this algorithm a by a d-cell we indicate a truncated vertical prism whose cross section is a $(d-1)$-simplex. We are given a set U of n pairwise interior-disjoint $(d-1)$-simplices in E^d. The construction of the triangulation $T(U)$ of $E^d \setminus U$ proceeds in stages. We build a sequence of sets $C_0, .., C_l$ where $l = \log_{r_0} n$ and r_0 is a suitable constant. The basis of the construction is C_0 and we proceed inductively from C_{k-1} to build C_k. The set C_k is a collection of triples $(s, P(s), Q(s))$ where s is a d-cell, $P(s)$ is the subset of $(d-1)$-simplices in U *partially covering* s, and $Q(s)$ is the subset of $(d-1)$-simplices in U *fully covering* s. If $s \in C_k$ and we will have associated sets $P(s)$ and $Q(s)$ with the invariant properties that

(I) $|P(s)| \leq n/{r_0}^k$ (first invariant).

(II) $|Q(s)| \leq nr_0/{r_0}^k$ (second invariant).

C_k is a refinement of C_{k-1} and moreover the d-cells in C_k *partition* E^d. At the last stage C_l contains only d-cells intersecting a constant number of original simplices in U, thus we obtain a triangulation $T(U)$ of size $O(|C_l|)$ by a final greedy decomposition. The first set is $C_0 = \{(E^d, U, \emptyset)\}$, which satisfies invariants (I) and (II). We assume that the two invariants hold for C_{k-1}, with $k > 0$ and we show how to construct C_k.

5.1 The algorithm

Let s be an elementary cell in C_{k-1}, since the the first and second invariant hold by inductive hypothesis, s is partially covered by at most $n/{r_0}^{k-1}$ $(d-1)$-simplices and is fully covered by at most $nr_0/{r_0}^{k-1}$ $(d-1)$-simplices. The covering $(d-1)$-simplices $Q(s)$ are disjoint and therefore they are linearly ordered in the vertical direction within s. We split $Q(s)$ into at most r_0 groups of $\lceil Q(s)/r_0 \rceil$ simplices each by selecting every $\lceil Q(s)/r_0 \rceil$-th $(d-1)$-simplex in the vertical ordering. The selected totally covering $(d-1)$-simplices slice s into elementary cells numbered $i = 1, .., r_0$, which we denote $\sigma(s,i)$ (or σ_i whenever s is clear from the context). We also denote with $\sigma'(s,i)$ the vertical projection of $\sigma(s,i)$.

From the above construction each σ_i is covered by $|Q(\sigma_i)| \leq nr_0/{r_0}^k$ $(d-1)$-simplices. Each σ_i is partially intersected by $|P(\sigma_i)| = q_i$ $(d-1)$-simplices, where $\sum_i q_i = |P(s)| \leq n/{r_0}^{k-1}$. This property is easily proved since a partially covering $(d-1)$-simplex of $P(s)$ can intersect to only one elementary cell σ_i.

The average σ_i has a partial cover of size $|P(s)|/r_0$ which is exactly what is needed to have a valid element of the set C_k. To make this averaging argument work in the worst case we proceed as follows for each elementary cell σ_i. If $P(\sigma_i) < n/r_0^k$ we do nothing. Otherwise, we project, independently for each cell σ_i, all the $(d-2)$-boundaries of the partially covering simplices of $P(\sigma_i)$ onto a $(d-1)$-subspace. We extend these sets into full hyperplanes, obtaining a set $P'(s,i)$ of q_i hyperplanes in $(d-1)$-space. Since $n/r^k \leq |P(\sigma_i)| \leq n/r^{k-1}$ we have the conditions for using Lemma 1. We build the sparse net of Lemma 1 in $(d-1)$-space and its triangulation. So we generate elementary $(d-1)$-dimensional cells which are $(d-1)$-simplices. Then each such cell is extended in the vertical direction and intersected with σ_i, thus obtaining a into truncated d-prism.

We choose as parameter of the sparse net construction a number ρ_{0i} such that $q_i/\rho_{0i} = |P(s)|/r_0$, and thus $\rho_{0i} = q_i r_0/|P(s)|$. The interesting property we use in the analysis is that the sum of the ρ_{0i}'s is less than r_0:

$$\sum_i \rho_{0i} = \sum_i q_i r_0/|P(s)| = r_0/|P(s)| \sum_i q_i \leq r_0|P(s)|/|P(s)| = r_0.$$

Let η be a cell so obtained. We have that the number of $(d-1)$-simplices partially covering η is $|P(\eta)| \leq q_i/\rho_{0i} = P(s)/r_0 = n/{r_0}^k$ (Lemma 4), thus η satisfies the first invariant of C_k. The number of $(d-1)$-simplices in $Q(\eta)$ is at most $|Q(s)|/r_0 + |P(s)|$ thus is at most $n/{r_0}^{k-1} + n/{r_0}^{k-1} = 2nr_0/{r_0}^k$.

We take η and use the median simplex in the vertical ordering of $Q(\eta)$ to split η into two cells η_1 and η_2 which satisfy both invariants for C_k. Collecting all of these cells we have C_k which is a partition of E^d. This is the end of the algorithm that builds C_k starting from C_{k-1}.

5.2 Analysis of the algorithm

We now derive a recursive equation linking the size of C_k with the size of C_{k-1} by summing the number of new cells produced by the algorithm for each cell s in C_{k-1}. Let $A_i(s)$ be the number of d-dimensional cells generated within $\sigma(s,i)$ in one phase of the algorithm. Using the bound of Lemma 2, the number of cells obtained is:

$$A_i(s) \leq c_9[\rho_i^{d-2} + (\rho_i/q_i)^{d-1} v(P'(s,i), \sigma'(s,i)) + \rho_i^{d-1}/\rho_{0i}].$$

where $\rho_i = \rho_{0i} \log \rho_{0i}$. Thus the total number of cells in C_k is bounded by:

$$|C_k| \leq 2 \sum_{s \in C_{k-1}} \sum_{\sigma(s,i)} A_i(s) = \sum_{s \in C_{k-1}} \sum_{\sigma(s,i)} c_{10}[\rho_i^{d-2} + (\rho_i/q_i)^{d-1} v(P'(s,i), \sigma'(s,i)) + \rho_i^{d-1}/\rho_{0i}].$$

Bounding terms of the summation separately we have:

$$\sum_i \rho_i^{d-2} \leq (\sum_i \rho_{0i} \log \rho_{0i})^{d-2} \leq {r_0}^{d-2} \log^{d-2} r_0$$

and

$$\sum_i \rho_i^{d-1}/\rho_{i0} \leq ((\sum_i \rho_{0i})^{d-2})(\log^{d-1}(\sum_i \rho_{0i})) \leq {r_0}^{d-2} \log^{d-1} r_0.$$

Since $\rho_i/q_i \leq {r_0}^k (\log r_0)/n$ we obtain a term

$$\sum_s \sum_i ({r_0}^{k(d-1)} (\log r_0)^{d-1}/n^{d-1}) v(P'(s,i), \sigma'(s,i)).$$

Since $\sum_s \sum_i v(P'(s,i), \sigma'(s,i)) = O(n^{d-1})$ we obtain the following recursive inequality:

$$|C_k| \leq c_{11} (\log r_0)^{d-1} {r_0}^{k(d-1)} + c_{11} (\log r_0)^{d-1} {r_0}^{d-2} |C_{k-1}|,$$

for a constant c_{11} which is independent of r_0. In the next lemma we solve the recurrence obtaining a bound for $|C_k|$ as a function of k.

Lemma 7 $|C_k| \leq D{r_0}^{k(d-1)}$ *for a constant* D*, independent of* k*.*

Proof. We use an induction on k. For $k = 0$, $C_0 = (E^d, U, \emptyset)$ so $|C_0| \leq D$,for $D \geq 1$. Inductively we assume the bound on $|C_{k-1}|$.

$$\begin{aligned} |C_k| &\leq c_{11}(\log r_0)^{d-1}{r_0}^{k(d-1)} + c_{11}D(\log r_0)^{d-1}{r_0}^{d-2}{r_0}^{(k-1)(d-1)} \leq \\ &\leq c_{11}(\log r_0)^{d-1}{r_0}^{k(d-1)} + (c_{11}D(\log r_0)^{d-1}/r_0){r_0}^{k(d-1)} \end{aligned}$$

Choosing r_0 and D large enough we can make sure that $c_{11}(\log r_0)^{d-1} + c_{11}D(\log r_0)^{d-1}/r_0 \leq D$ and thus the bound is proved. ∎

The total number of cells in the sequence of sets C_k is a summation of a geometric sequence of ratio $r_0^{d-1} > 1$ so its value is proportional to the last term, which is $O(n^{d-1})$.

$$\sum_{k=0}^{l} |C_k| \leq \sum_{k=0}^{l} D r_0^{k(d-1)} \leq 2D{r_0}^{l(d-1)} = O(n^{d-1}).$$

What is the time to compute all of the C_i's? The sparse nets are computed in time linear in the number of simplices partially covering the elementary cells by Lemma 3. The slicing hyperplanes are found by repeated applications of a selection algorithm in time $O(r_0|Q(s)|)$ for any elementary cell s. Thus we have running time:

$$\sum_{k=0}^{l}((n/{r_0}^k) + (n{r_0}^2/{r_0}^k))|C_k| \leq 2(1+{r_0}^2)Dn{r_0}^{l(d-2)} = O(nn^{d-2}) = O(n^{d-1})$$

The above discussion proves the following theorem:

Theorem 4 *Give a set* U *of* n *pairwise interior-disjoint* $(d-1)$*-simplices in* d*-space with* $d \geq 3$ *we can build deterministically a triangulation* $T(U)$ *of size* $O(n^{d-1})$ *in time* $O(n^{d-1})$*.*

Following the tree structure of the triangulation we can locate the cells in $T(U)$ containing a query point p in time $O(\log n)$.

Corollary 2 *We can find in time* $O(\log n)$ *the cells of* $T(U)$ *containing a query point* p *and the simplex in* U *immediately below* p*.*

6 Conclusions

In this paper we have presented algorithms for solving point location and motion planning queries in arrangements of simplices in E^d, for any fixed $d \geq 2$. The preprocessing time and storage bounds depend explicitly on the complexity of the arrangement induced by the simplices. This feature is important in order to exploit the "sparsity" of arrangements which are obtained, for example, in motion planning and visibility applications. For sets of interior-disjoint simplices in E^d, with $d \geq 3$, we improve bounds for building triangulations and we obtain a fast point location methods as well.

7 Acknowledgments

I wish to thank Bruno Codenotti and Nina Amenta for their observations on an early draft and the referees for their useful comments.

References

[AHU74] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, 1974.

[Ale56] P.S. Alexandrov. *Combinatorial Topology*. Graylock Press, Rochester N.Y., 1956.

[AM92] 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.

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

[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.

[Ber93] M. Bern. Compatible tetrahedralizations. In *Proceedings of the 9th ACM Symposium on Computational Geometry*, pages 281–288, 1993.

[Can87] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1987.

[CE92] B. Chazelle and H. Edelsbrunner. An optimal algorithm for intersecting line segments in the plane. *Journal of the ACM*, 39(1):1–54, 1992.

[CEGS89] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. Report UIUCDCS-R-90-1578, Dept. Comput. Sci., Univ. Illinois, Urbana, IL, 1989.

[CF92] B. Chazelle and J. Friedman. Point location among hyperplanes and vertical ray-shooting. *Computational Geometry: Theory and Applications*, 1992. To appear.

[Cha84] B. Chazelle. Convex partitions of polyhedra: a lower bound and worst-case optimal algorithm. *SIAM J. on Computing*, 13(3):488–507, 1984.

[Cha87] B. Chazelle. Approximation and decomposition of shapes. In *Advances in Robotics, Vol. 1: Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, 1987.

[Cha93] B. Chazelle. Cutting hyperplanes for divide and conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

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

[Col86] R. Cole. Searching and storing similar lists. *Journal of Algorithms*, (7):202–220, 1986.

[dBGH94] M. de Berg, L. Guibas, and D. Haperin. Vertical decompositions for triangles in 3-space. In *Proceedings of the 10h ACM Symposium on Computational Geometry*, 1994.

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

[EGS86] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, (15):317–339, 1986.

[GT91] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proc. 23rd Annu. ACM Sympos. Theory Comput.*, pages 523–533, 1991.

[Kir83] D. Kirkpatrick. Optimal search in planar subdivision. *SIAM Journal of Computing*, (12):28–35, 1983.

[Lat91] J.-C. Latombe. *Robot Motion Planning.* Kluwer Academic Publisher, 1991.

[LP77] D. Lee and F. Preparata. Location of a point in a planar subdivision and its applications. *SIAM Journal of Computing*, (6):594–606, 1977.

[LT80] R. Lipton and R.E. Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, (9):614–627, 1980.

[Mat91] J. Matoušek. Cutting hyperplane arrangements. *Discrete Comput. Geom.*, 6:385–406, 1991.

[Mat92] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 276–285, 1992.

[Mat93] J. Matoušek. On vertical ray shooting in arrangements. *Computational Geometry: Theory and Applications*, 2:279–285, 1993.

[Meh84] K. Mehlhorn. *Multidimensional Searching and Computational Geometry*, chapter 2. Springer Verlag, 1984.

[Mul91] K. Mulmuley. Hidden surface removal with respect to a moving view point. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 512–522, 1991.

[Pre81] F. Preparata. A new approach to planar point location. *SIAM Journal of Computing*, (10):473–482, 1981.

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

[PT88] F. Preparata and R. Tamassia. Fully dynamic techniques for point location and transitive closure in planar structures. In *Proceedings of the 29th IEEE Symposium on Foundations of Computer Science*, pages 558–567, 1988.

[PT89] F. Preparata and R. Tamassia. Efficient spatial point location. In *Proceedings of the 1989 Workshop on Algorithms and Data Structures*, pages 3–11, 1989.

[PY90] M.S. Paterson and F.F. Yao. Efficient binary space partitions for hidden surface removal and solid modeling. *Discrete & Computational Geometry*, 5:485–503, 1990.

[Sch92] O. Schwarzkopf. Ray-shooting in convex polytopes. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 286–295, 1992.

[SS83] J.T. Schwartz and M. Sharir. On the piano mover's problem: II. General techniques for computing topological properties of real algebraic manifolds. *Adv. in Appl. Math*, 4:298–351, 1983.

[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.

[SS90] J.T. Schwartz and M. Sharir. Algorithmic motion planning in robotics. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume A, pages 157–169. Elsevier, Amsterdam, 1990.

[ST86] N. Sarnak and R.E. Tarjan. Planar point location using persistent search trees. *Comm. ACM*, (29):669–679, 1986.