# On Counting Pairs of Intersecting Segments and Off-Line Triangle Range Searching

M. Pellegrini
Istituto di Matematica Computazionale del CNR
Via S.Maria 46, 56126-Pisa (Italy)

October 6, 1995

**Abstract**

We describe a method for decomposing planar sets of segments and points. Using this method we obtain new efficient *deterministic* algorithms for counting pairs of intersecting segments, and for answering off-line triangle range queries. In particular we obtain the following results:

(1) Given $n$ segments in the plane, the number of pairs of intersecting segments is counted in time $O(n^{1+\epsilon} + K^{1/3}n^{2/3+\epsilon})$, where $K$ is the number of intersection points among the segments, and $\epsilon > 0$ an arbitrarily small constant.

(2) Given $n$ segments in the plane which are coloured with two colours, the number of pairs of *bi-chromatic* intersecting segments is counted in time $O(n^{1+\epsilon} + K_m^{1/3}n^{2/3+\epsilon})$, where $K_m$ is the number of *mono-chromatic* intersection points, and $\epsilon > 0$ is an arbitrarily small constant.

(3) Given $n$ weighted points and $n$ triangles on a plane, the sum of weights of points in each triangle is computed in time $O(n^{1+\epsilon} + \mathcal{K}^{1/3}n^{2/3+\epsilon})$, where $\mathcal{K}$ is the number of vertices in the arrangement of the triangles, and $\epsilon > 0$ an arbitrarily small constant.

The above bounds depend sub-linearly on the number of intersections among input segments $K$ (resp. $K_m$, $\mathcal{K}$), which is desirable since $K$ (resp. $K_m$, $\mathcal{K}$) can range from zero to $O(n^2)$. All of the above algorithms use optimal $\Theta(n)$ storage. The constants of proportionality in the big-Oh notation increase as $\epsilon$ decreases. These results are based on properties of the sparse nets introduced by Chazelle [Cha93].

# 1 Introduction

**Intersection counting and problems in manufacturing.** In geometric models of two and three dimensional objects, intersections of objects are important features. Often intersections are related to (desirable or undesirable) facts in the situation that the model should represent. A typical example is in modelling pieces cut from a flat panel, since the same material cannot belong to two pieces, an intersection of two polygons is a mistake to be avoided in the design of the cutting procedure. The importance of detecting, counting and reporting intersections has been recognized in the early days of computational geometry and a substantial research effort has produced several efficient algorithms for this class of problems in two-dimensional space. If we model the boundaries of objects in the plane using segments we have efficient algorithms for *detecting* intersections and *reporting* them [PS85, Ede87, Meh84] [SH76, NP82, BO79, CE92, CS89, Mul88]. However, important computations in manufacturing would benefit from efficient algorithms for *counting* intersections of segments rather than reporting them. Consider a decomposition of a three-dimensional polyhedral object (e.g. a car engine) obtained by projecting the edges of this object onto a plane, by decomposing the two dimensional arrangement of segments, and by back-projecting the two dimensional decomposition in three-space. Since the direction of projection can be chosen arbitrarily, among several directions one resulting in fewer intersections is preferable. In order to select a good projection, counting intersections is sufficient. In the design of VLSI and PCB boards the rules for minimum distance among wires (each wire is modelled as several overlapping thin polygons[1]) can be checked by expanding each polygon in every direction by half of the minimum distance. An increase in the number of intersections is a symptom of a possible design error. We should notice that, in this example, detection of one intersection is not sufficient since the overlap of polygons forming a single wire is not a design error, while, on the other hand, reporting all intersections is not necessary. Wires of two different types (e.g. polysilicon and diffusion in n-MOS technology) are used to fabricate integrated transistors. In this case we can check design rules for transistors by counting *only* the number of intersections among wires of the two types. An interesting variation on the counting problem is finding the segments contributing the largest number of intersections or the minimum number.

In this paper we describe a new deterministic method for partitioning planar sets of segments and points. Using this method and other tools we are able to obtain improved worst case time bounds for three basic problems. The first problem is that of counting pairs of intersecting segments. The second problem is an important variation of the first one, in

[1] Although it is common in VLSI to use axis-parallel rectangles to model wires during high levels of layout design, after the action of some optimization tools such as homotopic compactation [Mal91] the wires are unrestricted polygons.

which we have two sets of segments (coloured, say, yellow and blue) and we are interested in counting only bi-chromatic pairs of intersecting segments. In this special case we obtain time bounds that improve on those for counting all intersections. The third problem is that of solving off-line simplex range searching queries over sets of points. We obtain a result that holds in any dimension $d$, but the planar case ($d = 2$) is the one where the benefit over previous methods is more evident. Consider the problem of integrating a sampled scalar function of two variables over a series of *possibly intersecting* planar polygonal domains. The supersampling technique used for antialiasing is a case where such problem arises in the area of computer graphics [FVFH90]. If we model each domain as a disjoint union of triangles and the samples as weighted points on the plane, we reduce the integration problem to a case of off-line triangle range-searching for which our efficient algorithm can be used.

**Counting Intersections: Summary of results**. Bentley and Ottman describe a classical algorithm for reporting intersections of segments in time $O((n + K)\log n)$ and $O(n\log^2 n)$ storage, where $K$ is the number of reported intersections and $n$ the number of segments [BO79, PS91]. Work on this problem culminated with several optimal randomized and deterministic algorithms [CE92, CS89, Mul88] with $O(n\log n + K)$ time bounds.

At the moment it is not known how to *count points of intersection* more efficiently than by reporting them, if we allow degenerate sets of segments [2] (see [ES93] for a lower bound argument applicable to quite a large family of algorithms). What we can do efficiently is *count pairs of intersecting segments*. If the input set is in general position the two measures coincide. If the input set is degenerate, then by counting pairs of intersecting segments we always estimate from above the number of intersections. The running time of our algorithms depends on the number of points of intersection. For most of the applications in manufacturing it is sufficient to count intersecting pairs, therefore from now on we will not dwell on the distinction.

Algorithms for reporting intersections can be used to count pairs of intersecting segments, but they are efficient only when there are few intersections (say $K < n\log n$). When the number of intersections is high, algorithms based on plane partitioning have been proposed [Cha86],[GOS88, Aga90a, Aga90b, Cha93]. Currently the best deterministic method uses $O(n^{4/3}\log^{1/3} n)$ time and linear storage for a set of $n$ segments [Cha93]. Note that these methods improve on the reporting algorithms only when $K > n^{4/3}$. In this paper we show a new method for counting intersections among segments whose time bound interpolates between those of previously known deterministic algorithms. We obtain the following result:

**Theorem 1** *Given a set of $n$ segments forming an arrangement with $K$ vertices we can count the pairs of intersecting segments in time $O(n^{1+\epsilon} + n^{2/3+\epsilon}K^{1/3})$, using linear storage, where $\epsilon > 0$ is an arbitrarily small constant.*

[2] A set of relatively open segments is degenerate if it contains three segments sharing a point.

This result improves on previous deterministic methods in the range $n^{1+\epsilon} < K < n^{2-\epsilon}$. For $K < n^{1+\epsilon}$ or $K > n^{2-\epsilon}$, our algorithm is within an $n^\epsilon$ factor from the previously known best algorithms in those ranges. If we allow randomization, then a method in [dBS92] attains expected time $O(n \log n \log K + K^{1/3} n^{2/3} \log^{1/3} n)$.

In the bi-chromatic case we are given two coloured sets of segments, and we are interested in counting only the *bi-chromatic* intersections [MS88, CEGS94, AS90, Aga90b]. Agarwal [Aga90b] gives an algorithm with time and storage bounds $O(n^{4/3} \log^{1.78} n)$. By using the approach in [Cha93] it is possible to reduce the running time to $O(n^{4/3} \log^{1/3} n)$ and the storage to linear. For the special case when there are no monochromatic intersections an $O(n \log n)$ algorithm is described in [CEGS94]. In this paper we obtain the following result:

**Theorem 2** *Given a set of $n$ blue segments forming an arrangement with $K_b$ vertices, and a set of $n$ yellow segments forming an arrangement with $K_y$ vertices, we can count the number of bi-chromatic pairs of intersecting segments in time $O(n^{1+\epsilon} + n^{2/3+\epsilon}(K_y + K_b)^{1/3})$ and linear storage, where $\epsilon > 0$ is an arbitrarily small constant.*

Note that the time bound depends only on the number of monochromatic intersections, and not on the reported number of bi-chromatic pairs of intersecting segments. The bound of our algorithms again interpolates between the two best previous algorithms and improves on both in the range $n^{1+\epsilon} < (K_y + K_b) < n^{2-\epsilon}$.

**On-line versus Off-line simplex range searching.** Simplex range searching has emerged in recent years as one of the basic problems in computational geometry [Wil82, CW89, CY85, EW86, HW87, Meh84, YY85, CSW92, Mat92a]. The problem in its on-line version is the following. We are given $n$ points in $d$-dimensional real space $E^d$ (here $d$ is a constant and the multiplicative constants in the bounds depend on $d$). We wish to answer efficiently the following class of queries: given a query $d$-simplex[3] $q$, count the number of points in $q$. Usually the problem is stated in a slightly more general setting for points with associated weights drawn from a semi-group. In this case the query asks for the cumulative weight of the points in $q$. For simplicity of exposition we will concentrate on the counting problem, but all the results extend to weighted sets of points. In the off-line case, both the points and the query simplices are known at pre-processing time.

A remarkable result of Chazelle [Cha89] is that, if $m$ units of storage are allowed, than for every data organization scheme[4], there exists a query that costs $\Omega(n/m^{1/2})$ for $d = 2$, and $\Omega(n/m^{1/d} \log n)$ for $d > 2$. The final step of the proof of the lower bound rests on an adversary argument. Given any organization of partial sums in a pre-processed data structure, an adversary is going to choose the hardest possible query. Almost matching

[3] A $d$-simplex is the convex hull of $d+1$ linearly independent points in $d$-space. For $d = 2$ it is a triangle, for $d = 3$ a tetrahedron.

[4] Such data structure satisfies very general conditions.

upper bounds have been found [CSW92, Mat92b] which are off the lower bound by small polynomial or polylogarithmic factors. Thus, for a sequence of queries on a set of points, the total cost of the sequence can be bounded by multiplying the number of queries times the worst case cost of a query, plus the cost for preprocessing. A sequence of $n$ queries over $n$ points in $E^d$ has a worst cost $\Omega(n^{4/3})$ for $d = 2$ and $\Omega(n^{2d/(d+1)}/\log^{d/(d+1)} n)$ for $d > 2$. In the off-line case, when the queries are known in advance, Chazelle has recently proved a lower bound $\Omega(n^{2-\frac{2}{d+1}}(\log n)^{-5/2})$ in the semigroup model [Cha95]. The fastest algorithms presented in literature for the off-line case match, up to small factors, the lower bound (see e.g.[Aga90b]). In the planar case we could aim at solving the off-line problem in time, roughly $O((n + \mathcal{K}) \log n)$, by using a sweeping line approach and dynamic data structures, where $\mathcal{K}$ is the number of vertices in the arrangement of query triangles.

The type of argument that leads to the lower bound on a sequence of queries does not take into account the fact that some sequences might be easier than others, and therefore in many practical applications such (upper and lower) bounds may be unduly pessimistic (see e.g. [Kar92]) for both the on-line and the off-line cases. In this paper we study the effect of sparsity on the simplex range searching problem as well as on the intersection counting problem. We consider the complexity of the arrangement of query simplices as a measure on the "sparsity"[5] of the sequence of queries. Our main contribution on this problem is that in the off-line case we obtain a bound on the total cost of a sequence that depends on the number of simplices $n$ and on the sparsity parameter $\mathcal{K}$. In particular we prove:

**Theorem 3** *Given $n$ points and $n$ simplices in $d$-dimensional space we count the number of points in each simplex* deterministically *in time $O(\mathcal{K}^{1/(d+1)} n^{d/(d+1)+\epsilon} + n^{f(d)+\epsilon})$, where $\mathcal{K}$ is the number of vertices in the arrangement of query simplices, $\epsilon$ an arbitrarily small constant, and $f(d) = (2d^2 - 4d + 1)/(d^2 - d - 1)$. The storage used is $O(n)$.*

For the case $d = 2$, as a corollary, we obtain thus a bound $O(n^{1+\epsilon} + \mathcal{K}^{1/3} n^{2/3+\epsilon})$, which improves on the previous fastest algorithms in the range $n^{1+\epsilon} < \mathcal{K} < n^{2-\epsilon}$.

Our result leaves open the question whether in the *on-line* case one can obtain bounds on the cost of answering a sequence of queries which depend on the sparsity of the sequence.

**The method used.** We extend a method for partitioning of a set of simplices, developed in [Pel94b], which is based on the properties of sparse nets introduced by Chazelle [Cha93], to the case of simplices and points.

The general strategy is a divide and conquer approach. We have an input set $S$ of $(d-1)$-simplices[6] and an input set $M$ of points. We select a subset of $S$ that is used to partition

[5]Naturally one could prefer other ways to capture the informal notion of sparsity. An advantage of choosing the complexity of the arrangement is that this measure is invariant under projective transformations.

[6]A $(d-1)$-simplex is the convex hull of $d$ linearly independent points in $d$-space. It is segment for $d = 2$, a triangle for $d = 3$.

$E^d$ into cells of constant descriptive complexity (also called elementary cells) in such way as to balance the number of elements of $S$ intersecting each elementary cell and the number of points in $M$ inside each cell. We apply the method recursively in each elementary cell. The elementary cells are then organized in a *partition* tree. The gist of the method is to keep a tight control on the size of the partition tree. This tight counting is attained using the sparse nets [Cha93] to select the subset of $S$. The construction of the partition tree is carried on up to a certain depth. At the leaves of the tree we use known methods that are not sensitive to the sparsity of the input. In order to count pairs of intersecting segments we specialize the analysis for $d = 2$ and we add auxiliary computations specific to the intersection counting problem. In the second part of the paper we use the partition tree to solve a variation of Hopcroft's problem, which consists in detecting incidences of points and $(d-1)$-simplices[7] in $E^d$. Once we have a solution to this problem we add auxiliary operations on the data stored in the partition tree to obtain the main result on off-line simplex range searching (Theorem 3).

The paper is organized as follows. In Section 2 we recall the main properties of sparse nets and several useful operations. In Section 3 we discuss a method for obtaining a balanced partition tree for simplices and points, which is used as a basis by the other algorithms. In section 4 we discuss the algorithm to count the intersections in a set of segments. In Section 5 we discuss how to count efficiently bi-chromatic intersections of coloured segments. In Section 6 we discuss a variation of Hopcroft's problem for points and simplices. In Section 7 we extend the solution to Hopcroft's problem to solve the off-line simplex range searching problem.

# 2 Sparse Nets and Cuttings

For sake of completeness we summarize here some definitions and constructions taken mainly from [Cha93]. Let $H$ be a set of $n$ hyperplanes in Euclidean $d$-dimensional space $E^d$. 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 $s$. Let $\epsilon$ be a positive real number with $0 < \epsilon \leq 1$.

**Definition 1** *$R$ is a $\epsilon$-approximation for $H$ if, for any segment $e$:*

$$\left| \frac{R_e}{\rho} - \frac{H_e}{n} \right| < \epsilon.$$

[7]A randomized method for the planar case is given in [dBS92].

**Definition 2** *$R$ is a $\epsilon$-net for $H$ if for any segment $e$, $H_e > \epsilon n$ implies $R_e > 0$.*

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

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

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

The next natural question is how fast can we compute approximations, sparse nets and cuttings for hyperplanes in $d$-dimensional space. Let $s$ be a $d$-simplex in $E^d$. We denote by $H(s)$ the subset of hyperplanes of $H$ intersecting $s$. Computing a sparse net for $(H(s), s)$ directly can be time-consuming, therefore first an approximation $A$ of $H(s)$ is generated, then a sparse net for $(A, s)$ is computed. In particular, let $r_0$ be a positive integer constant and $i$ an integer such that $n/r_0^{i-1} > |H(s)| > n/r_0^i$. Define $\rho_0 = r_0^i|H(s)|/n$ and $\rho = \rho_0 \log \rho_0$. The following lemma states the important property that a sparse net of an approximation cannot generate an arrangement with too many vertices in $s$:

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

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

By exploiting the fact that $r_0$ is a constant we can actually compute the approximation $A$ and the sparse net $R$ of Lemma 1 quite efficiently. We will use such construction as a black-box subroutine and we refer the reader to [Cha93, Mat91] for details.

**Lemma 2 ([Cha93])** *The approximation $A$ and the sparse net $R$ of Lemma 1 is computed in time $O(|H(s)|)$, once $H(s)$ is given.*

Our final objective in this section is to compute a cutting, that is a decomposition of (a portion of) $E^d$ into simplices with some good intersection property with respect to the set of hyperplanes $H$. We proceed by taking the set of hyperplanes $R$ of Lemma 1 and by building the arrangement $\mathcal{A}(R)$ of such hyperplanes with $s = E^d$ (see [Ede87, EOS86, ESS93]). Moreover, we subdivide each convex cell of $\mathcal{A}(R)$ into $d$-simplices. We can attain this decomposition, for example, by triangulating each facet of a cell recursively, and then by joining the $(d-1)$-dimensional simplices so obtained to the bottom-most vertex of the cell (see e.g. [Cla87]). The collection of all the $d$-simplices so generated is called a triangulation $\mathcal{T}(R, s)$ of $\mathcal{A}(R)$ within $s$. We compute for each cell $\sigma \in \mathcal{T}(R, s)$ the set $H(\sigma)$ and repeat the above construction recursively. The following lemma tells us that we will actually achieve our goal of computing a cutting with the desired intersection property.

**Lemma 3 ([Cha93])** *The triangulation $\mathcal{T}(R, s)$ of the sparse net $R$ of Lemma 1 is a $1/r_0$-cutting for $H(s)$ in $s$. The size of $\mathcal{T}(R, s)$ is $O(\rho^{d-1} + v(R, s))$ and it is computed in time $O(\rho^{d-1} + v(R, s))$, once $R$ is given.*

Again we will use the above triangulation algorithm as a black-box procedure in the rest of the paper.

So far we have dealt only with an input composed of hyperplanes. In order to deal with an input composed on $(d-1)$-simplices we need two additional important definitions. We fix once and for all a vertical direction in $E^d$. A $(d-1)$-simplex $t$ *partially covers* a $d$-simplex $s$ if $t$ intersects $s$ and some $(d-2)$-face of $t$ intersects $s$. If a $(d-1)$-simplex $t$ partially covers a $d$-simplex $s$ then the vertical projection of some $(d-2)$-face of $t$ will intersect the vertical projection of $s$. A $(d-1)$-simplex $t$ *totally covers* a $d$-simplex $s$ if $t$ intersects $s$ and no $(d-2)$-face of $t$ intersects $s$.

# 3 Partitioning points and $(d-1)$-simplices in $E^d$

In this section we describe a general algorithm for partitioning sets of points and $(d-1)$-simplices. The objective is to maintain a tight control on the number of simplices and points incident to any cell associated with nodes of a partition tree, as well as a tight control on the size of the partition tree. We are given a set $M$ of $m$ points in $E^d$ and a set $S$ of $n$ $(d-1)$-simplices. We denote with $K$ the number of vertices in the arrangement of $S$, and with $r$ a positive integer smaller than $n$. Our aim in this section is to establish the following result:

**Lemma 4** *We can build a partition tree $\mathcal{P}$ such that the region $s$ associated with any leaf of the partition tree contains at most $m/r^{d-1}$ points of $M$, such that $s$ is associated with a subset of at most $n/r$ simplices of $S$ partially covering $s$, and such that $s$ is associated with a subset of at most $O(n/r)$ simplices of $S$ totally covering $s$. $\mathcal{P}$ and all data sets associated with the internal nodes, as well as with the leaves, use storage $O(nr^{d-2+\epsilon} + n(K/n^d)r^{d-1} + mr^\epsilon + m(K/n^d)r)$, where $\epsilon$ is a fixed positive constant that can be chosen arbitrarily small. The same bound holds for the time needed to construct $\mathcal{P}$.*

Note that the size of $\mathcal{P}$ depends explicitly on the complexity $K$ of the arrangement $\mathcal{A}(S)$. Moreover, the freedom we have in choosing the value for $r$ will be used later to balance the costs of the auxiliary computations we will perform over the data stored in $\mathcal{P}$. Although Lemma 4 states some of the most important properties of the tree $\mathcal{P}$, it is not a full characterization of $\mathcal{P}$. In order to derive the results of Theorems 1, 2 and 3 we will exploit also other properties of $\mathcal{P}$ that are implicit in the construction method.

## 3.1 The method

It is useful to look at the partition tree we are going to build in two different ways. In the first view, which is used in this section, we concentrate on the levels of the partition tree and we are mainly interested in counting the number of nodes in each level. In the second view we shall consider the partition tree as a two-layered tree, with a backbone tree and auxiliary trees.

We choose a constant $r_0$ that is large enough to satisfy the conditions implicit in the subsequent proofs and a free parameter $r$, with $r_0 \leq r \leq n$, which governs the depth of the partition tree. We build a sequence of sets $C_0, .., C_l$, with $l = \log_{r_0} r$, where each set represents a level of the partition tree. The set $C_i$ is a collection of quadruples $(s, P(s), Q(s), M(s))$ where $s$ is $d$-simplex in $E^d$ or a $d$-cylinder (i.e. a degenerate $d$-simplex with a vertex at infinity), also called an elementary cell. $P(s)$ is a subset of simplices in $S$ *partially covering* $s$, $Q(s)$ is a subset of simplices in $S$ *totally covering* $s$, $M(s)$ is a set of points of $M$ in $s$. The invariants maintained over the sets $C_i$ are:

1. $|P(s)| \leq n/r_0^i$,
2. $|Q(s)| \leq nr_0/r_0^i$,
3. $|M(s)| \leq m/r_0^{i(d-1)}$.

The first set is $C_0 = (E^d, S, \emptyset, M)$ which trivially satisfies our invariants. The algorithm to construct $C_k$ from $C_{k-1}$ works in three main phases.

(1) **Algorithmic step.** For any $s$ in $C_{k-1}$, if $P(s)$ is empty we skip phase (1). If $|P(s)| \leq n/r_0^k$ then we build a region $\sigma$ with $\sigma := s$, $Q(\sigma) := \emptyset$, $P(\sigma) := P(s)$ and $M(\sigma) := M(s)$. Otherwise, we are in the conditions for applying the construction of Lemma 1. Project vertically the simplices in $P(s)$ and $s$ in $(d-1)$-dimensional space obtaining a set of $(d-1)$-simplices in $(d-1)$-space $P'(s)$ and a $(d-1)$-simplex $s'$. Extend each $(d-2)$-face of simplices in $P'(s)$ into a full hyperplane and make a sparse net of Lemma 1 for this set of hyperplanes within $s'$, with parameter $\rho_0'' = r_0^k|P(s)|/n$. Build a triangulation $\mathcal{T}$ of the net and obtain a set of $(d-1)$-dimensional elementary cells. Extend the elementary cells vertically in $d$-space within $s$, obtaining a set of cylinders $\Sigma$. Let $\eta$ be a cylinder in $\Sigma$. Set $M(\eta) := M(s) \cap \eta$, $Q(\eta)$ equal to the subset of simplices in $P(s)$

totally covering $\eta$, and $P(\eta)$ equal to the subset of simplices in $P(s)$ partially covering $\eta$.

**Invariants**. Inductively $|P(s)| \leq n/r_0^{k-1}$ and we choose $\rho_0'' = r_0^k|P(s)|/n$ so as to obtain $|P(\eta)| \leq n/r_0^k$ and $|Q(\eta)| \leq |P(s)| \leq r_0 n/r_0^k$. Note again that for every $\eta \in \Sigma$ we have $\rho_0'' < r_0$.

(2) **Algorithmic step.** For any elementary cell $s$ in $C_{k-1}$, if $Q(s)$ is empty skip phase (2). If $|Q(s)| \leq r_0 n/r_0^k$, then generate a cell $\sigma$ with $\sigma := s$, $P(\sigma) := \emptyset$, $Q(\sigma) := Q(s)$ and $M(\sigma) := M(s)$. Otherwise, by induction hypothesis we are in the conditions for applying Lemma 1. Build a sparse net of Lemma 1 with parameter $\rho_0' = r_0^{k-1}|Q(s)|/n$ for the hyperplanes spanning simplices in $Q(s)$, restricted to $s$. Build a triangulation $\mathcal{T}$ of the net, as described in Section 2, thus obtaining a set of simplices $\Sigma$. For every $\sigma \in \Sigma$, set $P(\sigma) := \emptyset$, $M(\sigma) := M(s) \cap \sigma$ and $Q(\sigma)$ equal to the set of simplices in $Q(s)$ that meet $\sigma$.

**Invariants**. By induction we have that $r_0 n/r_0^k \leq |Q(s)| \leq r_0 n/r_0^{k-1}$. We choose the parameter $\rho_0' = r_0^{k-1}|Q(s)|/n$ and thus obtain $|Q(\sigma)| \leq r_0 n/r_0^k$ as follows from Lemma 1. Also note that $\rho_0' \leq r_0$ for every $s \in C_{k-1}$, which is an important property we shall use in the analysis of the algorithms.

(3) **Algorithmic step.** We consider, for each cell $\sigma$ built in phase (1) and (2), the points in $M(\sigma)$. If $M(\sigma)$ does not satisfy the third invariant, we split $\sigma$ into subcells with at most $m/r_0^{k(d-1)}$ points. Any greedy method will do for our purposes. For each cell $s \in C_{k-1}$ we introduce at most $r_0^{d-1}$ new cells in this phase.

We collect all the cells generated in phases (1), (2) and (3) in the set $C_k$. The set $C_k$ satisfies all three invariants. The simplices produced are organized in a two-layered search tree. The primary tree consists of cells built in phase (1) with the refinement in phase (3), and is called the $P$-tree. The secondary trees correspond to the cells built in phase (2), with the refinement if phase (3), and are called $Q$-trees.

## 3.2 Analysis of the algorithm

We denote by $s'$ the vertical projection of a simplex $s$ onto $(d-1)$-dimensional subspace, and by $R'$ the sparse net for $P'(s)$. We use Lemmas 1 and 3 to bound the number of simplices obtained at each iteration of the algorithm. We obtain the following inequalities in which we denote with $c_1, c_2, ...$ absolute multiplicative constants that depend on $d$. Notice that $\sum_{s \in C_{k-1}} v(P'(s), s') = O(n^{d-1})$. We denote with $K$ the total number of points which are the intersection of $d$ $(d-1)$ simplices in $S$. From the definition we have $K \leq \binom{n}{d}$.

$$\begin{aligned}
|C_k| &\leq \sum_{s\in C_{k-1}} c_1[\rho_0'^{d-1}\log^{d-1}\rho_0' + v(R,s) + r_0^{d-1}] + \sum_{s\in C_{k-1}} c_2[\rho_0''^{d-2}\log^{d-2}\rho_0'' + v(R',s') + r_0^{d-1}] \\
&\leq \sum_{s\in C_{k-1}} c_3[r_0^{d-1}\log^d r_0 + (r_0^k \log r_0/n)^d v(Q(s),s) + r_0^{d-1}] \\
&\quad + \sum_{s\in C_{k-1}} c_4[r_0^{d-2}\log^{d-1} r_0 + (r_0^k\log r_0/n)^{d-1} v(P'(s),s') + r_0^{d-1}] \\
&\leq \sum_{s\in C_{k-1}} c_5[r_0^{d-1}\log^d r_0 + (r_0^k\log r_0/n)^d v(Q(s),s) + (r_0^k \log r_0/n)^{d-1} v(P'(s),s')] \\
&\leq c_6[r_0^{d-1}(\log^d r_0)|C_{k-1}| + r_0^{kd}(\log^d r_0)(K/n^d) + r_0^{k(d-1)}(\log^{d-1} r_0)].
\end{aligned}$$

We obtain the following recursive inequality in the variable $k$:

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

**Lemma 5** $|C_k| \leq Dr_0^{k(d-1+\epsilon)} + F(K/n^d)r_0^{kd}$, *where $F$ and $D$ are constants with respect to $k$, $n$ and $K$.*

*Proof.* The recursive inequality is the same as one derived in [Pel94b] and the proof of the lemma is in [Pel94b]. ∎

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

$$\sum_{k=0}^{l} |C_k| \leq \sum_{k=0}^{l}[Dr_0^{k(d-1+\epsilon)} + F(K/n^d)r_0^{kd}]. \tag{2}$$

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 $O(Dr_0^{l(d-1+\epsilon)}+F(K/n^d)r_0^{ld}) = O(r^{d-1+\epsilon}+(K/n^d)r^d)$ on the number of nodes of the partition tree. The time spent on each $d$-simplex in the partition tree is linear in the number of $(d-1)$-simplices intersecting it by Lemma 2. So the construction of the nets in phases (1) and (2) has the following cost:

$$\begin{aligned}
&\sum_{k=0}^{l}(nr_0/r_0^k)|C_k| \leq \\
&\sum_{k=0}^{l}(nr_0/r_0^k)[Dr_0^{k(d-1+\epsilon)} + F(K/n^d)r_0^{kd}] \leq \\
&\sum_{k=0}^{l}(nr_0)[Dr_0^{k(d-2+\epsilon)} + F(K/n^d)r_0^{k(d-1)}].
\end{aligned}$$

We obtain two sums of geometric series of ratio greater than 1 which are proportional to the last element of the series. Thus we have a bound $O(n[r^{d-2+\epsilon} + (K/n^d)r^{d-1}])$ on the time used on phases (1) and (2). The location of the points and the splitting of phase (3) is done in time proportional to the number of points $|M(s)|$. Phase (3) over the whole algorithm costs:

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

We have two sums of geometric series of ratio $r_0^\epsilon$ and $r_0$, which are bounded by $O(mr^\epsilon + m(K/n^d)r)$. This analysis accounts for the construction of a partition tree that is common to several applications described in this paper.

# 4 Counting intersections of segments

Given a set $S$ of $n$ segments in $E^2$, let $\mathcal{A}(S)$ be the arrangement formed by the segments in $S$ and let $K$ be the number of vertices in such an arrangement. We specialize the construction of Section 3 for $d = 2$ and we use as the set $M$ of input points the end-points of the segments in $S$. Thus, by using the algorithm of the previous section we obtain a partition tree whose leaves correspond to the set $C_l$. Let $\sigma$ be one such leaf. The following conditions hold:

1. $|P(\sigma)| \leq n/r$,
2. $|Q(\sigma)| \leq nr_0/r$,
3. $|M(\sigma)| \leq n/r$,
4. $|C_l| = O(r^{(1+\epsilon)} + (K/n^2)r^2)$.

We build the partition tree up to level $l = \log_{r_0}(n^2/K)$, corresponding to $r = n^2/K$. We consider several types of intersections that can be accounted for using different techniques on the search tree. The main difficulty lies in the fact that during the construction of the partition tree we separate totally covering segments from the partially covering ones. We will describe a scheme for "merging" back again those sets of segments.

We shall discuss how to count intersections in a set $P(s)$, for a generic cell $s$. The result follows by starting with the whole space, $s = E^d$. Let $P(s)$ be the set of simplices partially covering $s$. We then apply phases (1) and (3) of the main algorithm obtaining a set of elementary cells $\Sigma$ whose union is $s$. For each cell $\sigma$ we obtain sets $P(\sigma)$ and $Q(\sigma)$. We consider the following sets of intersections:

(i) The set $QQ(\sigma)$ of intersections between two segments in $Q(\sigma)$ clipped in $\sigma$.

(ii) The set $PQ(\sigma)$ of intersections between a segment in $Q(\sigma)$ and a segment in $P(\sigma)$, clipped in $\sigma$.

(iii) The set $PP(\sigma)$ of intersections between two segments in $P(\sigma)$ clipped in $\sigma$.

Clearly:

$$|PP(s)| = \sum_{\sigma \in \Sigma} |QQ(\sigma)| + |PQ(\sigma)| + |PP(\sigma)|.$$

Next, we describe how to compute each type of intersection:

(1) The intersections in $QQ(\sigma)$ can be computed in time $O(|Q(\sigma)| \log |Q(\sigma)|)$ by using a method in [Aga90a, Lemma 3.1], which is based on counting inversions in permutations.

(2a) The intersections in $PP(\sigma)$ are counted recursively using the method we are describing for $PP(s)$, unless $\sigma$ is a leaf of the upper search tree.

(2b) If $\sigma$ is a leaf of the upper search tree, we apply directly the method in [Cha93], which uses time $O(|Q(\sigma)|^{4/3} \log^{1/3} |Q(\sigma)|)$ and storage $O(|Q(\sigma)|)$.

(3a) The intersections $PQ(\sigma)$ are computed in the following way. Let us consider the $Q$-tree rooted at $\sigma$ and built by repeated applications of phases (2) and (3) of the algorithm of Section 3. Note that the points $M(\sigma)$ that we have used in phase (3) are the endpoints of segments in $P(\sigma)$. Each point in $M(\sigma)$ will appear in one cell for each level of the $Q$-tree rooted at $\sigma$. For each such cell and for each point, we find the sibling cells intersecting the segment in $P(\sigma)$ incident to that point. Since the $Q$-tree has degree bounded by a constant, each segment of $P(\sigma)$ is associated with at most a constant number of cells for each level of the $Q$-tree. Moreover, for each level, a segment crosses all but at most two of the cells to which it has been associated at that level. Now, we can count the intersections between $Q$-sets at a cell and crossing associated segments again by using Lemma 3.1 in [Aga90a]. Thus, accounting for these intersections will cost overall at most $O((|P(\sigma)| + |Q(\sigma)|) \log^2 n)$ over the whole $Q$-tree rooted at $\sigma$. If an associated segment is short at a level of the $Q$-tree, than we deal with that segment recursively on the next level of the $Q$-tree rooted at that cell.

(3b) When a segment from $P(\sigma)$ is associated to a leaf $\eta$ of the $Q$-tree and it is short for that leaf, we use at that leaf a non-sensitive bi-chromatic method [Aga90b, Cha93] on the set $Q(\eta)$ and on the set of segments in $P(\sigma)$ incident to points in $M(\eta)$. From the invariants we have that the input to the non-sensitive method at a leaf is of size at most $2n/r$.

The proof of correctness of this method derives from the fact that all possible pairs of intersecting segments are accounted for. The details are routine and are left as an exercise. Next, we analyze the cost of the algorithm.

(I) The total cost for setting up the partition tree is:

$$nr^{\epsilon} + rn(K/n^2) + nr^{\epsilon} + n(K/n^2)r.$$

For $r = n^2/K$, we obtain a cost $O(n^{1+\epsilon})$.

(II) The cost of the accounting on all levels of the partition tree except the leaves is:

$$\sum_{k=0}^{l}[(nr_0/r_0^k)^{1+\epsilon} + (n/r_0^k)^{1+\epsilon}]|C_k| \leq$$
$$\sum_{k=0}^{l}[(nr_0/r_0^k)^{1+\epsilon}][Dr_0^{k(1+\epsilon)} + F(K/n^d)r_0^{k2}] \leq$$
$$\sum_{k=0}^{l}(nr_0)[Dr_0^0 + F(K/n^d)r_0^{k(1-\epsilon)}].$$

We obtain two sums of geometric series of ratio grater than 1, which are proportional to the last element of the series. We obtain a bound $O(n[\log r + (K/n^2)r^{1-\epsilon}])$ which, for $r = n^2/K$, is $O(n^{1+\epsilon})$.

(III) The cost incurred at the leaves of the partition tree is:

$$\sum_{\sigma \in C_l}(n/r)^{4/3}\log^{1/3} n \leq (r^{1+\epsilon} + (K/n^2)r^2)(n/r)^{4/3}\log^{1/3} n.$$

By substituting our choice of $r$, we obtain a bound $O(n^{2/3+\epsilon}K^{1/3})$.

We can reduce the working storage to linear in $n$ by building the partition tree in a depth first manner. The working storage is used to store one path from the root to one leaf of the partition tree, plus all siblings of the nodes on the path, together with the associated sets $Q$, $P$ and $M$. The size of these sets decreases geometrically, therefore the sum of all the size of sets along the path is linear in $n$. The result claimed in the introduction is almost proved, except for the issue of how the algorithm guesses the correct value of the parameter $r$, which depends on the unknown value of $K$. We overcome this problem by using a well known trick of doubling. We start with guess $K_0 = n$ and we run the algorithm until it exceeds the time bound. At this point we double the guess using the general rule $K_i = 2K_{(i-1)}$. There are at most a logarithmic number of guesses before the algorithm terminates having counted all of the pairs of intersecting segments. The time bound is unchanged except for a slightly greater value of $\epsilon$. This trick will be used also for the other results in this article. The above discussion constitutes the proof of Theorem 1.

# 5 Bi-chromatic intersections

## 5.1 Building a bi-chromatic partition tree

In the bi-chromatic case we show an algorithm whose time bound depends only on the number of mono-chromatic intersections. Because of this such an algorithm may be much faster than the general algorithm to count segment intersections, when the two coloured input sets are separately sparse. We are given a set $A$ of $n$ yellow segments and a set $B$ of $n$ blue segments. The problem is to count the number of yellow-blue intersecting pairs of segments.

The general approach is to build a partition tree that stores the segments and their end-points compactly, and then add auxiliary computations that use such data. Each node of the partition tree is associated with a region $s$ of the plane. Moreover, for any such region $s$, we will consider the following sets:

(1) $P_y(s)$ of yellow segments partially covering $s$,

(2) $Q_y(s)$ of yellow segments totally covering $s$,

(3) $M_b(s)$ of end-points of blue segment in $s$,

(4) $P_b(s)$ of blue segments partially covering $s$,

(5) $Q_b(s)$ of blue segments totally covering $s$,

(6) $M_y(s)$ of end-points of yellow segments in $s$.

We will describe the partition tree as a two-level tree where each node of the primary tree (called $P$-tree) is associated with two secondary trees (called $Q_y$-tree and $Q_b$-tree). At the root of the primary tree, the region of interest is $s = E^d$, and we set $P_y(s) = A$, $P_b(s) = B$, $Q_y(s) = \emptyset$, $Q_b(s) = \emptyset$. $M_y(s)$ consists of the end-points of segments in $A$, and $M_b(s)$ consists of the endpoints of segments in $B$. Now, we show how to build recursively the next level of the $P$-tree. Let $s$ be a region associated with a node of the $P$-tree. We build a cell decomposition for $P_y(s)$ and $M_b(s)$, and a separate decomposition for $P_b(s)$ for $M_y(s)$, by using phases (1) and (3) of the algorithm of Section 3. We obtains sets $\Sigma_y$ and $\Sigma_b$ which are sets of vertical strips. We merge the two decompositions obtaining a set $\Sigma$ of cells. Note that this merge ensures that we build a *single* primary tree. Since $\Sigma_y$ and $\Sigma_b$ are just back-projections of a 1-dimensional arrangement, their intersection has size $O(r_0)$. This fact helps to keep the branching factor of the partition tree small.

For any $\sigma \in \Sigma$, we set $P_y(\sigma)$ (resp. $P_b(\sigma)$) to be the set of segments in $P_y(s)$ (resp. $P_b(s)$ ) partially covering $\sigma$, $Q_y(\sigma)$ (resp. $Q_b(\sigma)$) to be the set of segments in $P_y(s)$ (resp. $P_b(s)$) totally covering $\sigma$, $M_y(\sigma) := M_y(s) \cap \sigma$, and $M_b(\sigma) := M_b(s) \cap \sigma$. We build the auxiliary $Q_y$-tree rooted at $\sigma$ by repeatedly applying phases (2) and (3) of the algorithm in Section 3 with inputs simplices $Q_y(\sigma)$ and points $M_b(\sigma)$. We build the auxiliary $Q_b$-tree rooted at $\sigma$ by repeatedly applying phases (2) and (3) of the algorithm in Section 3 with inputs simplices $Q_b(\sigma)$ and points $M_y(\sigma)$. The primary $P$-tree is expanded from $\sigma$ using as data $P_y(\sigma)$, $PMb(\sigma)$, $P_y(\sigma)$, and $M_b(\sigma)$.

If we consider the partition tree as a unique tree (without distinguishing primary and secondary structures) it is easy to show that for a cell $s$ at level $i$ the following invariants hold:

1. $P_y(s) \leq n/r_0^i$ (yellow segments partially covering $s$),
2. $Q_y(s) \leq nr_0/r_0^i$ (yellow segments covering $s$),
3. $M_b(s) \leq n/r_0^i$ (end-points of blue segment in $s$),
4. $P_b(s) \leq n/r_0^i$ (blue segments partially covering $s$),
5. $Q_b(s) \leq nr_0/r_0^i$ (blue segments covering $s$),
6. $M_y(s) \leq n/r_0^i$ (end-points of yellow segments in $s$).

As in Section 3, we first write a recursive equation linking the number of nodes at level $k$ to the number of nodes at level $k-1$. We denote with $v(R_y, s)$ (resp. $v(R_b, s)$) the number of vertices of a sparse net of yellow (resp. blue) segments totally covering $s$, and with $\rho_{0y}$ and $\rho_{0b}$ the parameters of the yellow and blue sparse nets, chosen according to the rules of Section 3.

$$
\begin{aligned}
|C_k| &\leq \sum_{s\in C_{k-1}} c_2 r_0 + \sum_{s\in C_{k-1}} c_1[\rho_{0y}\log\rho_{0y} + v(R_b,s) + r_0] + \\
&\quad \sum_{s\in C_{k-1}} c_1[\rho_{0b}\log\rho_{0b} + v(R_y,s) + r_0] \\
&\leq \sum_{s\in C_{k-1}} c_4 r_0 + \sum_{s\in C_{k-1}} c_3[r_0\log^2 r_0 + (r_0^k \log r_0/n)^2 v(Q_y(s),s) + r_0] + \\
&\quad \sum_{s\in C_{k-1}} c_3[r_0\log^2 r_0 + (r_0^k\log r_0/n)^2 v(Q_b(s),s) + r_0] \\
&\leq \sum_{s\in C_{k-1}} c_5[r_0\log^2 r_0 + (r_0^k\log r_0/n)^2[v(Q_y(s),s) + v(Q_b(s),s)] + r_0] \\
&\leq c_6[r_0(\log^2 r_0)|C_{k-1}| + r_0^{2k}(\log^2 r_0)((K_y+K_b)/n^2) + r_0^k(\log r_0)].
\end{aligned}
$$

We obtain the following recursive inequality in the variable $k$:

$$
|C_k| \leq c_6[r_0(\log^2 r_0)|C_{k-1}| + r_0^{2k}(\log^2 r_0)((K_y+K_b)/n^2) + r_0^k(\log r_0)]. \tag{3}
$$

The solution is the same as in Lemma 5 for $d = 2$, with the difference that $K$ in Lemma 5 is now to be interpreted as $K_y + K_b$, i.e. the number of monochromatic intersections.

## 5.2 Counting bi-chromatic intersections

For a generic region $s$ associated with a node in the bi-chromatic partition tree we consider the following sets of intersections:

1. The set $Q_bQ_y(s)$ of intersections between segments of $Q_b(s)$ and $Q_y(s)$ clipped in $s$.
2. The set $P_bP_y(s)$ of intersections between segments of $P_b(s)$ and $P_y(s)$ clipped in $s$.
3. The set $Q_bP_y(s)$ of intersections between segments of $Q_b(s)$ and $P_y(s)$ clipped in $s$.
4. Symmetrically, we define the set $Q_yP_b(s)$.

Clearly, our objective is to compute $P_yP_b(s)$ when $s = E^d$, i.e. at the root of the bi-chromatic $P$-tree. It is easy to show that the following relation holds between a node $s$ and its children in the $P$-tree:

$$
|P_yP_b(s)| = \sum_{\sigma\in\Sigma} |Q_yP_b(\sigma)| + |Q_bP_y(\sigma)| + |Q_bQ_y(\sigma)| + |P_bP_y(\sigma)|.
$$

Now, we augment the partition tree with the methods for counting the different types of intersections.

1. The intersections in $Q_yQ_b(\sigma)$ can be computed in time $O(|Q(\sigma)|\log|Q(\sigma)|)$ by using a method in [Aga90a, Lemma 3.1].

2a. The intersections in $P_yP_b(\sigma)$ are counted recursively using the method we are describing for $P_yP_b(s)$, unless $\sigma$ is a leaf of the $P$-tree.

2b. The method for counting intersections described in [Cha93] can be easily changed in a method for counting bi-chromatic intersections within the same time and storage bounds. If $\sigma$ is a leaf of the upper search tree, we apply directly the variation of the method in [Cha93] in order to count pairs in $P_yP_b(\sigma)$.

3. We can count the intersections $Q_bP_y(\sigma)$ by using, on the $Q_b$-tree rooted at $\sigma$, the methods at (3a) and (3b) in Section 4.

4. Similarly, we can compute $Q_yP_b(\sigma)$ on the $Q_y$-tree rooted at $\sigma$.

An analysis similar to the monochromatic case accounts for all of cost associated to tracing the four types of intersections in the tree. With the caveat the $K$ is now only the number of monochromatic intersections, the time analysis is exactly as in the case of non coloured segments. Again, by expanding the partition tree in depth first manner and by storing at any given time only the data relative to one path from the root to a leaf, plus the siblings of nodes on the path, we can achieve linear storage. This concludes the proof of Theorem 2.

# 6 An Incidence problem for points and $(d-1)$-simplices

In the second part of this paper we derive a result for off-line simplex range searching. As a preliminary step we solve the problem of detecting an incidence between $(d-1)$-simplices and points in $d$-space. We assume $d \geq 2$. Let us consider the construction of the partition tree in the Section 3. It is easy to see that, if a point and a simplex are incident, then such point and simplex are associated to one leaf of the partition tree. We recall that the leaves of the partition tree are associated with cells in $C_l$, and each cell $\sigma \in C_l$ has associated sets $P(\sigma)$, $Q(\sigma)$ and $M(\sigma)$ with the following conditions:

1. $|P(\sigma)| \leq n/r$,
2. $|Q(\sigma)| \leq nr_0/r$,
3. $|M(\sigma)| \leq m/r^{(d-1)}$.

For the remainder of this chapter, in order to keep the calculations simple we assume that $m = n$ and we drop constant, logarithmic and epsilon factors. We will take them into account in the statement of the final result.

Our intermediate goal is to describe a non-output-sensitive method to detect incidences. Let us set $q = r^{d-1+\epsilon} + (K/n^d)r^d$, which is proportional to the number of leaves of the tree. The number of simplices cutting each cell at a leaf is $O(n/r)$. We take the points at each leaf and we divide them in groups of equal size $m/q$. This operation takes time[8] $O(qm/r^{(d-1)})$.

The incidence of a $(d-1)$-simplex and a point can be expressed as a conjunction of inequalities involving bilinear forms whose terms are functions of the coordinates of the point and of the linear subspaces supporting faces of the $(d-1)$-simplices (see e.g [Pel94a] for a general treatment of these classes of problems). Using results in [Pel94a] and the deterministic cuttings in [Mat91] (see also [CF90]) it is possible to find whether in a point of a set of $p_i$ points is incident to a $(d-1)$-simplex in a set of $h_i$ simplices deterministically in time $O(p_i^{d/(d+1)+\epsilon} h_i^{d/(d+1)} + p_i^{1+\epsilon} + h_i \log^{d+2} p_i)$. Also, by applying a depth-first strategy in the order of executions of subproblems in the algorithm in [Pel94a], it is possible to use only $O(n+m)$ storage.

The total cost for building the partition tree is:

$$mr^\epsilon + rm(K/n^d) + nr^{d-2+\epsilon} + n(K/n^d)r^{d-1}.$$

At the leaves of the partition tree we spend time:

$$mq/r^{d-1} + q[(m/q)^{d/(d+1)}(n/r)^{d/(d+1)} + m/q + n/r].$$

Thus, modulo constant, logarithmic and $\epsilon$-factors we have a cost:

$$n + nr^{d-2} + n(K/n^d)r^{d-1} + nq/r + n^{2d/(d+1)}q^{1/(d+1)}r^{-d/(d+1)}.$$

We obtain the best performance by finding the value for the parameter $r$ that balances the several costs. To simplify the analysis we consider two cases.

1. **Case** $r^{d-1} > (K/n^d)r^d$.

   In this case $q < 2r^{d-1}$, therefore $nq/r = nr^{d-2}$. Also, since $r < n$, we have $r^{d-1} < nr^{d-2}$. Therefore the cost is:

   $$n + nr^{d-2} + nr^{d-2} + nr^{d-2} + n^{2d/(d+1)}q^{1/(d+1)}r^{-d/(d+1)}.$$

   The optimal trade-off is:

[8] A rough estimate but sufficient for our purposes.

$$
\begin{aligned}
&nr^{d-2} = n^{2d/(d+1)}r^{(d-1)/(d+1)}r^{-d/(d+1)}\\
&nr^{d-2} = n^{2d/(d+1)}r^{-1/(d+1)}\\
&r^{d-2+1/(d+1)} = n^{2d/(d+1)-1}\\
&r^{d-2+1/(d+1)} = n^{2d/(d+1)-1}\\
&r^{(d^2-d-1)/(d+1)} = n^{(d-1)/(d+1)}\\
&r = n^{(d-1)/(d^2-d-1)}.
\end{aligned}
$$

So the cost is $O(nn^{(d-1)(d-2)/(d^2-d-1)}) = O(n^{(2d^2-4d+1)/(d^2-d-1)})$.

2. **Case** $r^{d-1} < (K/n^d)r^d$.

   In this case $q = (K/n^d)r^d$, therefore $nq/r = n(K/n^d)r^{d-1}$. As a consequence, the cost is:

$$n + n(K/n^d)r^{d-1} + n(K/n^d)r^{d-1} + n(K/n^d)r^{d-1} + n^{2d/(d+1)}q^{1/(d+1)}r^{-d/(d+1)}.$$

   The optimum is found by balancing:

$$
\begin{aligned}
&n(K/n^d)r^{d-1} = n^{2d/(d+1)}q^{1/(d+1)}r^{-d/(d+1)}\\
&n(K/n^d)r^{d-1} = n^{2d/(d+1)}[(K/n^d)r^d]^{1/(d+1)}r^{-d/(d+1)}\\
&n(K/n^d)r^{d-1} = n^{2d/(d+1)}(K/n^d)^{1/(d+1)}\\
&r^{d-1} = n^{(d-1)/(d+1)}(K/n^d)^{-d/(d+1)}.
\end{aligned}
$$

   The total cost is thus:

$$n(K/n^d)n^{(d-1)/(d+1)}(K/n^d)^{-d/(d+1)}) = (K/n^d)^{1/(d+1)}n^{2d/(d+1)} = K^{1/(d+1)}n^{d/(d+1)}$$

We keep in memory at any given time only one path from root to a leaf and we explore the partition tree in depth-first manner. Thus, the storage is only linear in $n$. Summarizing:

**Theorem 4** *Given $n$ points and $n$ $(d-1)$-simplices in $E^d$, we can determine whether any $(d-1)$-simplex is incident to any point in time $O(K^{(1/(d+1)}n^{d/(d+1)+\epsilon} + n^{f(d)+\epsilon})$, where*

$$f(d) = \frac{2d^2-4d+1}{d^2-d-1}$$

*and $K$ is the number of vertices in the arrangement of the $(d-1)$-simplices, $\epsilon > 0$ an arbitrarily small constant. The storage is linear in $n$.*

The highest gain over the non-sensitive method is attained for low-dimensional spaces. For $d = 2$, we have $f(2) = 1$; for $d = 3$ we have $f(3) = 7/5$.

# 7 Off-line simplex range searching

We present the algorithm for solving off-line simplex range searching queries as a modification of the algorithm in the previous section. At the leaves of the partition tree we need a non-sensitive method, which we describe next. Similarly to the incidence, the inclusion of a point in a $d$-simplex can be expressed as a Boolean formula involving only bilinear expressions and inequalities. By applying the theory in [Pel94a] and the deterministic cuttings in [Mat91], we can derive easily a method that uses for $n$ simplices and $m$ points time $O(m^{d/(d+1)+\epsilon}n^{d/(d+1)} + m^{1+\epsilon} + n\log^{d+2} m)$ and $O(n+m)$ storage.

We consider the algorithm in the previous section using the facets of the input $d$-simplices as the set of $(d-1)$-simplices. Also, for each cell $\sigma \in C_k$, we compute the list of $d$-simplices that contain the cell $\sigma$, but do not contain the parent of $\sigma$. It is easy to see that, if $s$ is the parent of $\sigma$, then such set of $d$-simplices must have facets in $Q(s) \cup P(s)$. Thus, the total length of such lists and the total time to compute them is asymptotically $O(nr^{d-2+\epsilon} + n(K/n^d)r^{d-1})$.

In the data structure so modified we can collect for each $d$-simplex the number of points contained in it. By using a depth first strategy we can keep the storage linear. This concludes the proof of Theorem 3.

# 8 Conclusions

We have shown a technique for decomposing arrangements of simplices and points. Using this technique we have obtained efficient algorithms to count the number of mono-chromatic and bi-chromatic intersections in sets of segments. Also, we have obtained an algorithm for off-line simplex range searching in $E^d$ that is sensitive to the sparsity of the set of queries.

One obvious open problem is that of extending this technique to problems involving triangles in 3-space, such as counting intersection of triangles. A second challenging open problem is whether it is possible to achieve a time bound for an *on-line* sequence of queries that is sensitive to the sparsity of the sequence.

# 9 Acknowledgments

I wish to thank an anonymous referee for many suggestions that improved the presentation of the results in this paper.

## References


[Aga90a] P.K. Agarwal. Partitioning arrangements of lines I: An efficient deterministic algorithm. *Discrete & Computational Geometry*, 5:449–483, 1990.

[Aga90b] P.K. Agarwal. Partitioning arrangements of lines II: Applications. *Discrete & Computational Geometry*, 5:533–573, 1990.

[AS90] P. K. Agarwal and M. Sharir. Red-blue intersection detection algorithms, with applications to motion planning and collision detection. *SIAM J. Comput.*, 19:297–321, 1990.

[BO79] J.L. Bentley and T. Ottman. Algorithms for reporting and counting geometric intersections. *IEEE Trans. on Computers*, C-28:643–647, 1979.

[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.

[CEGS94] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedra terrains. *Algorithmica*, 11:116–132, 1994.

[CF90] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[Cha86] B. Chazelle. Reporting and counting segment intersections. *J. Comput. Syst. Sci.*, 32:156–182, 1986.

[Cha89] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

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

[Cha95] B. Chazelle. Lower bounds for off-line range range searching. In *Proceedings of the 27th ACM Symposium on Theory of Computing*, 1995.

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

[CS89] K. Clarkson and P. Shor. Applications of random sampling in computational geometry II. *Discrete & Computational Geometry*, 4:387–422, 1989.

[CSW92] B. Chazelle, M. Sharir, and E. Welzl. Quasi-Optimal Upper Bounds for Simplex Range Searching and New Zone Theorems. *Algorithmica*, (8):407–429, 1992.

[CW89] B. Chazelle and E. Welzl. Quasi-optimal range searching in spaces of finite VC-dimension. *Discrete Comput. Geom.*, 4:467–489, 1989.

[CY85] R. Cole and C. K. Yap. Geometric retrieval problems. *Inform. Control*, 63:39–57, 1985.

[dBS92] M. de Berg and O. Schwarzkopf. Cuttings and applications. Technical Report TR CS-92-26, Department of Computer Science, Utrecht University, 1992.

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

[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.

[ES93] J. Erickson and R. Seidel. Better lower bounds on detecting affine and spherical degeneracies. In *Proceedings of the 34th Symposium on Foundations of Computer Science*, 1993. 528–536.

[ESS93] H. Edelsbrunner, R. Seidel, and M. Sharir. On the zone theorem for hyperplane arrangements. *SIAM J. on Computing*, 22(2):418–429, 1993.

[EW86] H. Edelsbrunner and E. Welzl. Halfplanar range search in linear space and $o(n^{0,695})$ query time. *Inform. Process. Lett.*, (23):289–293, 1986.

[FVFH90] J. D. Foley, A. Van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley, Reading, MA, 1990.

[GOS88] L. Guibas, M. Overmars, and M. Sharir. Intersecting line segments, ray shooting, and other applications of geometric partitioning techniques. In *Proc. 1st Scand. Workshop Algorithm Theory*, volume 318 of *Lecture Notes in Computer Science*, pages 64–73. Springer-Verlag, 1988.

[HW87] D. Haussler and E. Welzl. $\epsilon$ nets and simplex range queries. *Discrete Comput. Geom.*, (2):127–151, 1987.

[Kar92] R. Karp. On-line algorithms versus off-line algorithms: how much is it worth to know the future? Technical Report TR-92-044, International Computer Science Institute, 1992.

[Mal91] F. M. Maley. A generic algorithm for one-dimsnsional homotopic compactation. *Algorithmica*, 6:103–128, 1991.

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

[Mat92a] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[Mat92b] J. Matoušek. Range searching with efficient hierarchical cuttings. In *Proceedings of the 8th ACM Symposium on Computational Geometry*, pages 276–285, 1992.

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

[MS88] H. G. Mairson and J. Stolfi. Reporting and counting intersections between two sets of line segments. In R. A. Earnshaw, editor, *Theoretical Foundations of Computer Graphics and CAD*, volume F40 of *NATO ASI*, pages 307–325. Springer-Verlag, Berlin, West Germany, 1988.

[Mul88] K. Mulmuley. A fast planar partition algorithm, I. In *Proc. 29th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 580–589, 1988.

[NP82] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Commun. ACM*, 25:739–747, 1982.

[Pel94a] M. Pellegrini. On collision-free placements of simplices and the closest pair of lines in 3-space. *SIAM J. on Computing*, 23(1):133–153, 1994.

[Pel94b] M. Pellegrini. On Point Location and Motion Planning in Arrangements of Simplices. In *Proceedings of the 26th ACM Symposium on Theory of Computing*, pages 95–104, 1994. Journal version to appear in *SIAM J. on Computing*.

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

[PS91] J. Pach and M. Sharir. On vertical visibility in arrangements of segments and the queue size in the Bentley-Ottman line sweeping algorithm. *SIAM J. Comput.*, 20:460–470, 1991.

[SH76] M. I. Shamos and D. Hoey. Geometric intersection problems. In *Proc. 17th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 208–215, 1976.

[Wil82] D.E. Willard. Polygon retrieval. *SIAM Journal of Computing*, pages 149–165, 1982.

[YY85] A. C. Yao and F. F. Yao. A general approach to $D$-dimensional geometric queries. In *Proc. 17th Annu. ACM Sympos. Theory Comput.*, pages 163–168, 1985.