# Geometric Range Searching and Its Relatives*

Pankaj K. Agarwal    Jeff Erickson

Center for Geometric Computing
Department of Computer Science
Duke University, Box 90129
Durham, NC 27708-0129
{pankaj, jeffe}@cs.duke.edu
http://www.cs.duke.edu/~{pankaj, jeffe}

September 2, 1997

## 1 Introduction

About ten years ago, the field of range searching, especially simplex range searching, was wide open. At that time, neither efficient algorithms nor nontrivial lower bounds were known for most range-searching problems. A series of papers by Haussler and Welzl [161], Clarkson [88, 89], and Clarkson and Shor [92] not only marked the beginning of a new chapter in geometric searching, but also revitalized computational geometry as a whole. Led by these and a number of subsequent papers, tremendous progress has been made in geometric range searching, both in terms of developing efficient data structures and proving nontrivial lower bounds. From a theoretical point of view, range searching is now almost completely solved. The impact of general techniques developed for geometric range searching — $\varepsilon$-nets, $1/r$-cuttings, partition trees, multi-level data structures, to name a few — is evident throughout computational geometry. This volume provides an excellent opportunity to recapitulate the current status of geometric range searching and to summarize the recent progress in this area.

Range searching arises in a wide range of applications, including geographic information systems, computer graphics, spatial databases, and time-series databases. Furthermore, a variety of geometric problems can be formulated as a range-searching problem. A typical range-searching problem has the following form. Let $S$ be a set of $n$ points in $\mathbb{R}^d$, and let

*Pankaj Agarwal's work on this paper was supported by National Science Foundation Grant CCR-93-01259, by Army Research Office MURI grant DAAH04-96-1-0013, by a Sloan fellowship, by an NYI award and matching funds from Xerox Corporation, and by a grant from the U.S.-Israeli Binational Science Foundation. Jeff Erickson's work was supported by National Science Foundation grant DMS-9627683 and by Army Research Office MURI grant DAAH04-96-1-0013.

$\mathcal{R}$ be a family of subsets of $\mathbb{R}^d$; elements of $\mathcal{R}$ are called *ranges*. We wish to preprocess $S$ into a data structure so that for a query range $\gamma \in \mathcal{R}$, the points in $S \cap \gamma$ can be reported or counted efficiently. Typical examples of ranges include rectangles, halfspaces, simplices, and balls. If we are only interested in answering a single query, it can be done in linear time, using linear space, by simply checking for each point $p \in S$ whether $p$ lies in the query range. Most applications, however, call for querying the same point set $S$ several times (and sometimes we also insert or delete a point periodically), in which case we would like to answer a query faster by preprocessing $S$ into a data structure.

Range counting and range reporting are just two instances of range-searching queries. Other examples include *emptiness queries*, where one wants to determine whether $S \cap \gamma = \emptyset$, and *optimization queries*, where one wants to choose a point with certain property (*e.g.*, a point in $\gamma$ with the largest $x_1$-coordinate). In order to encompass all different types of range-searching queries, a general range-searching problem can be defined as follows.

Let $(\mathbf{S}, +)$ be a commutative semigroup[1]. For each point $p \in S$, we assign a weight $w(p) \in \mathbf{S}$. For any subset $S' \subseteq S$, let $w(S') = \sum_{p \in S'} w(S)$, where addition is taken over the semigroup.[2] For a query range $\gamma \in \mathcal{R}$, we wish to compute $w(S \cap \gamma)$. For example, counting queries can be answered by choosing the semigroup to be $(\mathbb{Z}, +)$, where $+$ denotes standard integer addition, and setting $w(p) = 1$ for every $p \in S$; emptiness queries by choosing the semigroup to be $(\{0,1\}, \vee)$ and setting $w(p) = 1$; reporting queries by choosing the semigroup to be $(2^S, \cup)$ and setting $w(p) = \{p\}$; and optimization queries by choosing the semigroup to be $(\mathbb{R}, \max)$ and choosing $w(p)$ to be, for example, the $x_1$-coordinate of $p$.

We can, in fact, define a more general (decomposable) *geometric searching* problem. Let $S$ be a set of *objects* in $\mathbb{R}^d$ (*e.g.*, points, hyperplanes, balls, or simplices), $(\mathbf{S}, +)$ a commutative semigroup, $w : S \to \mathbf{S}$ a weight function, $\mathcal{R}$ a set of ranges, and $\Diamond \subseteq S \times \mathcal{R}$ a "spatial" relation between objects and ranges. Then for a range $\gamma \in \mathcal{R}$, we want to compute $\sum_{p \Diamond \gamma} w(p)$. Range searching is a special case of this general searching problem, in which $S$ is a set of points in $\mathbb{R}^d$ and $\Diamond = \in$. Another widely studied searching problem is *intersection searching*, where $p \Diamond \gamma$ if $p$ intersects $\gamma$. As we will see below, range-searching data structures are useful for many other geometric searching problems.

The performance of a data structure is measured by the time spent in answering a query, called the *query time*, by the *size* of the data structure, and by the time constructed in the data structure, called the *preprocessing time*. Since the data structure is constructed only once, its query time and size are generally more important than its preprocessing time. If a data structure supports insertion and deletion operations, its *update time* is also relevant. We should remark that the query time of a range-reporting query on any reasonable machine depends on the output size, so the query time for a range-reporting query consists of two parts — *search time*, which depends only on $n$ and $d$, and *reporting time*, which depends on $n$, $d$, and the output size. Throughout this survey paper we will use $k$ to denote the output size.

[1]A *semigroup* $(\mathbf{S}, +)$ is a set $\mathbf{S}$ equipped with an associative addition operator $+ : \mathbf{S} \times \mathbf{S} \to \mathbf{S}$. A semigroup is *commutative* if $x + y = y + x$ for all $x, y \in \mathbf{S}$.

[2]Since $\mathbf{S}$ need not have an additive identity, we may need to assign a special value *nil* to the empty sum.

We assume that $d$ is a small fixed constant, and that big-Oh and big-Omega notation hides constants depending on $d$. The dependence on $d$ of the performance of almost all the data structures mentioned in this survey is exponential, which makes them unsuitable in practice for large values of $d$.

The size of any range-searching data structure is at least linear, since it has to store each point (or its weight) at least once, and the query time in any reasonable model of computation such as pointer machines, RAMs, or algebraic decision trees is $\Omega(\log n)$ even when $d = 1$. Therefore, we would like to develop a linear-size data structure with logarithmic query time. Although near-linear-size data structures are known for orthogonal range searching in any fixed dimension that can answer a query in polylogarithmic time, no similar bounds are known for range searching with more complex ranges such as simplices or disks. In such cases, we seek a tradeoff between the query time and the size of the data structure — How fast can a query be answered using $O(n \operatorname{polylog} n)$ space, how much space is required to answer a query in $O(\operatorname{polylog} n)$ time, and what kind of tradeoff between the size and the query time can be achieved?

In this paper we survey the known techniques and data structures for range-searching problems and describe their applications to other related searching problems. As mentioned in the beginning, the quest for efficient range-searching data structure has led to many general, powerful techniques that have had a significant impact on several other geometric problems. The emphasis of this survey is on describing known results and general techniques developed for range searching, rather than on open problems. The paper is organized as follows. We describe, in Section 2, different models of computation that have been used to prove upper and lower bounds on the performance of data structures. Next, in Section 3, we review data structures for orthogonal range searching and its variants. Section 4 surveys known techniques and data structures for simplex range searching, and Section 5 discusses some variants and extensions of simplex range searching. Finally, we review data structures for intersection searching and optimization queries in Sections 6 and 7, respectively.

## 2 Models of Computation

Most algorithms and data structures in computational geometry are implicitly described in the familiar *random access machine* (RAM) model, described in [17], or the *real RAM* model described by Preparata and Shamos [242]. In the traditional RAM model, memory cells can contain arbitrary $(\log n)$-bit integers, which can be added, multiplied, subtracted, divided (computing $\lfloor x/y \rfloor$), compared, and used as pointers to other memory cells in constant time. A few algorithms rely on a variant of the RAM model, proposed by Fredman and Willard [129], that allows memory cells to contain $w$-bit integers, for some parameter $w \geq \log n$, and permits both arithmetic and bitwise logical operations in constant time. In a real RAM, we also allow memory cells to store arbitrary real numbers (such as coordinates of points). We allow constant-time arithmetic on and comparisons between real numbers, but we do not allow conversion between integers and reals. In the case of range searching over a semigroup other than the integers, we also allow memory cells to contain arbitrary values from the semigroup, but these values can only be added (using the semigroup's addition operator, of course).

Almost all known range-searching data structures can be described in the more restrictive *pointer machine* model, originally developed by Tarjan [271].[3] The main difference between the two models is that on a pointer machine, a memory cell can be accessed only through a series of pointers, while in the RAM model, any memory cell can be accessed in constant time. Tarjan's basic pointer machine model is most suitable for studying range-reporting problems. In this model, a data structure is a directed graph with outdegree 2. To each node $v$ in this graph, we associate a label $\ell(v)$, which is an integer between 0 and $n$. Nonzero labels are indices of the points in $S$. The query algorithm, given a range $\gamma$, begins at a special starting node and performs a sequence of the following operations: (1) visit a new node by traversing an edge from a previously visited node, (2) create a new node $v$ with $\ell(v) = 0$, whose outgoing edges point to previously visited nodes, and (3) redirect an edge leaving a previously visited node, so that it points to another previously visited node. When the query algorithm terminates, the set of visited nodes $W(\gamma)$, called the *working set*, is required to contain the indices of all points in the query range; that is, if $p_i \in \gamma$, then there must be a node $v \in W(\gamma)$ such that $\ell(v) = i$. The working set $W(\gamma)$ may contain labels of points that are not in the query range. The size of the data structure is the number of nodes in the graph, and the query time for a range $\gamma$ is the size of the smallest possible working set $W(\gamma)$. The query time ignores the cost of other operations, including the cost of deciding which edges to traverse. There is no notion of preprocessing or update time in this model. Note that the model accommodates both static and self-adjusting data structures.

Chazelle [58] defines several generalizations of the pointer-machine model that are more appropriate for answering counting and semigroup queries. In Chazelle's generalized pointer-machine models, nodes are labeled with arbitrary $O(\log n)$-bit integers. In addition to traversing edges in the graph, the query algorithm is also allowed to perform various arithmetic operations on these integers. An *elementary* pointer machine can add and compare integers; in an *arithmetic* pointer machine, subtraction, multiplication, integer division, and shifting ($x \mapsto 2^x$) are also allowed. When the query algorithm terminates in these models, some node in the working set is required to contain the answer. If the points have weights from an additive semigroup other than the integers, nodes in the data structure can also be labeled with semigroup values, but these values can only be added.

Most lower bounds, and a few upper bounds, are described in the so-called *semigroup arithmetic model*, which was originally introduced by Fredman [133] and refined by Yao [292]. In the semigroup arithmetic model, a data structure can be *informally* regarded as a set of precomputed partial sums in the underlying semigroup. The size of the data structure is the number of sums stored, and the query time is the minimum number of semigroup operations required (on the precomputed sums) to compute the answer to a query. The query time ignores the cost of various auxiliary operations, including the cost

[3]Several very different models of computation with the name "pointer machine" have been proposed; these are surveyed by Ben-Amram [39], who suggests the less ambiguous term *pointer algorithm* for the model we describe.

of determining which of the precomputed sums should be added to answer a query. Unlike the pointer-machine model, the semigroup model allows immediate access, at no cost, to any precomputed sum.

The informal model we have just described is much too powerful. For example, in this informal model, the optimal data structure for counting queries consists of the $n+1$ integers $0, 1, \ldots, n$. To answer a counting query, we simply return the correct answer; since no additions are required, we can answer queries in zero "time", using a "data structure" of only linear size!

Here is a more formal definition that avoids this problem. Let $(\mathbf{S}, +)$ be a commutative semigroup. A *linear form* is a sum of variables over the semigroup, where each variable can occur multiple times, or equivalently, a homogeneous linear polynomial with positive integer coefficients. The semigroup is *faithful* if any two identically equal linear forms have the same set of variables, although not necessarily with the same set of coefficients.[4] For example, the semigroups $(\mathbb{Z}, +)$, $(\mathbb{R}, \min)$, $(\mathbb{N}, \gcd)$, and $(\{0,1\}, \vee)$ are faithful, but the semigroup $(\{0,1\}, + \bmod 2)$ is not faithful.

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of objects, $\mathbf{S}$ a faithful semigroup, $\mathcal{R}$ a set of ranges, and $\Diamond$ a relation between objects and ranges. (Recall that in the standard range-searching problem, the objects in $S$ are points, and $\Diamond$ is containment.) Let $x_1, x_2, \ldots, x_n$ be a set of $n$ variables over $\mathbf{S}$, each corresponding to a point in $S$. A *generator* $g(x_1, \ldots, x_n)$ is a linear form $\sum_{i=1}^{n} \alpha_i x_i$, where the $\alpha_i$'s are non-negative integers, not all zero. (In practice, the coefficients $\alpha_i$ are either 0 or 1.) A *storage scheme* for $(S, \mathbf{S}, \mathcal{R}, \Diamond)$ is a collection of generators $\{g_1, g_2, \ldots, g_s\}$ with the following property: For any query range $\gamma \in \mathcal{R}$, there is an set of indices $I_\gamma \subseteq \{1, 2, \ldots, s\}$ and a set of labeled nonnegative integers $\{\beta_i \mid i \in I_\gamma\}$ such that the linear forms

$$\sum_{p_i \Diamond \gamma} x_i \quad \text{and} \quad \sum_{i \in I_\gamma} \beta_i g_i$$

are identically equal. In other words, the equation

$$\sum_{p_i \Diamond \gamma} w(p_i) = \sum_{i \in I_\gamma} \beta_i g_i(w(p_1), w(p_2), \ldots, w(p_n))$$

holds for *any* weight function $w : S \to \mathbf{S}$. (Again, in practice, $\beta_i = 1$ for all $i \in I_\gamma$.) The size of the smallest such set $I_\gamma$ is the query time for $\gamma$; the time to actually choose the indices $I_\gamma$ is ignored. The space used by the storage scheme is measured by the number of generators. There is no notion of preprocessing time in this model.

[4] More formally, $(\mathbf{S}, +)$ is faithful if for each $n > 0$, for any sets of indices $I, J \subseteq \{1, \ldots, n\}$ so that $I \neq J$, and for every sequence of positive integers $\alpha_i, \beta_j$ $(i \in I, j \in J)$, there are semigroup values $s_1, s_2, \ldots, s_n \in \mathbf{S}$ such that

$$\sum_{i \in I} \alpha_i s_i \neq \sum_{j \in J} \beta_j s_j.$$

We emphasize that although a storage scheme can take advantage of special properties of the set $S$ or the semigroup $\mathbf{S}$, it must work for *any* assignment of weights to $S$. In particular, this implies that lower bounds in the semigroup model do not apply to the problem of counting the number of points in the query range, even though $(\mathbb{N}, +)$ is a faithful semigroup, since a storage scheme for the counting problem only needs to work for the particular weight function $w(p) = 1$ for all $p \in S$. Similar arguments apply to emptiness, reporting, and optimization queries, even though the semigroups $(\{0,1\}, \vee)$, $(2^S, \cup)$, and $(\mathbb{R}, \min)$ are all faithful.

The requirement that the storage scheme must work for any weight assignment even allows us to model problems where the weights depend on the query. For example, suppose for some set $S$ of objects with real weights, we have a storage scheme that lets us quickly determine the minimum weight of any object hit by a query ray. In other words, we have a storage scheme for $S$ under the semigroup $(\mathbb{R}, \min)$ that supports intersection searching, where the query ranges are rays. We can use such a storage scheme to answer ray-shooting queries, by letting the weight of each object be its distance along the query ray from the basepoint. If we want the first object hit by the query ray instead of just its distance, we can use the faithful semigroup $(S \times \mathbb{R}, \diamond)$, where

$$(p_1, \delta_1) \diamond (p_2, \delta_2) = \begin{cases} (p_1, \delta_1) & \text{if } \delta_1 \leq \delta_2, \\ (p_2, \delta_2) & \text{otherwise,} \end{cases}$$

and letting the weight of an object $p \in S$ be $(p, \delta)$, where $\delta$ is the distance along the query ray between the basepoint and $p$. We reiterate, however, that lower bounds in the semigroup model do not imply lower bounds on the complexity of ray shooting.

Although in principle, storage schemes can exploit of special properties of the semigroup $\mathbf{S}$, in practice, they never do. All known upper and lower bounds in the semigroup arithmetic model hold for all faithful semigroups. In other models of computation where semigroup values can be manipulated, such as RAMs and elementary pointer machines, slightly better upper bounds are known for some problems when the semigroup is $(\mathbb{N}, +)$.

The semigroup model is formulated slightly differently for offline range-searching problems. Here we are given a set of weighted points $S$ and a finite set of query ranges $\mathcal{R}$, and we want to compute the total weight of the points in each query range. This is equivalent to computing the product $Aw$, where $A$ is the incidence matrix of the points and ranges, and $w$ is the vector of weights. In the offline semigroup model, introduced by Chazelle [65], an algorithm can be described as a circuit (or straight-line program) with one input for every point and one output for every query range, where every gate (respectively, statement) performs a binary semigroup addition. The running time of the algorithm is the total number of gates (respectively, statements). For any weight function $w : S \to \mathbf{S}$, the output associated with a query range $\gamma$ is $w(S \cap \gamma)$. Just as in the online case, the circuit is required to work for any assignment of weights to the points; in effect, the outputs of the circuit are the linear forms $\sum_{p_i \in \gamma} x_i$. See Figure 1 for an example.

A serious weakness of the semigroup model is that it does not allow subtractions even if the weights of points belong to a group. Therefore, we will also consider the *group model*,

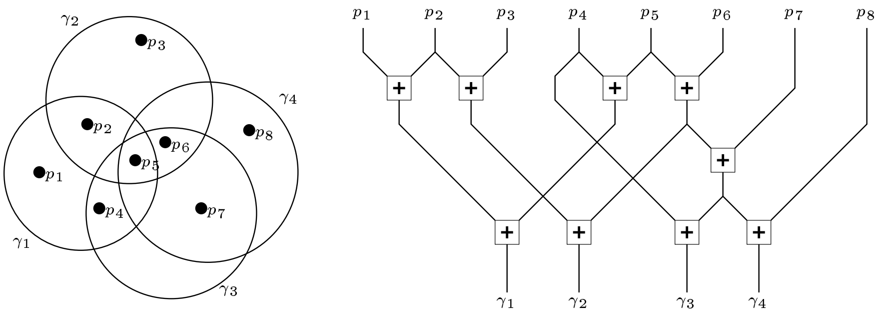


**Figure 1.** A set of eight points and four disks, and an offline semigroup arithmetic algorithm to compute the total weight of the points in each disk.

in which both additions and subtractions are allowed [287, 64, 65]. Chazelle [65] considers an extension of the offline group model in which circuits are allowed a limited number of *help gates*, which can compute arbitrary binary functions.

Of course it is natural to consider arithmetic circuits which also allow multiplication ("the ring model"), division ("the field model"), or even more general functions such as square roots or exponentiation. There is a substantial body of literature on the complexity of various types of arithmetic circuits [278, 269, 53], but almost nothing is known about the complexity of geometric range searching in these models. Perhaps the only relevant result is that any circuit with operations $+, -, \times, \div, \sqrt{\ }$ requires $\Omega(\log n)$ time to answer any reasonable range query, or $\Omega(n \log n)$ time to solve any reasonable offline range searching problem, since such a circuit can be modeled as an algebraic computation tree with no branches [40] or as a straight-line program on a real RAM [38]. (Computation trees with more general functions are considered in [147].)

Almost all geometric range-searching data structures are constructed by subdividing space into several regions with nice properties and recursively constructing a data structure for each region. Range queries are answered with such a data structure by performing a depth-first search through the resulting recursive space partition. The *partition graph* model, recently introduced by Erickson [117, 118, 119], formalizes this divide-and-conquer approach, at least for hyperplane and halfspace range searching data structures. The partition graph model can be used to study the complexity of emptiness queries, unlike the semigroup arithmetic and pointer machine models, in which such queries are trivial.

Formally, a partition graph is a directed acyclic graph with constant outdegree, with a single source, called the root, and several sinks, called leaves. Associated with each internal node is a cover of $\mathbb{R}^d$ by a constant number of connected subsets called *query regions*, each associated with an outgoing edge. Each internal node is also labeled either *primal* or *dual*, indicating whether the query regions should be considered a decomposition of "primal" or "dual" space. (Point-hyperplane duality is discussed in Section 4.2.) Any partition graph defines a natural search structure, which is used both to preprocess a set of points and to perform a query for a hyperplane or halfspace. The points are preprocessed one at a time. To preprocess a single point, we perform a depth-first search of the graph starting at the root. At each primal node, we traverse the outgoing edges corresponding to the query regions that contain the point; at each dual node, we traverse the edges whose query regions intersect the point's dual hyperplane. For each leaf $\ell$ of the partition graph, we maintain a set $P_\ell$ containing the points that reach $\ell$ during the preprocessing phase. The query algorithm for hyperplanes is an exactly symmetric depth-first search — at primal nodes, we look for query regions that intersect the hyperplane, and at dual nodes, we look for query regions that contain its dual point. The answer to a query is determined by the sets $P_\ell$ associated with the leaves $\ell$ of the partition graph that the query algorithm reaches. For example, the output of an emptiness query is "yes" (*i.e.*, the query hyperplane contains none of the points) if and only if $P_\ell = \emptyset$ for every leaf $\ell$ reached by the query algorithm. The size of the partition graph is the number of edges in the graph; the complexity of the query regions and the sizes of the sets $P_\ell$ are not considered. The preprocessing time for a single point and the query time for a hyperplane are given by the number of edges traversed during the search; the time required to actually construct the partition graph and to test the query regions is ignored.

We conclude this section by noting that most of the range-searching data structures discussed in this paper (halfspace range-reporting data structures being a notable exception) are based on the following general scheme. Given a point set $S$, they precompute a family $\mathcal{F} = \mathcal{F}(S)$ of *canonical subsets* of $S$ and store the weight $w(C) = \sum_{p \in C} w(p)$ of each canonical subset $C \in \mathcal{F}$. For a query range $\gamma$, they determine a partition $\mathcal{C}_\gamma = \mathcal{C}(S, \gamma) \subseteq \mathcal{F}$ of $S \cap \gamma$ and add the weights of the subsets in $\mathcal{C}_\gamma$ to compute $w(S \cap \gamma)$. Borrowing terminology from [203], we will refer to such a data structure as a *decomposition scheme*.

There is a close connection between decomposition schemes and storage schemes in the semigroup arithmetic model described earlier. Each canonical subset $C = \{p_i \mid i \in I\} \in \mathcal{F}$, where $I \subseteq \{1, 2, \ldots, n\}$, corresponds to the generator $\sum_{i \in I} x_i$. In fact, because the points in any query range are always computed as the *disjoint* union of canonical subsets, any decomposition scheme corresponds to a storage scheme that is valid for *any* semigroup. Conversely, lower bounds in the semigroup model imply lower bounds on the complexity of any decomposition scheme.

How exactly the weights of canonical subsets are stored and how $\mathcal{C}_\gamma$ is computed depends on the model of computation and on the specific range-searching problem. In the semigroup (or group) arithmetic model, the query time depends only on the number of canonical subsets in $\mathcal{C}_\gamma$, regardless of how they are computed, so the weights of canonical subsets can be stored in an arbitrary manner. In more realistic models of computation, however, some additional structure must be imposed on the decomposition scheme in order to efficiently compute $\mathcal{C}_\gamma$. In a *hierarchical* decomposition scheme, the weights are stored in a tree $T$. Each node $v$ of $T$ is associated with a canonical subset $C_v \in \mathcal{F}$, and the children of $v$ are associated with subsets of $C_v$. Besides the weight of $C_v$, some auxiliary information is also stored at $v$, which is used to determine whether $C_v \in \mathcal{C}_\gamma$ for a query range $\gamma$. Typically, this auxiliary information consists of some geometric object, which plays the same role as a query region in the partition graph model.

If the weight of each canonical subset can be stored in $O(1)$ memory cells, then the total size of the data structure is just $O(|\mathcal{F}|)$. If the underlying searching problem is a range-reporting problem, however, then the "weight" of a canonical subset is the set itself, and thus it is not realistic to assume that each "weight" requires only constant space. In this case, the size of the data structure is $O(\sum_{C\in\mathcal{F}}|C|)$ if each subset is stored explicitly at each node of the tree. As we will see below, the size can be reduced to $O(|\mathcal{F}|)$ by storing the subsets implicitly (*e.g.*, storing points only at leaves).

To determine the points in a query range $\gamma$, a query procedure performs a depth-first search of the tree $T$, starting from the root. At each node $v$, using the auxiliary information stored at $v$, the procedure determines whether the query range $\gamma$ contains $C_v$, intersects $C_v$, or is disjoint from $C_v$. If $\gamma$ contains $C_v$, then $C_v$ is added to $\mathcal{C}_\gamma$ (rather, the weight of $C_v$ is added to a running counter). Otherwise, if $\gamma$ intersects $C_v$, the query procedure identifies a subset of children of $v$, say $\{w_1, \ldots, w_a\}$, so that the canonical subsets $C_{w_i} \cap \gamma$, for $1 \le i \le a$, form a partition of $C_v \cap \gamma$. Then the procedure searches each $w_i$ recursively. The total query time is $O(\log n + |\mathcal{C}_\gamma|)$, provided constant time is spent at each node visited.

## 3 Orthogonal Range Searching

In $d$-dimensional orthogonal range searching, the ranges are $d$-rectangles, each of the form $\prod_{i=1}^{d}[a_i, b_i]$, where $a_i, b_i \in \mathbb{R}$. This is an abstraction of multi-key searching [43, 289], which is a central problem in statistical and commercial databases. For example, the points of $S$ may correspond to employees of a company, each coordinate corresponding to a key such as age, salary, or experience. Queries such as "Report all employees between the ages of 30 and 40 who earn more than $\$30,000$ and who have worked for more than 5 years" can be formulated as orthogonal range-reporting queries. Because of its numerous applications, orthogonal range searching has been studied extensively for the last 25 years. A survey of earlier results can be found in the books by Mehlhorn [211] and Preparata and Shamos [242]. In this section we review more recent data structures and lower bounds.

### 3.1 Upper bounds

Most of the recent orthogonal range-searching data structures are based on *range trees*, introduced by Bentley [42]. For $d = 1$, the range tree of $S$ is either a minimum-height binary search tree on $S$ or an array storing $S$ in sorted order. For $d > 1$, the range tree of $S$ is a minimum-height binary tree $T$ with $n$ leaves, whose $i$th leftmost leaf stores the point of $S$ with the $i$th smallest $x_1$-coordinate. To each interior node $v$ of $T$, we associate a canonical subset $C_v \subseteq S$ containing the points stored at leaves in the subtree rooted at $v$. For each $v$, let $a_v$ (resp. $b_v$) be the smallest (resp. largest) $x_1$-coordinate of any point in $C_v$, and let $C_v^*$ denote the projection of $C_v$ onto the hyperplane $x_1 = 0$. The interior node $v$ stores $a_v$, $b_v$, and a $(d-1)$-dimensional range tree constructed on $C_v^*$. For any fixed dimension $d$, the size of the overall data structure is $O(n \log^{d-1} n)$, and it can be constructed in time $O(n \log^{d-1} n)$. The range-reporting query for a rectangle $\gamma = \prod_{i=1}^{d}[a_i, b_i]$ can be answered as follows. If $d = 1$, the query can be answered by a binary search. For $d > 1$, we traverse the range tree as follows. Suppose we are at a node $v$. If $v$ is a leaf, then we report its corresponding point if it lies inside $\gamma$. If $v$ is an interior node and the interval $[a_v, b_v]$ does not intersect $[a_1, b_1]$, there is nothing to do. If $[a_v, b_v] \subseteq [a_1, b_1]$, we recursively search in the $(d-1)$-dimensional range tree stored at $v$, with the rectangle $\prod_{i=2}^{d}[a_i, b_i]$. Otherwise, we recursively visit both children of $v$. The query time of this procedure is $O(\log^d n + k)$, which can be improved to $O(\log^{d-1} n + k)$ using the *fractional-cascading* technique [76, 196]. A range tree can also answer a range-counting query in time $O(\log^{d-1} n)$. Range trees are an example of a multi-level data structure, which we will discuss in more detail in Section 5.1.

The best-known data structures for orthogonal range searching are by Chazelle [55, 58], who used compressed range trees and other techniques to improve the storage and query time. His results in the plane, under various models of computation, are summarized in Table 1; the preprocessing time of each data structure is $O(n \log n)$. If the query rectangles are "three-sided rectangles" of the form $[a_1, b_1] \times [a_2, \infty]$, then one can use a *priority search tree* of size $O(n)$ to answer a planar range-reporting query in time $O(\log n + k)$ [208].

| **Problem** | **Model** | **Size** | **Query time** |
|---|---|---|---|
| Counting | RAM | $n$ | $\log n$ |
| | APM | $n$ | $\log n$ |
| | EPM | $n$ | $\log^2 n$ |
| Reporting | RAM | $n$ | $\log n + k \log^{\varepsilon}(2n/k)$ |
| | | $n \log\log n$ | $\log n + k \log\log(4n/k)$ |
| | | $n \log^{\varepsilon} n$ | $\log n + k$ |
| | APM | $n$ | $k \log(2n/k)$ |
| | EPM | $n$ | $k \log^2(2n/k)$ |
| | | $\dfrac{n \log n}{\log\log n}$ | $\log n + k$ |
| Semigroup | Semigroup | $m$ | $\dfrac{\log n}{\log(2m/n)}$ |
| | RAM | $n$ | $\log^{2+\varepsilon} n$ |
| | | $n \log\log n$ | $\log^2 n \log\log n$ |
| | | $n \log^{\varepsilon} n$ | $\log^2 n$ |
| | APM | $n$ | $\log^3 n$ |
| | EPM | $n$ | $\log^4 n$ |

**Table 1.** Asymptotic upper bounds for planar orthogonal range searching, due to Chazelle [55, 58], in the random access machine (RAM), arithmetic pointer machine (APM), elementary pointer machine (EPM), and semigroup arithmetic models.

Each of the two-dimensional results in Table 1 can be extended to queries in $\mathbb{R}^d$ at a cost of an additional $\log^{d-2} n$ factor in the preprocessing time, storage, and query-search time. For $d \geq 3$, Subramanian and Ramaswamy [270] have proposed a data structure that can answer a range-reporting query in time $O(\log^{d-2} n \log^* n + k)$ using $O(n \log^{d-1} n)$ space, and Bozanis *et al.* [51] have proposed an a data structure with $O(n \log^d n)$ size and $O(\log^{d-2} n + k)$ query time. The query time (or the query-search time in the range-reporting case) can be reduced to $O((\log n / \log\log n)^{d-1})$ in the RAM model by increasing the space to $O(n \log^{d-1+\varepsilon} n)$. In the semigroup arithmetic model, a query can be answered in time $O((\log n / \log(m/n))^{d-1})$ using a data structure of size $m$, for any $m = \Omega(n \log^{d-1+\varepsilon} n)$ [61].

Willard [288] proposed a data structure of size $O(n \log^{d-1} n / \log\log n)$, based on fusion trees, that can answer an orthogonal range-reporting query in time $O(\log^{d-1} n / \log\log n + k)$. Fusion trees were introduced by Fredman and Willard [129] for an $O(n\sqrt{\log n})$ sorting algorithm in a RAM model that allows bitwise logical operations.

Overmars [231] showed that if $S$ is a subset of a $u \times u$ grid $U$ in the plane and the vertices of query rectangles are also a subset of $U$, then a range-reporting query can be answered in time $O(\sqrt{\log u} + k)$, using $O(n \log n)$ storage and preprocessing, or in $O(\log\log u + k)$ time, using $O(n \log n)$ storage and $O(u^3 \log u)$ preprocessing. See [183] for some other results on range-searching for points on integer grids.

Orthogonal range-searching data structures based on range tress can be extended to handle $c$-oriented ranges in a straightforward manner. The performance of such a data structure is the same as that of a $c$-dimensional orthogonal range-searching structure. If the ranges are homothets of a given triangle, or translates of a convex polygon with constant number of edges, a two-dimensional range-reporting query can be answered in $O(\log n + k)$ time using linear space [67, 68]. If the ranges are octants in $\mathbb{R}^3$, a range-reporting query can be answered in either $O((k+1)\log n)$ or $O(\log^2 n + k)$ time using linear space [68].

### 3.2 Lower bounds

Fredman [131, 132, 133, 135] was the first to prove nontrivial lower bounds on orthogonal range searching, in a version of semigroup arithmetic model in which the points can be inserted and deleted dynamically. He showed that a mixed sequence of $n$ insertions, deletions, and queries requires $\Omega(n \log^d n)$ time. These bounds were extended by Willard [287] to the group model, under some fairly restrictive assumptions.

Yao [292] proved a lower bound for two-dimensional static data structures in the semigroup arithmetic model. He showed that if only $m$ units of storage is available, a query takes $\Omega(\log n / \log((m/n)\log n))$ in the worst case. Vaidya [272] proved lower bounds for orthogonal range searching in higher dimensions, which were later improved by Chazelle [61]. In particular, Chazelle proved the following strong result about the *average*-case complexity of orthogonal range searching:

**Theorem 1** (Chazelle [61]). *Let $d, n, m$ be positive integers with $m \geq n$. If only $m$ units of storage are available, then the expected query time for a random orthogonal range query in a random set of $n$ points in the unit hypercube $[0,1]^d$ is $\Omega((\log n / \log(2m/n))^{d-1})$ in the semigroup arithmetic model.*

A rather surprising result of Chazelle [60] shows that any data structure on a basic pointer machine that answers a $d$-dimensional range-reporting query in $O(\text{polylog}\, n + k)$ time must have size $\Omega(n(\log n / \log\log n)^{d-1})$; see also [18]. Notice that this lower bound is greater than the $O(n \log^{d-2+\varepsilon} n)$ upper bound in the RAM model (see Table 1).

These lower bounds do not hold for offline orthogonal range searching, where given a set of $n$ weighted points in $\mathbb{R}^d$ and a set of $n$ rectangles, one wants to compute the weight of the points in each rectangle. Recently, Chazelle [65] proved that the offline version takes

$\Omega(n(\log n/\log\log n)^{d-1})$ time in the semigroup model, and $\Omega(n \log\log n)$ time in the group model. An $\Omega(n \log n)$ lower bound also holds in the algebraic decision tree and algebraic computation tree models [267, 40].

## 3.3 Secondary memory structures

If the input point set is rather large and does not fit into main memory, then the data structure must be stored in secondary memory — on disk, for example — and portions of it must moved into main memory when needed to answer a query. In this case the bottleneck is the time spent in transferring data between main and secondary memory. We assume that data is stored in secondary memory in blocks of size $B$, where $B$ is a parameter. Each access to the secondary memory transfers one block (*i.e.*, $B$ words), and we count this as one input/output (I/O) operation. The size of a data structure is the number of blocks required to store it, and the query (resp. preprocessing) time is defined as the number of I/O operations required to answer a query (resp. to construct the structure). To simplify our notation, let $N = n/B$, the number of blocks required to hold the input, and let $\operatorname{Log} n = \log_B n$. Under this model, the size of any data structure is at least $N$, and the query time is at least $\operatorname{Log} n$. I/O-efficient orthogonal range-searching structures have received much attention recently, but most of the results are known only for the planar case. The main idea underlying these structures is to construct high-degree trees instead of binary trees. For example, variants of B-trees are used to answer 1-dimensional range-searching queries [35, 96]. A number of additional tricks are developed to optimize the size and the query time. See [20, 21, 232] for I/O efficient data structures that have been used for answering range searching and related queries.

Table 2 summarizes the known results on secondary-memory structures for orthogonal range searching. The data structure by Subramanian and Ramaswamy [270] for 3-sided queries supports insertion/deletion of a point in time $O(\operatorname{Log} n + (\operatorname{Log}^2 n)/B)$. Using the argument by Chazelle [60], they proved that any secondary-memory data structure that answers a range-reporting query using $O(\operatorname{polyLog} n + k/B)$ I/O operations requires $\Omega(N \log N/\log \operatorname{Log} n)$ storage. Hellerstein *et al.* [162] have shown that if a data structure for two-dimensional range query uses at most $O(N)$ disk blocks for a constant $r \geq 1$, then a query requires at least $\Omega((k/B)\sqrt{\log B/\log\log B})$ disk accesses; this extends an earlier lower bound by Kanellakis *et al.* [180].

## 3.4 Practical data structures

None of the data structures described in Section 3.1 are used in practice, even in two dimensions, because of the polylogarithmic overhead in the size and the query time. In many real applications, the input is too large to be stored in the main memory, so the number of disk accesses is a major concern. On the other hand, the range-searching data structures described in Section 3.3 are not simple enough to be of practical use for $d \geq 2$. For a data structure to be used in real applications, its size should be at most $cn$, where $c$ is a very small constant, the time to answer a typical query should be small — the lower

| $d$ | Range | Size | Query Time | Source |
|---|---|---|---|---|
| $d = 1$ | Interval | $N$ | $\text{Log}\, n + k/B$ | [35, 96] |
| $d = 2$ | Quadrant | $N \log\log B$ | $\text{Log}\, n + k/B$ | [244] |
| | 3-sided rectangle | $N$ | $\text{Log}\, n + k/B + \log^* B$ | [270] |
| | 3-sided rectangle | $N \log B \log\log B$ | $\text{Log}\, n + k/B$ | [244] |
| | Rectangle | $N \log N / \log \text{Log}\, n$ | $\text{Log}\, n + k/B + \log^* B$ | [270] |
| | Rectangle | $cN$ | $k/B^{1-1/2c}$ | [270] |
| $d = 3$ | Octant | $N \log N$ | $\beta(n)\, \text{Log}\, n + k/B$ | [277] |
| | Rectangle | $N \log^4 N$ | $\beta(n)\, \text{Log}\, n + k/B$ | [277] |

**Table 2.** Asymptotic upper bounds for secondary memory structures; here $N = n/B$, $\text{Log}\, n = \log_B n$, and $\beta(n) = \log\log \text{Log}\, n$.

bounds proved in Section 3.2 imply that we cannot hope for small worst-case bounds — and it should support insertions and deletions of points. Keeping these goals in mind, a plethora of data structures have been proposed. We will sketch the general ideas and mention some of the data structures in a little detail. For the sake of simplicity, we will present most of the data structures in two dimensions. The book by Samet [251] is an excellent survey of data structures developed in 1970's and 80's; more recent results are described in the survey papers [145, 156, 169, 226, 227].

The most widely used data structures for answering 1-dimensional range queries are B-trees and their variants [35, 96]. Since a B-tree requires a linear order on the input elements, one needs to define such an ordering on points in higher dimensions in order to store them into a B-tree. An obvious choice is lexicographical ordering, also known as the *bit concatenation* method, but this ordering performs rather poorly for higher dimensional range searching because a separate disk access may be required to report each point. A better scheme for ordering the points is the *bit-interleaving* method, proposed by Morton [217]. A point $p = (x, y)$, where the binary representations of $x$ and $y$ are $x = x_{m-1}x_{m-2}\dots x_0$ and $y = y_{m-1}y_{m-2}\dots y_0$, is regarded as the integer whose binary representation is $x_{m-1}y_{m-1}x_{m-2}\dots y_0$. A B-tree storing points based on the bit-interleaving ordering is referred to as an *N-tree* [285] or a *zkd-tree* [228] in the literature. See [251] for a more detailed discussion on the applications of bit interleaving in spatial data structures. Faloutsos [121] suggested using Gray codes to define a linear order on points. In general, space-filling curves[5] can be used to define a linear ordering on input points; Hilbert and Morton curves, shown in Figure 2, are the some of the space-filling curves commonly used for this purpose. See [1, 30, 126, 175] for a comparison of the performance of various space-filling curves in the context of range searching. Since B-trees require extra space to store pointers, several hashing schemes, including *linear hashing* [189], *dynamic z-hashing* [171] and *spiral hashing* schemes [219] are proposed to minimize the size of the data structure. The performance of any method that maps higher-dimensional points to a set of points in

[5]Formally speaking, a curve $\mathbb{R} \to [0,1]^d$ is called a *space-filling curve* if it visits each point of the unit hypercube exactly once. However, the same term often refers to approximations of space-filling curves that visit every point in a cubical lattice, such as the curves drawn in Figure 2. See the book by Sagan [248] for a detailed discussion on space-filling curves and [48] for some other applications of these curves.

one dimension deteriorates rapidly with the dimension because such a mapping does not preserve neighborhoods, though there has been some recent work on locality preserving hashing schemes [174].

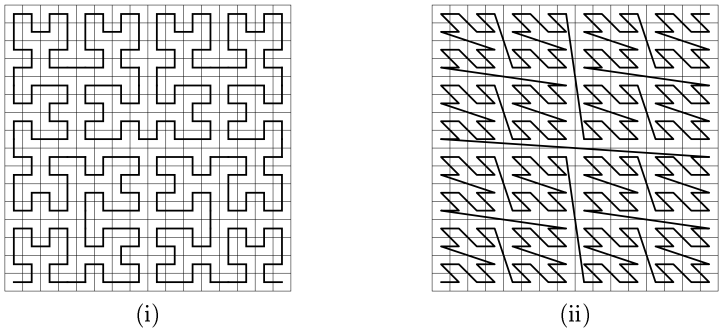


**Figure 2.** Examples of space-filling curves used for range searching. (i) Hilbert curve. (ii) Morton curve.

A more efficient approach to answer range queries is to construct a recursive partition of space, typically into rectangles, and to construct a tree induced by this partition. The simplest example of this type of data structure is the *quad tree* in the plane. A quad tree is a 4-way tree, each of whose nodes is associated with a square $B_v$. $B_v$ is partitioned into four equal-size squares, each of which is associated with one of the children of $v$. The squares are partitioned until one at most point is left inside a square. Quad trees can be extended to higher dimensions in an obvious way (they are called *oct-trees* in 3-space). In $d$-dimensions, a node has $2^d$ children. A range-search query can be answered by traversing the quad tree in a top-down fashion. Because of their simplicity, quad trees are one of the most widely used data structures for a variety of problems. For example, they were used as early as in 1920's, by Weyl [282] for computing the complex roots of a univariate polynomial approximately; Greengard used them for the so-called $n$-body problem [146]. See the book by Samet [250, 251] for a detailed discussion on quad trees and their applications.

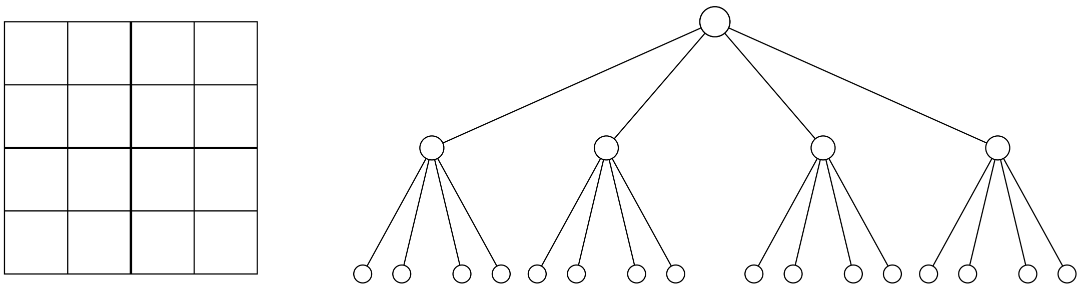

**Figure 3.** A quad tree

One disadvantage to quad trees is that arbitrarily many levels of partitioning may be required to separate tightly clustered points. Finkel and Bentley [127] described a variant of the quad tree for range searching, called a *point quad-tree*, in which each node is associated with a rectangle and the rectangle is partitioned into four rectangles by choosing a point in the interior and drawing horizontal and vertical lines through that point. Typically the point is chosen so that the height of the tree is $O(\log n)$. A recent paper by Faloutsos and Gaede [122] analyzes the performance of quad trees using Hausdorff fractal dimension. See also [138, 170] for other data structures based on quad trees.

In order to minimize the number of disk accesses, one can partition the square into many squares (instead of four) by a drawing either a uniform or a nonuniform grid. The *grid file*, introduced by Nievergelt *et al.* [224] is based on this idea. Since grid files are used frequently in geographic information systems, we describe them briefly. A grid file partitions the plane into a nonuniform grid by drawing horizontal and vertical lines. The grid lines are chosen so that the points in each cell can be stored in a single block of the disk. The grid is then partitioned into rectangles, each rectangle being the union of a subset of grid cells, so that the points in each rectangle can still be stored in a single block of the disk. The data within each block can be organized in an arbitrary way. The grid file maintains two pieces of information: a *grid directory*, which stores the index of the block that stores the points lying in each grid cell, and two arrays, called *linear scales*, which store the $x$-coordinates (resp. $y$-coordinates) of the vertical (resp. horizontal) lines. It is assumed that the linear scales are small enough to be stored in main memory. A point can be accessed by two disk accesses as follows. By searching with the $x$- and $y$-coordinates of the points in the linear scales, we determine the grid cell that contains the point. We then access that cell of the grid directory (using one disk access) to determine the index of the block that stores $p$, and finally we access that block and retrieve the point $p$ (second disk access). A range query is answered by locating the cells that contain the corners of the query rectangle and thus determining all the grid cells that intersect the query rectangle. We then access each of these cells to report all the points lying in the query rectangle. Several heuristics are used to minimize the number of disk accesses required to answer a query and to update the structures as points are inserted or deleted. Note that a range query reduces to another range query on the grid directory, so one can store the grid directory itself as a grid file. This notion of a hierarchical grid file was proposed by Hinrichs [165] and Krishnamurthy and Wang [190]. A related data structure, known as the *BANG file*, was proposed by Freestone [136]; other variants of grid files are proposed in [165, 172, 229].

Quad trees, grid files, and their variants construct a grid on a rectangle containing all the input points. One can instead partition the enclosing rectangle into two rectangles by drawing a horizontal or a vertical line and partitioning each of the two rectangles independently. This is the idea behind the so called *kd-tree* due to Bentley [41]. In particular, a $kd$-tree is a binary tree, each of whose nodes $v$ is associated with a rectangle $B_v$. If $B_v$ does not contain any point in its interior, $v$ is a leaf. Otherwise, $B_v$ is partitioned into two rectangles by drawing a horizontal or vertical line so that each rectangle contains at most half of the points; splitting lines are alternately horizontal and vertical. A $kd$-tree can be extended to higher dimensions in an obvious manner.

In order to minimize the number of disk accesses, Robinson [245] suggested the following generalization of a $kd$-tree, which is known as a *kd-B-tree*. One can construct a B-tree instead of a binary tree on the recursive partition of the enclosing rectangle, so all leaves of the tree are at the same level and each node has between $B/2$ and $B$ children. The rectangles associated with the children are obtained by splitting $B_v$ recursively, as in a *kd*-tree approach; see Figure 4(i). Let $w_1, \ldots w_s$ be the children of $v$. Then $B_{w_1}, \ldots, B_{w_s}$ can be stored implicitly at $v$ by storing them as a *kd*-tree, or the coordinates of their corners can be stored explicitly. If points are dynamically inserted into a *kd*-B-tree, then some of the nodes may have to be split, which is an expensive operation, because splitting a node may require reconstructing the entire subtree rooted at that node. Several variants of *kd*-B-trees have been proposed to minimize the number of splits, to optimize the space, and to improve the query time [195, 137, 120, 36, 257, 163, 258, 255, 260]. We mention only two of the variants here: *Buddy trees* [257] and *hB-trees* [195, 120]. A buddy tree is a combination of a quad tree and *kd*-B-tree in the sense that rectangles are split into sub-rectangles only at some specific locations, which simplifies the split procedure; see [257] for details. If points are in degenerate position, then it may not be possible to split them into two halves by a line. Lomen and Salzberg [195] circumvent this problem by introducing a new data structure, called *hB-tree*, in which the region associated with a node is allowed to be $R_1 \setminus R_2$ where $R_1$ and $R_2$ are rectangles. A more refined version of this data structure, known as $hB^{\Pi}$*-tree*, is presented in [120].

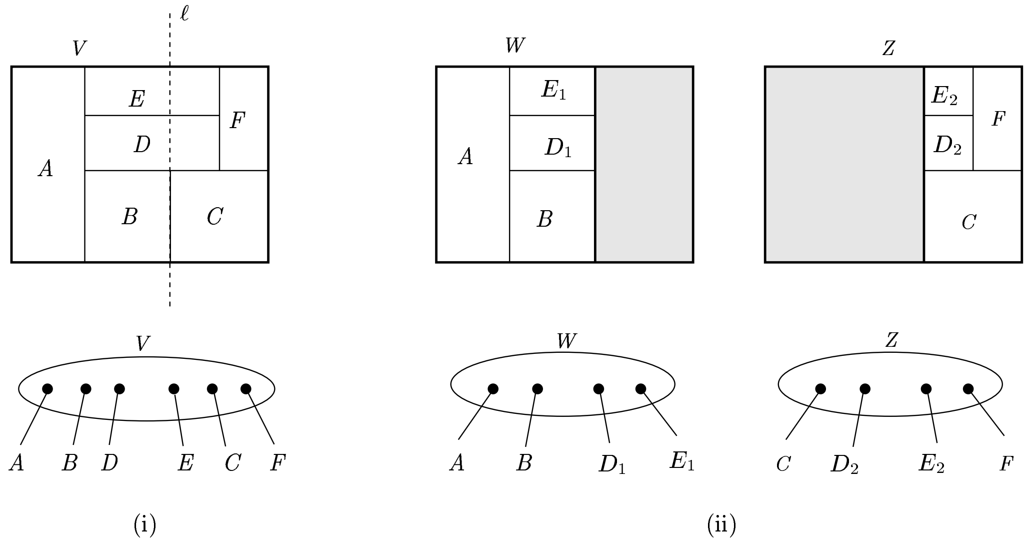


**Figure 4.** Splitting a node $V$ of a *kd*-B-tree by a line $\ell$. (i) $V$ before the split. (ii) $V$ is split into two nodes $W$ and $Z$; the subtrees rooted at $D$ and $E$ are split recursively.

In a *kd*-tree, a rectangle is partitioned into two rectangles by drawing a horizontal or vertical line. One can instead associate a convex polygon $B_v$ with each node $v$ of the tree, use an arbitrary line to partition $B_v$ into two convex polygons, and associate the two polygons with the children of $v$. This idea is the same as in binary space partition trees [139, 237]. Again, one can construct a B-tree on this recursive partitioning scheme to reduce the number of disk accesses. The resulting structure called *cell trees* is studied in [150, 151].

All the data structures described in this section construct a recursive partition of the space. There are other data structures (of which the $R$-tree is perhaps the most famous example) that construct a hierarchical cover of the space. We will discuss some of these data structures in the next subsection.

## 3.5 The Partial Sum Problem

Preprocess a $d$-dimensional array $A$ with $n$ entries, in an additive semigroup, into a data structure, so that for a $d$-dimensional rectangle $\gamma = [a_1, b_1] \times \cdots \times [a_d, b_d]$, the sum

$$\sigma(A, \gamma) = \sum_{(k_1, k_2, \ldots, k_d) \in \gamma} A[k_1, k_2, \ldots, k_d]$$

can be computed efficiently. In the offline version, given $A$ and $m$ rectangles $\gamma_1, \gamma_2, \ldots, \gamma_m$, we wish to compute $\sigma(A, \gamma_i)$ for each $i$. Note that this is just a special case of orthogonal range searching, where the points lie on a regular $d$-dimensional lattice.

Partial-sum queries are widely used for on-line analytical processing (OLAP) of commercial databases. OLAP allows companies to analyze aggregate databases built from their data warehouses. A popular data model for OLAP applications is the multidimensional database, known as *data cube* [144], which represents the data as $d$-dimensional array. Thus, an aggregate query can be formulated as a partial-sum query. Driven by this application, several heuristics have been proposed to answer partial-sum queries on data cubes [152, 159, 168, 262, 167, 247].

Yao [291] showed that, for $d = 1$, a partial-sum query can be answered in $O(\alpha(n))$ time using $O(n)$ space.[6] If the additive operator is *max* or *min*, then a partial-sum query can be answered in $O(1)$ time under the RAM model using a Cartesian tree, developed by Vuillemin [279], and the nearest-common-ancestor algorithm of Harel and Tarjan [158].

For $d > 1$, Chazelle and Rosenberg [80] gave a data structure of size $O(n \log^{d-1} n)$ that can answer a partial-sum query in time $O(\alpha(n) \log^{d-2} n)$. They also showed that the offline version takes $\Omega(n + m\alpha(m, n))$ time for any fixed $d \geq 1$. If points are allowed to insert into $S$, the query time is $\Omega(\log n / \log \log n)$ [130, 292] for the one-dimensional case; the bounds were extended by Chazelle [61] to $\Omega((\log n / \log \log n)^d)$, for any fixed dimension $d$. Chazelle [56] extended the data structure by Yao to the following variant of the partial-sum problem: Let $T$ be a rooted tree with $n$ nodes, each of whose node is associated with an element of a commutative semigroup. Preprocess $T$ so that for a query node $v$, the sum of the weights in the subtree rooted at $v$ can be computed efficiently. Chazelle showed that such a query can be answered in $O(\alpha(n))$ time, using $O(n)$ space.

[6] Here, $\alpha(n)$ and $\alpha(m, n)$ denote functional inverses of Ackermann's function. These functions go extremely slowly; for example, $\alpha(n) \leq 4$ for all $n \leq 2 \uparrow (2 \uparrow (2 \uparrow 2)) = 2 \uparrow 2^{16}$, where for any positive integer $k$, $2 \uparrow k = 2^{2 \uparrow (k-1)}$ denotes an exponential tower of $k$ twos. For formal definitions, see [259].

### 3.6 Rectangle-Rectangle Searching

Preprocess a set $S$ of $n$ rectangles in $\mathbb{R}^d$ so that for a query rectangle $q$, the rectangles of $S$ that intersect $q$ can be reported (or counted) efficiently. Rectangle-rectangle searching is central to many applications because, in practice, polygonal objects are approximated by rectangles. Chazelle [58] has shown that the bounds mentioned in Table 1 also hold for this problem.

In practice, two general approaches are used to answer a query. A rectangle $\prod_{i=1}^{d}[a_i, b_i]$ in $\mathbb{R}^d$ can be mapped to the point $(a_1, a_2, \ldots, a_d, b_1, b_2, \ldots, b_d)$ in $\mathbb{R}^{2d}$, and a rectangle-intersection query can be reduced to orthogonal range searching. Many heuristic data structures based on this scheme have been proposed; see [125, 235, 257] for a sample of such results. The second approach is to construct a data structure on $S$ directly in $\mathbb{R}^d$. The most popular data structure based on this approach is the *R-tree*, originally introduced by Guttman [157].

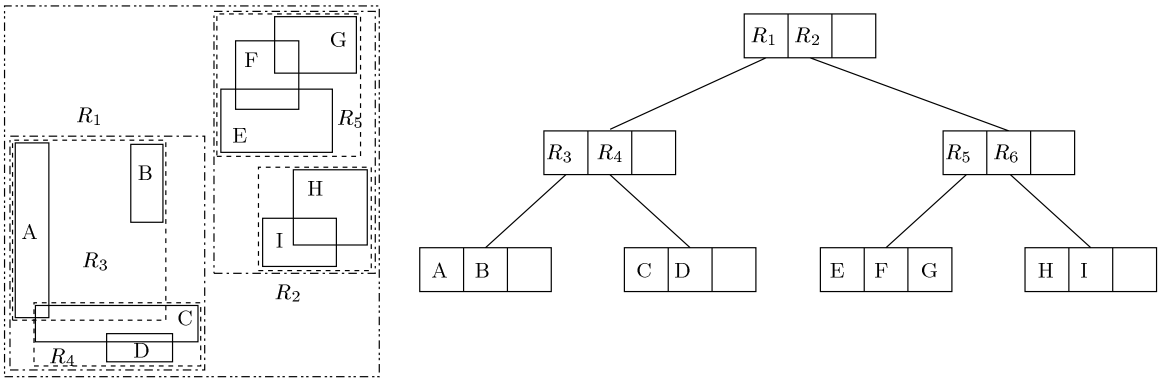


**Figure 5.** An R-tree.

An R-tree is a multiway tree (like a B-tree), each of whose nodes stores a set of rectangles. Each leaf stores a subset of input rectangles, and each input rectangle is stored at exactly one leaf. For each node $v$, let $R_v$ be the smallest rectangle containing all the rectangles stored at $v$; $R_v$ is stored at the parent of $v$ (along with the pointer to $v$). $R_v$ induces the subspace corresponding to the subtree rooted at $v$, in the sense that for any query rectangle intersecting $R_v$, the subtree rooted at $v$ is searched. Rectangles stored at a node are allowed to overlap. Therefore, unlike all the data structures discussed in Section 3.4, a R-tree forms a recursive cover of the data space, instead of a recursive partition. Although allowing rectangles to overlap helps reduce the size of the data structure, answering a query becomes more expensive. Guttman suggests some heuristics to construct a R-tree so that the overlap is minimized. Better heuristics for minimizing the overlap were developed by Beckmann *et al.* [37], Green [145], and Kamal and Faloutsos [177, 178, 179]. There are many variants of R-tree, depending on the application: an *$R^*$-tree* [37] uses more sophisticated techniques to minimize the overlap; a *Hilbert-R-tree* [179] defines a linear ordering on the rectangles, by sorting their centers along the Hilbert space-filling curve, and constructs a B-tree based on this ordering of rectangles; and an $R^+$*-tree* avoids overlapping directory cells by clipping rectangles [257]. Additional variants are suggested to avoid overlap in higher dimensions. Berchtold *et al.* [47] define the *X-tree*, in which the interior nodes are allowed to be arbitrarily large; Lin *et al.* [194] project rectangles onto a lower dimensional space and construct an R-tree (or some variant thereof) on these projections. Leutenegger *et al.* [193] compare different variants of R-trees and discuss advantages of different heuristics used to minimize the overlap of rectangles.

We discuss some more general rectangle-intersection searching problems in Section 6.3.

# 4 Simplex Range Searching

As mentioned in the introduction, simplex range searching has received considerable attention during the last few years. Besides its direct applications, simplex range-searching data structures have provided fast algorithms for numerous other geometric problems. See the survey paper by Matoušek [204] for an excellent review of techniques developed for simplex range searching.

Unlike orthogonal range searching, no simplex range-searching data structure is known that can answer a query in polylogarithmic time using near-linear storage. In fact, the lower bounds stated below indicate that there is very little hope of obtaining such a data structure, since the query time of a linear-size data structure, under the semigroup model, is roughly at least $n^{1-1/d}$ (thus saving only a factor of $n^{1/d}$ over the naïve approach). Since the size and query time of any data structure have to be at least linear and logarithmic, respectively, we consider these two ends of the spectrum: (i) How fast can a simplex range query be answered using a linear-size data structure, and (ii) how large should the size of a data structure be in order to answer a query in logarithmic time. By combining these two extreme cases, as we describe below, we obtain a tradeoff between space and query time.

Unless stated otherwise, each of the data structures we describe in this section can be constructed in time that is only a polylogarithmic or $n^{\varepsilon}$ factor larger than its size.

## 4.1 Linear-size data structures

Most of the linear-size data structures for simplex range searching are based on so-called *partition trees*, originally introduced by Willard [286]. Roughly speaking, partition trees are based on the following idea: Given a set $S$ of points in $\mathbb{R}^d$, partition the space into a few, say, a constant number of, regions, each containing roughly equal number of points, so that for any hyperplane $h$, the number of points lying in the regions that intersect $h$ is much less than the total number of points. Then recursively construct a similar partition for the subset of points lying in each region.

Willard's original partition tree for a set $S$ of $n$ points in the plane is a 4-way tree, constructed as follows. Let us assume that $n$ is of the form $4^k$ for some integer $k$, and that the points of $S$ are in general position. If $k = 0$, the tree consists of a single node that stores the coordinates of the only point in $S$. Otherwise, using the ham-sandwich theorem [108], find two lines $\ell_1, \ell_2$ so that each quadrant $Q_i$, for $1 \leq i \leq 4$, induced by $\ell_1, \ell_2$ contains exactly $n/4$ points. The root stores the equations of $\ell_1, \ell_2$ and the value of $n$. For each quadrant, recursively construct a partition tree for $S \cap Q_i$ and attach it as the $i^{th}$ subtree of the root. The total size of the data structure is linear, and it can be constructed in $O(n \log n)$ time. A halfplane range-counting query can be answered as follows. Let $h$ be a query halfplane. Traverse the tree, starting from the root, and maintain a global count. At each node $v$ storing $n_v$ nodes in its subtree, perform the following step: If the line $\partial h$ intersects the quadrant $Q_v$ associated with $v$, recursively visit the children of $v$. If $Q_v \cap h = \emptyset$, do nothing. Otherwise, since $Q_v \subseteq h$, add $n_v$ to the global count. The quadrants associated with the four children of any interior node are induced by two lines, so $\partial h$ intersects at most three of them, which implies that the query procedure does not explore the subtree of one of the children. Hence, the query time of this procedure is $O(n^\alpha)$, where $\alpha = \log_3 4 \leq 0.7925$. A similar procedure can answer a simplex range-counting query within the same time bound, and a simplex range-reporting query in time $O(n^\alpha + k)$. Edelsbrunner and Welzl [114] described a simple variant of Willard's partition tree that improves the exponent in the query-search time to $\log_2(1+\sqrt{5}) - 1 \approx 0.695$.

A partition tree for points in $\mathbb{R}^3$ was first proposed by Yao [294], which can answer a query in time $O(n^{0.98})$. This bound was improved slightly in subsequent papers [101, 110, 295]. Using the Borsuk-Ulam theorem, Yao *et al.* [295] showed that, given a set $S$ of $n$ points in $\mathbb{R}^3$, one can find three planes so that each of the eight (open) octants determined by them contains at most $\lfloor n/8 \rfloor$ points of $S$. Avis [31] proved that such a partition of $\mathbb{R}^d$ by $d$ hyperplanes is not always possible for $d \geq 5$; the problem is still open for $d = 4$. Weaker partitioning schemes were proposed in [94, 293].

After the initial improvements and extensions on Willard's partition tree, a major breakthrough was made by Haussler and Welzl [161]. They formulated range searching in an abstract setting and, using elegant probabilistic methods, gave a randomized algorithm to construct a linear-size partition tree with $O(n^\alpha)$ query time, where $\alpha = 1 - \frac{1}{d(d-1)+1} + \varepsilon$ for any $\varepsilon > 0$. The major contribution of their paper is the abstract framework and the notion of $\varepsilon$-nets. A somewhat different abstract framework for randomized algorithms was proposed by Clarkson [88, 92] around the same time; see also [220]. These abstract frameworks and the general results attained under these frameworks popularized randomized algorithms in computational geometry [223]. We briefly describe the framework and the main result by Haussler and Welzl because they are most pertinent to range searching.

A *range space* is a set system $\Sigma = (X, \mathcal{R})$ where $X$ is a set of objects and $\mathcal{R}$ is a family of subsets of $X$. The elements of $\mathcal{R}$ are called the *ranges* of $\Sigma$. $\Sigma$ is called a *finite range space* if the ground set $X$ is finite. Here are a few examples of geometric range spaces:

(i) $\Sigma_1 = (\mathbb{R}^d, \{h \mid h \text{ is a halfspace in } \mathbb{R}^d\})$,

(ii) $\Sigma_2 = (\mathbb{R}^d, \{B \mid B \text{ is a ball in } \mathbb{R}^d\})$,

(iii) Let $H$ be the set of all hyperplanes in $\mathbb{R}^d$. For a segment $s$, let $H_s \subseteq H$ be the set of all hyperplanes intersecting $s$. Define the range space $\Sigma_3 = (H, \{H_s \mid s \text{ is a segment in } \mathbb{R}^d\})$.

For a finite range space $\Sigma = (X, \mathcal{R})$, a subset $N \subseteq X$ is called an *$\varepsilon$-net* if $N \cap \gamma \neq \emptyset$ for every range $\gamma \in \mathcal{R}$ with $|\gamma| \geq \varepsilon|X|$. That is, $N$ intersects every "large" range of $\Sigma$. (The notion of $\varepsilon$-nets can be extended to infinite range spaces as well.) A subset $A \subseteq X$ can be *shattered* if every subset of $A$ has the form $A \cap \gamma$ for some $\gamma \in \mathcal{R}$. The *Vapnik-Chervonenkis dimension*, or *VC-dimension*, of a range space $\Sigma$ is the size of the largest subset $A$ that can be shattered. For example, the VC-dimensions of $\Sigma_1, \Sigma_2$, and $\Sigma_3$ are $d+1$, $d+2$, and $2d$, respectively. The main result of Haussler and Welzl is that, given a finite range space $\Sigma = (X, \mathcal{R})$ and parameters $0 < \varepsilon, \delta \leq 1$, if we choose a random subset $N \subset X$ of size

$$\max\left\{\frac{8d}{\varepsilon}\log\frac{8d}{\varepsilon}, \frac{4}{\varepsilon}\log\frac{2}{\delta}\right\},$$

then $N$ is an $\varepsilon$-net of $\Sigma$ with probability at least $1-\delta$. The bound on the size of $\varepsilon$-nets was improved by Blumer *et al.* [49] and Komlós *et al.* [187].

**Theorem 2** (Komlós *et al.* [187])**.** *For any finite range space $(X, \mathcal{R})$ of VC-dimension $d$ and for any $0 < \varepsilon < 1$, if $N$ is a subset of $X$ obtained by*

$$\frac{d}{\varepsilon}\left(\log\frac{1}{\varepsilon} + 2\log\log\frac{1}{\varepsilon} + 3\right)$$

*random independent draws, then $N$ is an $\varepsilon$-net of $(X, \mathcal{R})$ with probability at least $1 - e^{-d}$.*

Theorem 2 and some other similar results [88, 92] have been used extensively in computational geometry and learning theory; see the books by Motwani and Raghavan [218], Mulmuley [223], and Anthony and Biggs [19] and the survey papers [90, 200, 256].

The first linear-size data structure with near-optimal query time for simplex range queries in the plane was developed by Welzl [280]. His algorithm is based on the following idea. A *spanning path* of a set $S$ of points is a polygonal chain whose vertices are the points of $S$. The *crossing number* of a polygonal path is the maximum number of its edges that can be crossed by a hyperplane. Using Theorem 2, Welzl constructs a spanning path $\Pi = \Pi(S)$ of any set $S$ of $n$ points in $\mathbb{R}^d$ whose crossing number is $O(n^{1-1/d}\log n)$. The bound on the crossing number was improved by Chazelle and Welzl [83] to $O(n^{1-1/d})$, which is tight in the worst case. Let $p_1, p_2, \ldots, p_n$ be the vertices of $\Pi$. If we know the edges of $\Pi$ that cross $h$, then the weight of points lying in one of the halfspaces bounded by $h$ can be computed by answering $O(n^{1-1/d})$ partial-sum queries on the sequence $W = \langle w(p_1), \ldots, w(p_n)\rangle$. Hence, by processing $W$ for partial-sum queries, we obtain a linear-size data structure for simplex range searching, with $O(n^{1-1/d}\alpha(n))$ query time, in the semigroup arithmetic model. (Recall that the time spent in finding the edges of $\Pi$ crossed by $h$ is not counted in the semigroup model.) In any realistic model of computation such as pointer machines or RAMs, however, we also need an efficient linear-size data structure for computing the edges of $\Pi$ crossed by a hyperplane. Chazelle and Welzl [83] produced such a data structure for $d \leq 3$, but no such structure is known for higher dimensions. Although spanning paths were originally introduced for simplex range searching, they have been successfully applied to solve a number of other algorithmic as well as combinatorial problems; see, for example, [3, 85, 109, 207, 234, 281].

Matoušek and Welzl [206] gave an entirely different algorithm for the halfspace range-counting problem in the plane, using a combinatorial result of Erdős and Szekeres [116]. The query time of their data structure is $O(\sqrt{n}\log n)$, and it uses $O(n)$ space and $O(n^{3/2})$ preprocessing time. If subtractions are allowed, their algorithm can be extended to the triangle range-counting problem. An interesting open question is whether the preprocessing time can be improved to near linear. In order to make this improvement, we need a near-linear time algorithm for the following problem, which is interesting its own right: Given a sequence $X$ of $n$ integers, partition $X$ into $O(\sqrt{n})$ subsequences, each of which is either monotonically increasing or decreasing. The existence of such a partition of $X$ follows from the result by Erdős and Szekeres, but the best known algorithm for computing such a partition runs in time $O(n^{3/2})$ [33]. However, a longest monotonically increasing subsequence of $X$ can be computed in $O(n\log n)$ time. The technique by Matoušek and Welzl has also been applied to solve some other geometric-searching problems, including ray shooting and intersection searching [34].

The first data structure with roughly $n^{1-1/d}$ query time and near-linear space, for $d > 3$, was obtained by Chazelle *et al.* [82]. Given a set $S$ of $n$ points in $\mathbb{R}^d$, they construct a family $\mathcal{F} = \{\Xi_1, \ldots, \Xi_k\}$ of triangulations of $\mathbb{R}^d$, each of size $O(r^d)$. For any hyperplane $h$, there is at least one $\Xi_i$ so that only $O(n/r)$ points lie in the simplices of $\Xi_i$ that intersect $h$. Applying this construction recursively, they obtain a tree structure of size $O(n^{1+\varepsilon})$ that can answer a halfspace range-counting query in time $O(n^{1-1/d})$. The extra $n^\varepsilon$ factor in the space is due to the fact that they maintain a family of partitions instead of a single partition. Another consequence of maintaining a family of partitions is that, unlike partition trees, this data structure cannot be used directly to answering simplex range queries. Instead, Chazelle *et al.* [82] construct a multi-level data structure (which we describe in Section 5.1) to answer simplex range queries.

Matoušek [203] developed a simpler, slightly faster data structure for simplex range queries, by returning to the theme of constructing a single partition, as in the earlier partition-tree papers. His algorithm is based on the following partition theorem, which can be regarded as an extension of the result by Chazelle and Welzl.

**Theorem 3** (Matoušek [198]). *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, and let $1 < r \le n/2$ be a given parameter. Then there exists a family of pairs*

$$\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$$

*such that each $S_i \subseteq S$ lies inside the simplex $\Delta_i$, $n/r \le |S_i| \le 2n/r$, $S_i \cap S_j = \emptyset$ for all $i \ne j$, and every hyperplane crosses at most $cr^{1-1/d}$ simplices of $\Pi$; here $c$ is a constant. If $r \le n^\alpha$ for some suitable constant $0 < \alpha < 1$, then $\Pi$ can be constructed in $O(n\log r)$ time.*

Note that although $S$ is being partitioned into a family of subsets, unlike the earlier results on partition trees, it does not partition $\mathbb{R}^d$ because $\Delta_i$'s may intersect. In fact, it is an open problem whether $\mathbb{R}^d$ can be partitioned into $O(r)$ disjoint simplices that satisfy the above theorem.

Using Theorem 3, a partition tree $T$ can be constructed as follows. Each interior node $v$ of $T$ is associated with a canonical subset $C_v \subseteq S$ and a simplex $\Delta_v$ containing $C_v$; if $v$ is the root of $T$, then $C_v = S$ and $\Delta_v = \mathbb{R}^d$. Choose $r$ to be a sufficiently large constant. If $|S| \leq 4r$, $T$ consists of a single node, and it stores all points of $S$. Otherwise, we construct a family of pairs $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ using Theorem 3. The root $u$ stores the value of $n$. We recursively construct a partition tree $T_i$ for each $S_i$ and attach $T_i$ as the $i$th subtree of $u$. The root of $T_i$ also stores $\Delta_i$. The total size of the data structure is linear, and it can be constructed in time $O(n \log n)$. A simplex range-counting query can be answered in the same way as with Willard's partition tree. Since any hyperplane intersects at most $cr^{1-1/d}$ simplices of $\Pi$, the query time is $O(n^{1-1/d+\log_r c})$; the $\log_r c$ term in the exponent can be reduced to any arbitrarily small positive constant $\varepsilon$ by choosing $r$ sufficiently large. The query time can be improved to $O(n^{1-1/d} \operatorname{polylog} n)$ by choosing $r = n^{\varepsilon}$.

In a subsequent paper Matoušek [203] proved a stronger version of Theorem 3, using some additional sophisticated techniques (including Theorem 5 described below), that gives a linear-size partition tree with $O(n^{1-1/d})$ query time.

If the points in $S$ lie on a $k$-dimensional algebraic surface of constant degree, the *crossing number* in Theorem 3 can be improved to $O(r^{1-1/\gamma})$, where $\gamma = 1/\lfloor (d+k)/2 \rfloor$ [8], which implies that in this case a simplex range query can be answered in time $O(n^{1-1/\gamma+\varepsilon})$ using linear space.

Finally, we note that better bounds can be obtained for the halfspace range-reporting problem, using the so-called *filtering search* technique introduced by Chazelle [55]. All the data structured mentioned above answer a range-reporting query in two stages. The first stage "identifies" the $k$ points of a query output, in time $f(n)$ that is independent of the output size, and the second stage explicitly reports these points in $O(k)$ time. Chazelle observes that since $\Omega(k)$ time will be spent in reporting $k$ points, the first stage can compute in $f(n)$ time a superset of the query output of size $O(k)$, and the second stage can "filter" the actual $k$ points that lie in the query range. This observation not only simplifies the data structure but also gives better bounds in many cases, including halfspace range reporting. See [15, 55, 66, 79] for some applications of filtering search.

An optimal halfspace reporting data structure in the plane was proposed by Chazelle *et al.* [78]. They compute *convex layers* $L_1, \ldots, L_m$ of $S$, where $L_i$ is the set of points lying on the boundary of the convex hull of $S \setminus \bigcup_{j<i} L_j$, and store them in a linear-size data structure, so that a query can be answered in $O(\log n + k)$ time. Their technique does not extend to three dimensions. After a few initial attempts [79, 16], Matoušek developed a data structure that answers a halfspace reporting query in $\mathbb{R}^d$ in time $O(n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n + k)$. His structure is based on the following two observations. A hyperplane is called $\lambda$-*shallow* if one of the halfspaces bounded by $h$ contains at most $\lambda$ points of $S$. If the hyperplane bounding a query halfspace is not $\lambda$-shallow, for some $\lambda = \Omega(n)$, then a simplex range-reporting data structure can be used to answer a query in time $O(n^{1-1/d+\varepsilon} + k) = O(k)$. For shallow hyperplanes, Matoušek proves the following theorem, which is an analog of

Theorem 3.

**Theorem 4** (Matoušek [199]). *Let $S$ be a set of $n$ points in $\mathbb{R}^d$ $(d \geq 4)$ and let $1 \leq r < n$ be a given parameter. Then there exists a family of pairs*

$$\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$$

*such that each $S_i \subseteq S$ lies inside the simplex $\Delta_i$, $n/r \leq |S_i| \leq 2n/r$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and every $(n/r)$-shallow hyperplane crosses $O(r^{1-1/\lfloor d/2 \rfloor})$ simplices of $\Pi$. If $r \leq n^\alpha$ for some suitable constant $0 < \alpha < 1$, then $\Pi$ can be constructed in $O(n \log r)$ time.*

Using this theorem, a partition tree for $S$ can be constructed in the same way as for simplex range searching, except that at each node $v$ of the tree, we also preprocess the corresponding canonical subset $C_v$ for simplex range searching and store the resulting data structure as a secondary data structure of $v$. While answering a query for a halfspace $h^+$, if $h^+$ crosses more than $O(r^{1-1/\lfloor d/2 \rfloor})$ simplices of the partition $\Pi_v$ associated with a node $v$, then it reports all points of $h^+ \cap C_v$ using the simplex range-reporting data structure stored at $v$. Otherwise, for each pair $(S_i, \Delta_i) \in \Pi_v$, if $\Delta_i \subseteq h^+$, it reports all points $S_i$, and if $\Delta_i$ is crossed by $h$, it recursively visits the corresponding child of $v$.

If we are interested only in determining whether $h^+ \cap S = \emptyset$, we do not have to store simplex range-searching structure at each node of the tree. Consequently, the query time and the size of the data structure can be improved slightly; see Table 3 for a summary of results.

| **Problem** | $d$ | **Size** | **Query Time** | **Source** |
|---|---|---|---|---|
| Reporting | $d = 2$ | $n$ | $\log n + k$ | [78] |
| Emptiness | | $n$ | $\log n$ | [242] |
| Reporting | $d = 3$ | $n \log n$ | $\log n + k$ | [16] |
| Emptiness | | $n$ | $\log n$ | [103] |
| Reporting | $d > 3$ | $n \log \log n$ | $n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n + k$ | [199] |
| Emptiness | | $n$ | $n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$ | [199] |

**Table 3.** Asymptotic upper bounds for halfspace range searching in near-linear space.

Since the query time of a linear-size simplex range-searching data structure is only a $n^{1/d}$ factor better than the naïve method, researchers have developed practical data structures that work well most of the time. For example, Arya and Mount [27] have developed a linear-size data structure for answering approximate range-counting queries, in the sense that the points lying within distance $\delta \cdot \operatorname{diam}(\Delta)$ distance of the boundary of the query simplex $\Delta$ may or may not be counted. Its query time is $O(\log n + 1/\delta^{d-1})$. Overmars and van der Stappen [233] developed fast data structures for the special case in which the ranges are "fat" and have bounded size. In practice, the data structures described in Section 3.4 are used even for simplex range searching. Recently, Goldstein *et al.* [141] presented an algorithm for simplex range searching using $R$-trees. Although these data structures do not work well in the worst case, they perform reasonably well in practice, for example, when the points are close to uniformly distributed. It is an open question whether simple data structures can be developed for simplex range searching that work well on typical data sets.

### 4.2 Data structures with logarithmic query time

For the sake of simplicity, we first consider the halfspace range-counting problem. We need a few definitions and concepts before we describe the data structures.

The dual of a point $(a_1, \ldots, a_d) \in \mathbb{R}^d$ is the hyperplane $x_d = -a_1x_1 - \cdots - a_{d-1}x_{d-1} + a_d$, and the dual of a hyperplane $x_d = b_1x_1 + \cdots b_{d-1}x_{d-1} - b_d$ is the point $(-b_1, \ldots, -b_d)$. A nice property of duality is that it preserves the above-below relationship: a point $p$ is above a hyperplane $h$ if and only if the dual hyperplane $p^*$ is above the dual point $h^*$; see Figure 6.

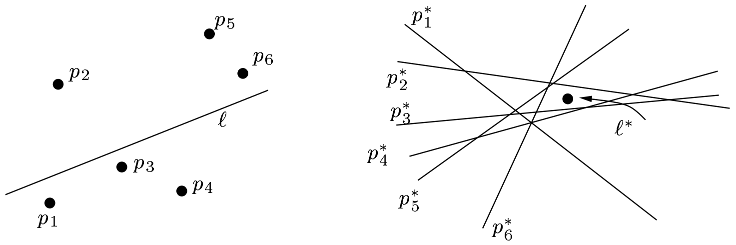


**Figure 6.** A set of points and the arrangement of their dual lines.

The *arrangement* of a set $H$ of hyperplanes in $\mathbb{R}^d$ is the subdivision of $\mathbb{R}^d$ into cells of dimensions $k$, for $0 \le k \le d$, each cell being a maximal connected set contained in the intersection of a fixed subset of $H$ and not intersecting any other hyperplane of $H$. The level of a point in $\mathcal{A}(H)$ is the number of hyperplanes lying strictly below the point. Let $\mathcal{A}_{\le k}(H)$ denote the (closure of the) set of points with level at most $k$. A $(1/r)$-*cutting* of $H$ is a set $\Xi$ of (relatively open) disjoint simplices covering $\mathbb{R}^d$ so that each simplex intersects at most $n/r$ hyperplanes of $H$. Clarkson [88] and Haussler and Welzl [161] were the first to show the existence of a $(1/r)$-cutting of $H$ of size $O(r^d \log^d r)$. Chazelle and Friedman [74] improved the size bound to $O(r^d)$, which is optimal in the worst case. Several efficient algorithms are developed for computing a $(1/r)$-cutting. The best algorithm known for computing a $(1/r)$-cutting was discovered by Chazelle [62]; his result is summarized in the following theorem.

**Theorem 5** (Chazelle [62])**.** *Let $H$ be a set of $n$ hyperplanes and $r \le n$ a parameter. Set $k = \lceil \log_2 r \rceil$. There exist $k$ cuttings $\Xi_1, \ldots, \Xi_k$ so that $\Xi_i$ is a $(1/2^i)$-cutting of size $O(2^{id})$, each simplex of $\Xi_i$ is contained in a simplex of $\Xi_{i-1}$, and each simplex of $\Xi_{i-1}$ contains a constant number of simplices of $\Xi_i$. Moreover, $\Xi_1, \ldots, \Xi_k$ can be computed in time $O(nr^{d-1})$.*

This theorem has been successfully applied to many geometric divide-and-conquer algorithms; see [2, 62, 99, 239] for a few such instances.

Returning to halfspace range searching, suppose that the query halfspace always lies below its bounding hyperplane. Then the halfspace range-counting problem reduces via duality to the following problem: Given a set $H$ of $n$ hyperplanes in $\mathbb{R}^d$, determine the number of hyperplanes of $H$ that lie above a query point. Since the same subset of hyperplanes lies above all points in a single cell of $\mathcal{A}(H)$, the arrangement of $H$, we can answer a halfspace range-counting query by locating the cell of $\mathcal{A}(H)$ that contains the point dual to the hyperplane bounding the query halfspace. Theorem 5 can be used in a straightforward manner to obtain a data structure of size $O((n/\log n)^d)$ with $O(\log n)$ query time.

The above approach for halfspace range counting can be extended to the simplex range-counting problem as well. That is, store the solution of every combinatorially distinct simplex (two simplices are combinatorially distinct if they do not contain the same subset of $S$). Since there are $\Theta(n^{d(d+1)})$ combinatorially distinct simplices, such an approach will require $\Omega(n^{d(d+1)})$ storage; see [95, 111].

Cole and Yap [95] were the first to present a near-quadratic size data structure that could answer a triangle range-counting query in the plane in $O(\log n)$ time. They present two data structures: the first one answers a query in time $O(\log n)$ using $O(n^{2+\varepsilon})$ space, and the other in time $O(\log n \log\log n)$ using $O(n^2/\log n)$ space. For $d = 3$, their approach gives a data structure of size $O(n^{7+\varepsilon})$ that can answer a tetrahedron range-counting query in time $O(\log n)$. Chazelle *et al.* [82] describe a multi-level data structure (see Section 5.1) of size $O(n^{d+\varepsilon})$ that can answer a simplex range-counting query in time $O(\log n)$. The space bound can be reduced to $O(n^d)$ by increasing the query time to $O(\log^{d+1} n)$ [203]. Both data structures can answer simplex range-reporting queries by spending an additional $O(k)$ time.

The size of a data structure can be significantly improved if we want to answer halfspace range-reporting queries. Using random sampling, Clarkson [88] showed that a halfspace-emptiness query can be answered in $O(\log n)$ time using $O(n^{\lfloor d/2\rfloor+\varepsilon})$ space. In order to extend his algorithm to halfspace range-reporting queries, we need the following additional idea. Let $H$ be a set of hyperplanes in $\mathbb{R}^d$. For a parameter $1 \le r < n$, we define a $(1/r)$-*cutting* for $\mathcal{A}_{\le l}(H)$ to be a collection $\Xi$ of (relatively open) disjoint simplices that cover $\mathcal{A}_{\le l}(H)$ and each simplex intersects at most $n/r$ hyperplanes of $H$. The following theorem by Matoušek [199] leads to a better data structure for answering halfspace range-reporting queries.

**Theorem 6** (Matoušek [199])**.** *Let $H$ be a collection of $n$ hyperplanes in $\mathbb{R}^d$, let $1 \le l, r < n$ be parameters, and let $q = lr/n + 1$. Then there exists a $(1/r)$-cutting for $\mathcal{A}_{\le l}(H)$, consisting of $O(r^{\lfloor d/2\rfloor} q^{\lceil d/2\rceil})$ simplices. If $r \le n^\alpha$ for some suitable constant $0 < \alpha < 1$, then $\Xi$ can be computed in $O(n \log r)$ time.*

Using Theorem 6, a halfspace range-reporting data structure $T$ can be constructed as follows. Each interior node $v$ of $T$ is associated with a canonical subset $C_v \subseteq H$ and a simplex $\Delta_v$; the root of $T$ is associated with $H$ and $\mathbb{R}^d$. Choose $r$ to be a sufficiently large constant. If $|C_v| \le 4r$, then $v$ is a leaf. Otherwise, set $l = |C_v|/r$, compute a $(1/r)$-cutting $\Xi_v$ of size $O(r^{\lfloor d/2\rfloor})$ for $\mathcal{A}_{\le l}(C_v)$, and create a child $w_i$ for each $\Delta_i \in \Xi_v$. Set $C_{w_i}$ to be the set of hyperplanes that either intersect or lie below $\Delta_i$. We also store $C_v$ at $v$. The size of the data structure is $O(n^{\lfloor d/2\rfloor+\varepsilon})$. Let $\gamma$ be a query point. The goal is to report all points lying above $\gamma$. Follow a path of $T$ as follows. Suppose the query procedure is visiting a node $v$ of $T$. If $v$ is a leaf or $\gamma$ does not lie in any simplex of $\Xi_v$ (*i.e.*, the level of $\gamma$ is at least $|C_v|/r$), then report all hyperplanes of $C_v$ lying above $\gamma$, by checking each hyperplane explicitly; this step takes $O(|C_v|) = O(kr) = O(k)$ time. Otherwise, recursively visit the node $w_i$ if $\Delta_i$ contains $\gamma$. The query time is obviously $O(\log n + k)$. The size of the data structure can be improved to $O(n^{\lfloor d/2 \rfloor} \operatorname{polylog} n)$ without affecting the asymptotic query time.

## 4.3 Trading space for query time

In the previous two subsections we surveyed data structures for simplex range searching that either use near-linear space or answer a query in polylogarithmic time. By combining these two types of data structures, a tradeoff between the size and the query time can be obtained [10, 82, 203]. Actually, the approach described in these papers is very general and works for any geometric-searching data structure that can be viewed as a decomposition scheme (described in Section 2), provided it satisfies certain assumptions. We state the general result here, though one can obtain a slightly better bounds (by a polylogarithmic factor) by exploiting special properties of the data structures.

It will be convenient to regard range-searching data structures in the following abstract form, previously described at the end of Section 2. Let $\mathcal{P}$ be a $d$-dimensional range-searching problem and $\mathcal{D}$ a decomposition scheme for $\mathcal{P}$. That is, for a given set $S$ of $n$ points in $\mathbb{R}^d$, $\mathcal{D}$ constructs a family (multiset) $\mathcal{F} = \mathcal{F}(S)$ of canonical subsets. For a query range $\gamma$, the query procedure implicitly computes a sub-family $\mathcal{C}_\gamma = \mathcal{C}(\gamma, S) \subseteq \mathcal{F}$ that partitions $\gamma \cap S$ into canonical subsets, and returns $\sum_{C \in \mathcal{C}_\gamma} w(C)$.

As we mentioned in Section 2, in order to compute $\mathcal{C}_\gamma$ efficiently, $\mathcal{D}$ must be stored in a hierarchical data structure. We call a decomposition scheme *hierarchical* if $\mathcal{F}$ is stored in a tree $T$. Each node $v$ of $T$ is associated with a canonical subset $C_v \in \mathcal{F}$ and each interior node $v$ satisfies the following property.

(P1) For any query range $\gamma$, there exists a subset $Q(v, \gamma) = \{z_1, \ldots, z_a\}$ of children of $v$ so that $\gamma \cap C_{z_1}, \ldots, \gamma \cap C_{z_a}$ partition $\gamma \cap C_v$.

For example, the linear-size partition trees described in Section 4.1 store a simplex $\Delta_v$ at each node $v$. In these partition trees, a child $z$ of a node $v$ is in $Q(v, \gamma)$, for any query halfspace $\gamma$, if $\Delta_z$ intersects the halfspace $\gamma$.

Property (P1) ensures that, for a node $v$, $w(C_v)$ can be computed by searching only in the subtree rooted at $v$. The query procedure performs a depth-first search on $T$ to compute $\mathcal{C}_\gamma$. Let $\overline{\mathcal{C}}_\gamma = \overline{\mathcal{C}}(\gamma, S)$ denote the canonical subsets in $\mathcal{F}$ associated with nodes visited by the query procedure; clearly, $\mathcal{C}_\gamma \subseteq \overline{\mathcal{C}}_\gamma$.

Let $r \geq 2$ be a parameter and let $\mathcal{D}$ be a hierarchical decomposition scheme. For any $0 \leq i \leq \lceil \log_r n \rceil$, let $\mathcal{F}_i = \{C \in \mathcal{F} \mid r^i \leq |C| < r^{i+1}\}$. We say that $\mathcal{D}$ is *r-convergent* if there exist constants $\alpha \geq 1$ and $0 \leq \beta < 1$ so that the following three conditions hold for all $i$.

(C1) The degree of each node in $T$ is $O(r^\alpha)$.

(C2) $|\mathcal{F}_i| = O\left((n/r^i)^\alpha\right)$.

(C3) For any query range $\gamma$, $|\overline{\mathcal{C}}_\gamma \cap \mathcal{F}_i| = O\left((n/r^i)^\beta\right)$.

The second and third conditions imply that the number of canonical subsets in $\mathcal{D}$ and the the number of subsets in $\overline{\mathcal{C}}_\gamma$, for any query range $\gamma$, decrease exponentially with size.

The size of $\mathcal{D}$ is $O(n^\alpha)$, provided the weight of each canonical subset can be stored in $O(1)$ space, and the query time of $\mathcal{D}$, under the semigroup model, is $O(n^\beta)$ if $\beta > 0$ and $O(\log n)$ if $\beta = 0$. $\mathcal{D}$ is called *efficient* if for any query range $\gamma$, each $\mathcal{C}_\gamma \cap \mathcal{F}_i$ can be computed in time $O\left((n/r^i)^\beta\right)$.

**Theorem 7.** *Let $S$ be a set of $n$ points in $\mathbb{R}^d$, and let $r$ be a sufficiently large constant. Let $\mathcal{P}$ be a range-searching problem. Let $D^1$ be a decomposition scheme for $\mathcal{P}$ of size $O(n^\alpha)$ and query time $O(\log n)$, and let $\mathcal{D}^2$ be another decomposition scheme of size $O(n)$ and query time $O(n^\beta)$. If either $\mathcal{D}^1$ or $\mathcal{D}^2$ is hierarchical, efficient, and $r$-convergent, then for any $n \le m \le n^\alpha$, we can construct a decomposition scheme for $\mathcal{P}$ of size $O(m)$ and query time*

$$O\left(\left(\frac{n^\alpha}{m}\right)^{\beta/(\alpha-1)} + \log\frac{m}{n}\right).$$

**Proof:** Suppose $\mathcal{D}^1$ is hierarchical, efficient, and $r$-convergent. We present a decomposition scheme $\mathcal{D}$ of size $O(m)$. We first define the canonical subsets $\mathcal{F}(S)$ constructed by $\mathcal{D}$ and then define $\mathcal{C}(\gamma, S)$ for each range $\gamma$.

Let $\mathcal{F}^1 = \mathcal{F}^1(S)$ be the family of canonical subsets constructed by $\mathcal{D}^1$ on $S$ and $T^1$ be the corresponding tree. If $\alpha = 1$, we can take $\mathcal{D} = \mathcal{D}^1$, so assume that $\alpha > 1$. We define a parameter

$$\tau = 1 + \left\lceil \frac{\log_r(n^\alpha/m)}{\alpha - 1} \right\rceil.$$

Informally, to construct $\mathcal{F}$, we discard all nodes in $T^1$ whose parents are associated with canonical subsets of size less than $r^\tau$. Then we replace the deleted subsets by constructing, for for every leaf $z$ of the pruned tree, the canonical subsets $\mathcal{F}^2(C_z)$ using the second decomposition scheme $\mathcal{D}^2$. See Figure 7.

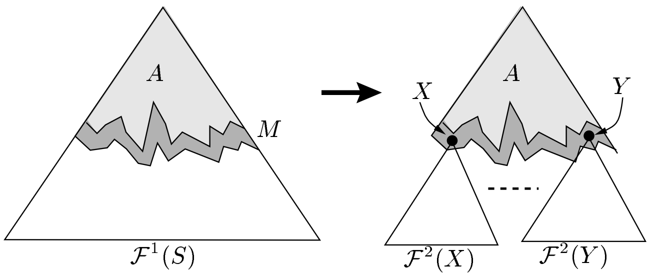


**Figure 7.** The general space query-time tradeoff scheme.

More formally, let $A = \bigcup_{i \geq \tau} \mathcal{F}_i^1$, and let $M \subseteq \mathcal{F}^1 \setminus A$ be the set of canonical subsets whose predecessors lie in $A$. Since $\mathcal{D}^1$ is $r$-convergent,

$$|A| = \sum_{i \geq \tau} |\mathcal{F}_i^1| = \sum_{i \geq \tau} O\left(\left(\frac{n}{r^i}\right)^\alpha\right) = O\left(\left(\frac{n}{r^\tau}\right)^\alpha\right).$$

The degree of each canonical subset in $\mathcal{F}^1$ is $O(r^\alpha)$, so

$$|M| = O(r^\alpha) \cdot |A| = O\left(\left(\frac{n}{r^{\tau-1}}\right)^\alpha\right).$$

For each canonical subset $C \in M$, we compute $\mathcal{F}^2(C)$ using the second decomposition scheme $\mathcal{D}^2$. The size of each subset in $M$ is at most $r^{\tau-1}$, so $|\mathcal{F}^2(C)| = O(r^{\tau-1})$. Set

$$\mathcal{F}(S) = A \cup \bigcup_{C \in M} \mathcal{F}^2(C).$$

The total number of canonical subsets in $\mathcal{F}(S)$ is

$$\begin{aligned} |A| + \sum_{C \in M} \left|\mathcal{F}^2(C)\right| &= O\left(\frac{n^\alpha}{r^{\tau\alpha}}\right) + O\left(\frac{n^\alpha}{r^{(\tau-1)\alpha}}\right) \cdot O(r^{\tau-1}) \\ &= O\left(\frac{n^\alpha}{r^{(\tau-1)(\alpha-1)}}\right) = O(m). \end{aligned}$$

For a query range $\gamma$, let $M_\gamma = M \cap \overline{\mathcal{C}}^1(\gamma, S)$ and $A_\gamma = A \cap \mathcal{C}^1(\gamma, S)$. We now define $\mathcal{C}(\gamma, S)$ as follows.

$$\mathcal{C}(\gamma, S) = A_\gamma \cup \bigcup_{C \in M_\gamma} \mathcal{C}^2(\gamma, C).$$

It can be shown that $\mathcal{C}(\gamma, S)$ forms a partition of $\gamma \cap S$. Since $\mathcal{D}^1$ is efficient, $A_\gamma$ and $M_\gamma$ can be computed in time $O(\log(n/r^\tau)) = O(\log(m/n))$. The size of each canonical subset $C \in M_\gamma$ is at most $r^{\tau-1}$; therefore, each $\mathcal{C}^2(\gamma, C)$ can be computed in time $O(r^{\beta(\tau-1)}) = O((n^\alpha/m)^{\beta/(\alpha-1)})$. By condition (C3), $|M_\gamma| = O(1)$, so the overall query time is

$$O\left(\left(\frac{n^\alpha}{m}\right)^{\beta/(\alpha-1)} + \log\frac{m}{n}\right),$$

as desired.

A similar approach can be used to construct $\mathcal{D}$ if $\mathcal{D}^2$ is $r$-convergent and efficient. We omit further details. □

For the $d$-dimensional simplex range-counting problem, for example, we have $\alpha = d + \varepsilon$ and $\beta = 1 - 1/d$. Thus, we immediately obtain the following space query-time tradeoff.

**Corollary 8.** *For any $n \le m \le n^{d+\varepsilon}$, a simplex range-counting query can be answered in time $O(n^{1+\varepsilon/d}/m^{1/d} + \log(m/n))$ using $O(m)$ space.*

We conclude this section by making a few remarks on Theorem 7.

(i) Theorem 7 can be refined to balance polylogarithmic factors in the sizes and query times of $\mathcal{D}^1$ and $\mathcal{D}^2$. For example, if the size of $\mathcal{D}^1$ is $O(n^\alpha \operatorname{polylog} n)$ and rest of the parameters are the same as in the theorem, then the query time of the new data structure is

$$O\left(\left(\frac{n^\alpha}{m}\right)^{\beta/(\alpha-1)} \operatorname{polylog}\left(\frac{m}{n}\right)\right).$$

Using a similar argument, Matoušek [203] showed that a simplex range-counting query can be answered in time $O((n/m^{1/d})\log^{d+1}(m/n))$, which improves Corollary 8 whenever $m = O(n^d)$.

(ii) Theorem 7 is quite general and holds for any decomposable geometric searching problem as long as there exists an efficient, $r$-convergent decomposition scheme for the problem. We will discuss some such results in the next two sections.

(iii) Theorem 7 actually holds under weaker assumptions on $\mathcal{D}^1$ and $\mathcal{D}^2$. For example, even though halfspace range-reporting data structures do not fit in the above framework, they nevertheless admit a tradeoff. In particular, a halfspace reporting query in $\mathbb{R}^d$ can be answered in $O((n \operatorname{polylog} n)/m^{1/\lfloor d/2 \rfloor} + k)$ using $O(m)$ space.

(iv) Finally, it is not essential for $\mathcal{D}_1$ or $\mathcal{D}_2$ to be tree-based data structures. It is sufficient to have an efficient, $r$-convergent decomposition scheme with a partial order on the canonical subsets, where each canonical subset satisfies a property similar to (P1).

### 4.4 Lower bounds

Fredman [134] showed that a sequence of $n$ insertions, deletions, and halfplane queries on a set of points in the plane requires $\Omega(n^{4/3})$ time, in the semigroup model. His technique, however, does not extend to static data structures. In a series of papers, Chazelle has proved nontrivial lower bounds on the complexity of online simplex range searching, using various elegant mathematical techniques. The following theorem is perhaps the most interesting result on lower bounds.

**Theorem 9** (Chazelle [59]). *Let $n, m$ be positive integers such that $n \le m \le n^d$, and let $S$ be a random set of points in $[0,1]^d$. If only $m$ units of storage are available, then with high probability, the worst-case query time for a simplex range query in $S$ is $\Omega(n/\sqrt{m})$ for $d = 2$, or $\Omega(n/(m^{1/d} \log n))$ for $d \ge 3$, in the semigroup model.*

It should be pointed out that this theorem holds even if the query ranges are wedges or strips, but not if the ranges are hyperplanes. Chazelle and Rosenberg [81] proved a lower bound of $\Omega(n^{1-\varepsilon}/m + k)$ for simplex range reporting under the pointer-machine model. These lower bounds do not hold for halfspace range searching. A somewhat weaker lower bound for halfspace queries was proved by Brönnimann*et al.* [52].

As we saw earlier, faster data structures are known for halfspace emptiness queries. A recent series of papers by Erickson established the first nontrivial lower bounds for online and offline emptiness query problems, in the partition-graph model of computation. His techniques were first applied to Hopcroft's problem — Given a set of $n$ points and $m$ lines, does any point lie on a line? — for which he obtained a lower bound of $\Omega(n \log m + n^{2/3}m^{2/3} + m \log n)$ [118], almost matching the best known upper bound $O(n \log m + n^{2/3}m^{2/3}2^{O(\log^*(n+m))} + m \log n)$, due to Matoušek [203]. Slightly better lower bounds are known for higher-dimensional versions of Hopcroft's problem [118, 117], but for the special case $n = m$, the best known lower bound is still only $\Omega(n^{4/3})$, which is quite far from the best known upper bound $O(n^{2d/(d+1)}2^{O(\log^* n)})$. More recently, Erickson established tradeoff lower bounds between space and query time, or preprocessing and query time, for online hyperplane emptiness queries [119]. The space-time tradeoffs are established by showing that a partition graph that supports hyperplane emptiness queries also (implicitly) supports halfspace semigroup queries, and then applying the lower bounds of Brönnimann *et al.*[52]. For $d$-dimensional hyperplane queries, $\Omega(n^d/\operatorname{polylog} n)$ preprocessing time is required to achieve polylogarithmic query time, and the best possible query time is $\Omega(n^{1/d}/\operatorname{polylog} n)$ if only $O(n \operatorname{polylog} n)$ preprocessing time is allowed. More generally, in two dimensions, if the preprocessing time is $p$, the query time is $\Omega(n/\sqrt{p})$. Erickson's techniques also imply nontrivial lower bounds for online and offline halfspace emptiness searching, but with a few exceptions, these are quite weak.

Table 4 summarizes the best known lower bounds for online simplex queries, and Table 5 summarizes the best known lower bounds for offline simplex range searching. Lower bounds for emptiness problems apply to counting and reporting problems as well. No nontrivial lower bound was known for any offline range searching problem under the group model until Chazelle's result [64].

See the survey papers [63, 204] for a more detailed discussion on lower bounds.

# 5 Variants and Extensions

In this section we review some extensions of range-searching data structures, including multi-level data structures, semialgebraic range searching, and dynamization. As in the previous section, the preprocessing time for each of the data structures we describe is at most a polylogarithmic or $n^\varepsilon$ factor larger than its size.

| Range | Problem | Model | Query Time | Source |
|---|---|---|---|---|
| Simplex | Semigroup | Semigroup ($d = 2$) | $\frac{n}{\sqrt{m}}$ | [59] |
| | Semigroup | Semigroup ($d > 2$) | $\frac{n}{m^{1/d} \log n}$ | [59] |
| | Reporting | Pointer machine | $\frac{n^{1-\varepsilon}}{m^{1/d}} + k$ | [81] |
| Hyperplane | Semigroup | Semigroup | $\left(\frac{n}{m^{1/d}}\right)^{2/(d+1)}$ | [119] |
| | Emptiness | Partition graph | $\left(\frac{n}{\log n}\right)^{\frac{d^2+1}{d^2+d}} \cdot \frac{1}{m^{1/d}}$ | [119] |
| Halfspace | Semigroup | Semigroup | $\left(\frac{n}{\log n}\right)^{\frac{d^2+1}{d^2+d}} \cdot \frac{1}{m^{1/d}}$ | [52] |
| | Emptiness | Partition graph | $\left(\frac{n}{\log n}\right)^{\frac{\delta^2+1}{\delta^2+\delta}} \cdot \frac{1}{m^{1/\delta}}$, where $d \geq \delta(\delta+3)/2$ | [119] |

**Table 4.** Asymptotic lower bounds for online simplex range searching using $O(m)$ space.

| Range | Problem | Model | Time | Source |
|---|---|---|---|---|
| Halfspace | Emptiness | Algebraic computation tree | $n \log n$ | [40] |
| | | Partition graph ($d \leq 4$) | $n \log n$ | [117] |
| | | Partition graph ($d \geq 5$) | $n^{4/3}$ | [117] |
| | Counting | Partition graph | $n^{4/3}$ | [118] |
| | Group | Group (with $n/2$ help gates) | $n \log n$ | [64] |
| Hyperplane | Emptiness | Algebraic computation tree | $n \log n$ | [40] |
| | | Partition graph | $n^{4/3}$ | [118] |
| | Semigroup | Semigroup | $n^{4/3}$ | [65, 118] |
| Simplex | Semigroup | Semigroup | $\frac{n^{2-2/(d-1)}}{\log^{5/2} n}$ | [65] |

**Table 5.** Asymptotic lower bounds for offline simplex range searching.

### 5.1 Multi-level data structures

A rather powerful property of data structures based on decomposition schemes (described in Section 2) is that they can be cascaded together to answer more complex queries, at the increase of a logarithmic factor in their performance. This property has been implicitly used for a long time; see, for example, [112, 192, 196, 290, 252]. The real power of the cascading property was first observed by Dobkin and Edelsbrunner [102], who used this property to answer several complex geometric queries. Since their result, several papers have exploited and extended this property to solve numerous geometric-searching problems; see [10, 149, 274, 203, 238]. In this subsection we briefly sketch the general cascading scheme, as described in [203].

Let $S$ be a set of weighted objects. Recall that a geometric-searching problem $\mathcal{P}$, with underlying relation $\Diamond$, requires computing $\sum_{p \Diamond \gamma} w(p)$ for a query range $\gamma$. Let $\mathcal{P}^1$ and $\mathcal{P}^2$ be two geometric-searching problems, and let $\Diamond^1$ and $\Diamond^2$ be the corresponding relations. Then we define $\mathcal{P}^1 \circ \mathcal{P}^2$ to be the conjunction of $\mathcal{P}^1$ and $\mathcal{P}^2$, whose relation is $\Diamond^1 \cap \Diamond^2$. That is, for a query range $\gamma$, we want to compute $\sum_{p \Diamond^1 \gamma, p \Diamond^2 \gamma} w(p)$. Suppose we have hierarchical decomposition schemes $\mathcal{D}^1$ and $\mathcal{D}^2$ for problems $\mathcal{P}^1$ and $\mathcal{P}^2$. Let $\mathcal{F}^1 = \mathcal{F}^1(S)$ be the set of canonical subsets constructed by $\mathcal{D}^1$, and for a range $\gamma$, let $\mathcal{C}^1_\gamma = \mathcal{C}^1(S, \gamma)$ be the corresponding partition of $\{p \in S \mid p \Diamond^1 \gamma\}$ into canonical subsets. For each canonical subset $C \in \mathcal{F}^1$, let $\mathcal{F}^2(C)$ be the collection of canonical subsets of $C$ constructed by $\mathcal{D}^2$, and let $\mathcal{C}^2(C, \gamma)$ be the corresponding partition of $\{p \in C \mid p \Diamond^2 \gamma\}$ into level-two canonical subsets. The decomposition scheme $\mathcal{D}^1 \circ \mathcal{D}^2$ for the problem $\mathcal{P}^1 \circ \mathcal{P}^2$ consists of the canonical subsets $\mathcal{F} = \bigcup_{C \in \mathcal{F}^1} \mathcal{F}^2(C)$. For a query range $\gamma$, the query output is $\mathcal{C}_\gamma = \bigcup_{C \in \mathcal{C}^1_\gamma} \mathcal{C}^2(C, \gamma)$. Note that we can cascade any number of decomposition schemes in this manner.

If we view $\mathcal{D}^1$ and $\mathcal{D}^2$ as tree data structures, then cascading the two decomposition schemes can be regarded as constructing a two-level tree, as follows. We first construct the tree induced by $\mathcal{D}^1$ on $S$. Each node $v$ of $\mathcal{D}^1$ is associated with a canonical subset $C_v$. We construct a second-level tree $\mathcal{D}^2_v$ on $C_v$ and store $\mathcal{D}^2_v$ at $v$ as its secondary structure. A query is answered by first identifying the nodes that correspond to the canonical subsets $C_v \in \mathcal{C}^1_\gamma$ and then searching the corresponding secondary trees to compute the second-level canonical subsets $\mathcal{C}^2(C_v, \gamma)$.

The range tree, defined in Section 3.1, fits in this framework. For example, a two dimensional range tree is obtained by cascading two one-dimensional range trees, as follows. Let $S$ be a set of $n$ weighted points and $\mathcal{R}$ the set of all orthogonal rectangles in the plane. We define two binary relations $\Diamond^1$ and $\Diamond^2$, where for any rectangle $\gamma = [\alpha_1, \beta_1] \times [\alpha_2, \beta_2]$, $p \Diamond^i \gamma$ if $x_i(p) \in [\alpha_i, \beta_i]$. Let $\mathcal{P}^i$ be the searching problem associated with $\Diamond^i$, and let $\mathcal{D}^i$ be the data structure corresponding to $\mathcal{P}^i$. Then the two-dimensional orthogonal range-searching problem is the same as $\mathcal{P}^1 \circ \mathcal{P}^2$. We can therefore cascade $\mathcal{D}^1$ and $\mathcal{D}^2$, as described above, to answer a two-dimensional orthogonal range-searching query. Similarly, a data structure for $d$-dimensional simplex range-searching can be constructed by cascading $d+1$ halfspace range-searching structures, since a $d$-simplex is an intersection of at most $d+1$ halfspaces. Multi-level data structures were also proposed for *range restriction*, introduced by Willard and Lueker [290] and Scholten and Overmars [252].

The following theorem, whose straightforward proof we omit, states a general result for multi-level data structures.

**Theorem 10.** *Let $S, \mathcal{P}^1, \mathcal{P}^2, \mathcal{D}^1, \mathcal{D}^2$ be as defined above, and let $r$ be a constant. Suppose the size and query time of each decomposition scheme are at most $S(n)$ and $Q(n)$, respectively. If $\mathcal{D}^1$ is efficient and $r$-convergent, then we obtain a hierarchical decomposition scheme $\mathcal{D}$ for $\mathcal{P}^1 \circ \mathcal{P}^2$ whose size and query time are $O(S(n)\log_r n)$ and $O(Q(n)\log_r n)$. If $\mathcal{D}^2$ is also efficient and $r$-convergent, then $\mathcal{D}$ is also efficient and $r$-convergent.*

In some cases, the added logarithmic factor in the query time or the space can be saved. The real power of multi-level data structures stems from the fact that there are no restrictions on the relations $\Diamond^1$ and $\Diamond^2$. Hence, any query that can be represented as a conjunction of a constant number of "primitive" queries, each of which admits an efficient, $r$-convergent decomposition scheme, can be answered by cascading individual decomposition schemes. We will describe a few multi-level data structures in this and the following sections.

## 5.2 Semialgebraic range searching

So far we assumed that the ranges were bounded by hyperplanes, but many applications involve ranges bounded by nonlinear functions. For example, a query of the form "For a given point $p$ and a real number $r$, find all points of $S$ lying within distance $r$ from $p$" is a range-searching problem in which ranges are balls.

As shown below, range searching with balls in $\mathbb{R}^d$ can be formulated as an instance of halfspace range searching in $\mathbb{R}^{d+1}$. So a ball range-reporting (resp. range-counting) query in $\mathbb{R}^d$ can be answered in time $O((n/m^{1/\lceil d/2\rceil})\,\mathrm{polylog}\, n+k)$ (resp. $O((n/m^{1/(d+1)})\log(m/n))$), using $O(m)$ space. (Somewhat better performance can be obtained using a more direct approach, which we will describe shortly.) However, relatively little is known about range-searching data structures for more general ranges.

A natural class of more general ranges is the family of Tarski cells. A *Tarski cell* is a real semialgebraic set defined by a constant number of polynomials, each of constant degree. In fact, it suffices to consider the ranges bounded by a single polynomial because the ranges bounded by multiple polynomials can be handled using multi-level data structures. We assume that the ranges are of the form

$$\gamma_f(a) = \{x \in \mathbb{R}^d \mid f(a,x) \geq 0\},$$

where $f$ is a $(d+b)$-variate polynomial specifying the type of range (disks, cylinders, cones, etc.), and $a$ is a $b$-tuple specifying a specific range of the given type (*e.g.*, a specific disk). Let $\Gamma_f = \{\gamma_f(a) \mid a \in \mathbb{R}^b\}$. We will refer to the range-searching problem in which the ranges are from the set $\Gamma_f$ as the *$\Gamma_f$-range searching.*

One approach to answer $\Gamma_f$-range queries is to use *linearization*, originally proposed by Yao and Yao [293]. We represent the polynomial $f(a,x)$ in the form

$$f(a,x) = \psi_0(a)\varphi_0(x) + \psi_1(a)\varphi_1(x) + \cdots + \psi_\ell(a)\varphi_\ell(x)$$

where $\varphi_0, \ldots, \varphi_\ell, \psi_0, \ldots, \psi_\ell$ are polynomials. A point $x \in \mathbb{R}^d$ is mapped to the point

$$\varphi(x) = [\varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots, \varphi_\ell(x)] \in \mathbb{R}^\ell,$$

represented in homogeneous coordinates. Then each range $\gamma_f(a) = \{x \in \mathbb{R}^d \mid f(x,a) \geq 0\}$ is mapped to a halfspace

$$\psi^{\#}(a) : \{y \in \mathbb{R}^\ell \mid \psi_0(a)y_0 + \psi_1(a)y_1 + \cdots + \psi_\ell(a)y_\ell \geq 0\},$$

where, again, $[y_0, y_1, \ldots, y_\ell]$ are homogeneous coordinates.

The constant $\ell$ is called the *dimension* of the linearization. The following algorithm, based on an algorithm of Agarwal and Matoušek [8], computes a linearization of smallest dimension.[7] Write the polynomial $f(a,x)$ as the sum of monomials

$$f(a,x) = \sum_{\mu \in M} \sum_{\nu \in N} c_{\mu,\nu} a^\mu x^\nu,$$

where $M \subset \mathbb{N}^b$ and $N \subset \mathbb{N}^d$ are finite sets of exponent vectors, $c_{\mu,\nu}$ are real coefficients, and $a^\mu$ and $x^\nu$ are shorthand for the monomials $a_1^{\mu_1} a_2^{\mu_2} \ldots a_d^{\mu_b}$ and $x_1^{\nu_1} x_2^{\nu_2} \ldots x_d^{\nu_d}$, respectively. Collect the coefficients $c_{\mu,\nu}$ into a matrix $C$ whose rows are indexed by elements of $M$ (*i.e.*, monomials in $a$) and whose columns are indexed by elements of $N$ (*i.e.*, monomials in $x$). The minimum dimension of linearization is one less than the rank of this matrix. The polynomials $\varphi_i(x)$ and $\phi_j(a)$ are easily extracted from any basis of the vector space spanned by either the rows or columns of the coefficient matrix $C$.

For example, a disk with center $(a_1, a_2)$ and radius $a_3$ in the plane can be regarded as a set of the form $\gamma_f(a)$, where $a = (a_1, a_2, a_3)$ and $f$ is a 5-variate polynomial

$$f(a_1, a_2, a_3; x_1, x_2) = -(x_1 - a_1)^2 - (x_2 - a_2)^2 + a_3^2$$

This polynomial has the following coefficient matrix.

| | $1$ | $x_1$ | $x_2$ | $x_1^2$ | $x_2^2$ |
|---|---|---|---|---|---|
| $1$ | 0 | 0 | 0 | $-1$ | $-1$ |
| $a_1$ | 0 | 2 | 0 | 0 | 0 |
| $a_2$ | 0 | 0 | 2 | 0 | 0 |
| $a_1^2$ | $-1$ | 0 | 0 | 0 | 0 |
| $a_2^2$ | $-1$ | 0 | 0 | 0 | 0 |
| $a_3^2$ | 1 | 0 | 0 | 0 | 0 |

This matrix has rank 4, so the linearization dimension of $f$ is 3. One possible linearization is given by the following set of polynomials:

$$\begin{array}{llll} \psi_0(a) = -a_1^2 - a_2^2 + a_3^2, & \psi_1(a) = 2a_1, & \psi_2(a) = 2a_2, & \psi_3(a) = -1, \\ \varphi_0(x) = 1, & \varphi_1(x) = x_1, & \varphi_2(x) = x_2, & \varphi_3(x) = x_1^2 + x_2^2. \end{array}$$

[7]In some cases, Agarwal and Matoušek's algorithm returns a dimension one higher than the true minimum, since they consider only linearizations with $\psi_0(a) = 1$.

In general, balls in $\mathbb{R}^d$ admit a linearization of dimension $d+1$; cylinders and other quadrics in $\mathbb{R}^3$ admit a linearization of dimension 9. One of the most widely used linearizations in computational geometry uses the so-called *Plücker coordinates*, which map a line in $\mathbb{R}^3$ to a point in $\mathbb{R}^5$; see [73, 265, 268] for more details on Plücker coordinates.

A $\Gamma_f$-range query can now be answered using a $\ell$-dimensional halfspace range-searching data structure. Thus, for counting queries, we immediately obtain a linear-size data structure with query time $O(n^{1-1/\ell})$ [203], or a data structure of size $O(n^\ell/\log^\ell n)$ with logarithmic query time [62]. When $d < \ell$, the performance of the linear-size data structures can be improved by exploiting the fact that the points $\varphi(x)$ have only $d$ degrees of freedom. Using results of Aronov *et al.* [23] on the size of the zone of an algebraic variety in a $k$-dimensional hyperplane arrangement, Agarwal and Matoušek [8] show that the query time for a linear-space data structure can be reduced to $O(n^{1-1/\lfloor(d+\ell)/2\rfloor+\varepsilon})$. It is an open problem whether one can similarly exploit the fact that the halfspaces $\psi^\#(a)$ have only $b$ degrees of freedom to reduce the size of data structures with logarithmic query time when $b < \ell$.

In cases where the linearization dimension is very large, semialgebraic queries can also be answered using the following more direct approach proposed by Agarwal and Matoušek [8]. Let $S$ be a set of $n$ points in $\mathbb{R}^d$. For each point $p_i$, we can define a $b$-variate polynomial $g_i(a) \equiv f(p_i, a)$. Then $\Gamma_f(a) \cap S$ is the set of points $p_i$ for which $g_i(a) \geq 0$. Hence, the problem reduces to point location in the arrangement of algebraic surfaces $g_i = 0$ in $\mathbb{R}^b$. Let $G$ be the set of resulting surfaces. The following result of Chazelle *et al.* [70, 71] leads to a point-location data structure.

**Theorem 11 (Chazelle *et al.* [70]).** *Let $\mathcal{F} = \{f_1, \ldots, f_n\}$ be a set of $n$ $d$-variate polynomials, with $d \geq 3$, where each $f_i$ has maximum degree $\delta$ in any variable. Then $\mathbb{R}^d$ be partitioned into a set $\Xi$ of $O\big(n^{2d-3}2^{\alpha(n)^{(2\delta)^{2^d-1}}}\big)$ Tarski cells so that the sign of each $f_i$ remains the same for all points within each cell of $\Xi$. Moreover, $\Xi$ can be computed in $O(n^{2d-1}\log n)$ time.*

Improving the combinatorial upper bound in Theorem 11 is an open problem. The best known lower bound is $\Omega(n^d)$, and this is generally believed to be the right bound. Any improvement would also improve the bounds for the resulting semialgebraic range searching data structures.

Returning to the original point-location problem for $g_i$'s, using this theorem and results on $\varepsilon$-nets and cuttings, $G$ can be preprocessed into a data structure of size $O(n^{2b-3+\varepsilon})$ if $b \geq 3$, or $O(n^{2+\varepsilon})$ if $b = 2$, so that for a query point $a \in \mathbb{R}^b$, we can compute $\sum_{g_i(a)\geq 0} w(p_i)$ in $O(\log n)$ time.

Using Theorem 11, Agarwal and Matoušek [8] also extended Theorem 3 to Tarski cells and showed how to construct partition trees using this extension, obtaining a linear-size data structure with query time $O(n^{1-1/\gamma+\varepsilon})$, where $\gamma = 2$ if $d = 2$ and $\gamma = 2d-3$ if $d \geq 3$.

As in Section 4.3, the best data structures with linear space and logarithmic query time can be combined to obtain the following tradeoff between space and query time.

**Theorem 12.** *Let $f : \mathbb{R}^d \times \mathbb{R}^b \to \mathbb{R}$ be a $(d+b)$-variate polynomial with linearization dimension $\ell$. Let $\lambda = \min(2d-3, \lfloor (d+\ell)/2 \rfloor, \ell)$, and let $\gamma = \min(2b-3, \ell)$. For any $n \le m \le n^\gamma$, we can build a data structure of size $O(m)$ that supports $\Gamma_f$-counting queries in time*

$$O\left(\left(\frac{n^\gamma}{m}\right)^{(\lambda-1)/\lambda(\gamma-1)+\varepsilon} + \log\frac{m}{n}\right).$$

For example, if our ranges are balls in $\mathbb{R}^d$, we have $b = d+1$, $\ell = d+1$, $\lambda = d$, and $\gamma = d+1$, so we can answer queries in time $O((n^{d+1}/m)^{(d-1)/d^2+\varepsilon} + \log(m/n))$ using space $O(m)$.

### 5.3 Dynamization

All the data structures discussed above assumed $S$ to be fixed, but in many applications one needs to update $S$ dynamically — insert a new point into $S$ or delete a point from $S$. We cannot hope to perform insert/delete operations on a data structure in less than $P(n)/n$ time, where $P(n)$ is the preprocessing time of the data structure. If we allow only insertions (i.e., a point cannot be deleted from the structure), static data structures can be modified using standard techniques [44, 211, 230], so that a point can be inserted in time $O(P(n)\log n/n)$ and a query can be answered in time $O(Q(n)\log n)$, where $Q(n)$ is the query time of the original static data structure. Roughly speaking, these techniques proceed as follows. Choose a parameter $r \ge 2$ and set $t = \lceil \log_r n \rceil$. Maintain a partition of $S$ into $t$ subsets $S_0, \ldots S_{t-1}$ so that $|S_i| \le (r-1)r^i$, and preprocess each $S_i$ for range searching separately. We call a subset $S_i$ *full* if $|S_i| = (r-1)r^i$. A query is answered by computing $w(S_i \cap \gamma)$ for each subset $S_i$ independently and then summing them up. The total time spent in answering a query is thus $O(t + \sum_{i=1}^{t} Q(r^i))$. Suppose we want to insert a point $p$. We find the least index $i$ such that the subsets $S_0, \ldots, S_{i-1}$ are full. Then we add the point $p$ and $\bigcup_{j<i} S_j$ to $S_i$, set $S_j = \emptyset$ for all $j < i$, and preprocess the new $S_i$ for range searching. The amortized insertion time is $O(\sum_{i=0}^{t-1} P(r^i)/r^i)$. We can convert this amortized behavior into a worst-case performance using known techniques [263]. In some cases the logarithmic overheard in the query or update time can be avoided.

Although the above technique does not handle deletions, many range-searching data structures, such as orthogonal and simplex range-searching structures, can handle deletions at polylogarithmic or $n^\varepsilon$ overhead in query and update time, by exploiting the fact that a point is stored at roughly $S(n)/n$ nodes [10]. Table 6 summarizes the known results on dynamic 2D orthogonal range-searching data structures; these results can be extended to higher dimensions at a cost of an additional $\log^{d-2} n$ factor in the storage, query time, and update time. Klein *et al.* [184] have described an optimal data structure for a special case of 2D range-reporting in which the query ranges are translates of a polygon.

Although Matoušek's $O(n\log n)$-size data structure for $d$-dimensional halfspace range reporting [199] can be dynamized, the logarithmic query time data structure is not easy to dynamize because some of the points may be stored at $\Omega(n^{\lfloor d/2 \rfloor})$ nodes of the tree. Agarwal

| Problem | Size | Query Time | Update Time | Source |
|---|---|---|---|---|
| Counting | $n$ | $\log^2 n$ | $\log^2 n$ | [58] |
| Reporting | $n$ | $k \log^2(2n/k)$ | $\log^2 n$ | [58] |
| | $n$ | $n^\varepsilon + k$ | $\log^2 n$ | [264] |
| | $n \log n$ | $\log n \log\log n + k$ | $\log n \log\log n$ | [212] |
| | $\dfrac{n \log n}{\log\log n}$ | $\dfrac{\log^{2+\varepsilon} n}{\log\log n} + k$ | $\dfrac{\log^2 n}{\log\log n}$ | [264] |
| Semigroup | $n$ | $\log^4 n$ | $\log^4 n$ | [58] |

**Table 6.** Asymptotic upper bounds for dynamic 2D orthogonal range-searching.

and Matoušek [9] developed a rather sophisticated data structure that can insert or delete a point in time $O(n^{\lfloor d/2 \rfloor - 1 + \varepsilon})$ time and can answer a query in $O(\log n + k)$ time. As in [82], at each node of the tree, this structure computes a family of partitions (instead of a single partition), each of size $O(r^{\lfloor d/2 \rfloor})$ for some parameter $r$. For every shallow hyperplane $h$, there is at least one partition so that $h$ intersects $O(r^{\lfloor d/2 \rfloor - 1})$ simplices of the partition.

Grossi and Italiano [148], generalizing and improving earlier results of van Kreveld and Overmars [275, 276], describe dynamic $d$-dimensional orthogonal range searching data structures that also support *split* and *merge* operations, defined as follows. Given a point set $S$, a point $p \in S$, and an integer $i$ between 1 and $d$, a split operation divides $S$ into two disjoint subsets $S_1, S_2$ separated by the hyperplane normal the $x_i$-axis passing through $p$, and splits the data structure for $S$ into data structures for $S_1$ and $S_2$. Given two point sets $S_1$ and $S_2$ separated by a hyperplane normal to some coordinate axis, the merge operation combines the data structures for $S_1$ and $S_2$ into a single data structure for their union $S_1 \cup S_2$. Grossi and Italiano's data structure, called a *cross tree*, requires linear space and $O(n \log n)$ preprocessing time and supports insertions and deletions in time $O(\log n)$; splits, merges, and counting queries in time $O(n^{1-1/d})$; and reporting queries in time $O(n^{1-1/d} + k)$. Their technique gives efficient solutions to many other order-decomposable problems involving split and merge operations, including external-memory range searching.

Since an arbitrary sequence of deletions is difficult to handle in general, researchers have examined whether a random sequence of insertions and deletions can be handled efficiently; see [221, 222, 254]. Mulmuley [221] proposed a reasonably simple data structure for halfspace range reporting that can process a random update sequence of length $m$ in expected time $O(m^{\lfloor d/2 \rfloor + \varepsilon})$ and can answer a query in time $O(k \log n)$. If the sequence of insertions, deletions, and queries is known in advance, the corresponding static data structures can be modified to handle such a sequence of operations by paying a logarithmic overhead in the query time [113]. These techniques work even if the sequence of insertions and queries is not known in advance, but the deletion time of a point is known when it is inserted [106]; see also [263]. See the survey paper by Chiang and Tamassia [86] for a more detailed review of dynamic geometric data structures.

## 6 Intersection Searching

A general intersection-searching problem can be formulated as follows. Given a set $S$ of objects in $\mathbb{R}^d$, a semigroup $(\mathbf{S}, +)$, and a weight function $w : S \to \mathbf{S}$, we wish to preprocess $S$ into a data structure so that for a query object $\gamma$, we can compute the weighted sum $\sum_{p \cap \gamma \neq \emptyset} w(p)$, where the sum is taken over all objects $p \in S$ that intersect $\gamma$. Range searching is a special case of intersection searching in which $S$ is a set of points. Just as with range searching, there are several variations on intersection searching: intersection counting ("How many objects in $S$ intersect $\gamma$?"), intersection detection ("Does any object in $S$ intersect $\gamma$?"), intersection reporting ("Which objects in $S$ intersect $\gamma$?"), and so on.

Intersection searching is a central problem in a variety of application areas such as robotics, geographic information systems, VLSI, databases, and computer graphics. For example, the *collision-detection* problem — Given a set $O$ of obstacles and a robot $B$, determine whether a placement $p$ of $B$ is free — can be formulated as a point intersection-detection query amid a set of regions. If $B$ has $k$ degrees of freedom, then a placement of $B$ can be represented as a point in $\mathbb{R}^k$, and the set of placements of $B$ that intersect an obstacle $O_i \in m$ is a region $K_i \subseteq \mathbb{R}^k$. If $B$ and the obstacles are semialgebraic sets, then each $K_i$ is also a semialgebraic set. A placement $p$ of $B$ is free if and only if $p$ does not intersect any of $K_i$'s. See [191] for a survey of known results on the collision-detection problem. Another intersection searching problem that arises quite frequently is the *clipping problem*: Preprocess a given set of polygons into a data structure so that all polygons intersecting a query rectangle can be reported efficiently.

An intersection-searching problem can be formulated as a semialgebraic range-searching problem by mapping each object $p \in S$ to a point $\varphi(p)$ in a parametric space $\mathbb{R}^\ell$ and every query range $\gamma$ to a semialgebraic set $\psi^{\#}(\gamma)$ so that $p$ intersects $\gamma$ if and only if $\varphi(p) \in \psi^{\#}(\gamma)$. For example, let $S$ be a set of segments in the plane and the query ranges be also segments in the plane. Each segment $e \in S$ with left and right endpoints $(p_x, p_y)$ and $(q_x, q_y)$, respectively, can be mapped to a point $\varphi(e) = (p_x, p_y, q_x, q_y)$ in $\mathbb{R}^4$ and a query segment $\gamma$ can be mapped to a semialgebraic region $\psi^{\#}(\gamma)$ so that $\gamma$ intersects $e$ if and only if $\varphi(e) \in \psi^{\#}(\gamma)$. Hence, a segment intersection query can be answered by preprocessing the set $\{\varphi(e) \mid e \in S\}$ for semialgebraic searching. A drawback of this approach is that the dimension of the parametric space is typically much larger than $d$, and, therefore, it does not lead to an efficient data structure.

The efficiency of an intersection-searching structure can be significantly improved by expressing the intersection test as a conjunction of simple primitive tests (in low dimensions) and then using a multi-level data structure to perform these tests. For example, a segment $\gamma$ intersects another segment $e$ if the endpoints of $e$ lie on the opposite sides of the line containing $\gamma$ and vice-versa. Suppose we want to report those segments of $S$ whose left endpoints lie below the line supporting a query segment (the other case can be handled in a similar manner). We define three searching problems $\mathcal{P}_1, \mathcal{P}_2$, and $\mathcal{P}_3$, with relations $\Diamond^1, \Diamond^2, \Diamond^3$, as follows:

**$e \Diamond^1 \gamma$:** The left endpoint of $e$ lies below the line supporting $\gamma$.

$e \lozenge^2 \gamma$**:** The right endpoint of $e$ lies above the line supporting $\gamma$.

$e \lozenge^3 \gamma$**:** The line $\ell_e$ supporting $e$ intersects $\gamma$; equivalently, in the dual plane, the point dual to $\ell_e$ lies in the double wedge dual to $e$.

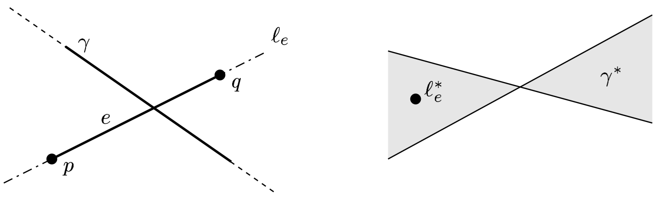


**Figure 8.** Segment intersection searching

For $1 \le i \le 3$, let $\mathcal{D}^i$ denote a data structure for $\mathcal{P}^i$. Then $\mathcal{D}^1$ (resp. $\mathcal{D}^2$) is a halfplane range-searching structure on the left (resp. right) endpoints of segments in $S$, and $\mathcal{D}^3$ is (essentially) a triangle range-searching structure for points dual to the lines supporting $S$. By cascading $\mathcal{D}^1$, $\mathcal{D}^2$, and $\mathcal{D}^3$, we obtain a data structure for segment-intersection queries. Therefore, by Theorem 10, a segment-intersection query can be answered in time $O(n^{1/2+\varepsilon})$ using $O(n \log^3 n)$ space, or in $O(\log^3 n)$ time using $O(n^{2+\varepsilon})$ space; the size in the first data structure and the query time in the second one can be improved to $O(n)$ and $O(\log n)$, respectively. As usual, we can obtain a tradeoff between query time and space using Theorem 7.

It is beyond the scope of this survey paper to cover all intersection-searching problems. Instead, we discuss a few basic problems that have been studied extensively. All intersection-counting data structures described here can also answer intersection-reporting queries at an additional cost that is proportional to the output size. In some cases, an intersection-reporting query can be answered faster. Moreover, using intersection-reporting data structures, intersection-detection queries can be answered in time proportional to their query-search time. Finally, all the data structures described in this section can be dynamized at an expense of $O(n^\varepsilon)$ factor in the storage and query time.

## 6.1 Point intersection searching

Preprocess a set $S$ of objects (such as balls, halfspaces, simplices, or Tarski cells) in $\mathbb{R}^d$ into a data structure so that all the objects of $S$ containing a query point can be reported (or counted) efficiently. This is the inverse or dual of the usual range-searching problem. As discussed in Section 4.2, using the duality transformation, a halfspace range-searching problem can be reduced to a point-intersection problem for a set of halfspaces, and vice versa. In general, as mentioned in Section 5.2, a $d$-dimensional $\Gamma_f$-range searching query, where $f$ is $(d+b)$-variate polynomial, can be viewed as a $b$-dimensional point-intersection searching problem. Therefore, a very close relationship exists between the data structures for range searching (including orthogonal range searching) and for point-intersection searching. Point intersection queries can also be viewed as locating a point in the subdivision of $\mathbb{R}^d$ induced by the objects in $S$.

Suppose the objects in $S$ are semialgebraic sets of the form $\{x \in \mathbb{R}^d \mid f_1(x) \geq 0, \ldots, f_u(x) \geq 0\}$, where each $f_i$ is a $(d+b)$-variate polynomial of bounded degree that admits a linearization of dimension at most $\ell$. Let $\lambda = \min(\ell, 2d-3)$ and $\gamma = \min(2b-3, \lfloor (b+\ell)/2 \rfloor, \ell)$. By constructing a multi-level data structure, point-intersection queries for $S$ can be answered in time $O(\log n)$ using $O(n^{\lambda+\varepsilon})$ space, or in time $O(n^{1-1/\gamma+\varepsilon})$ using $O(n)$ space. Once again, we can obtain a space-time tradeoff, similar to Theorem 12. Table 7 gives some of the specific bounds that can be attained using this general scheme.

| $d$ | Objects | Problem | Size | Query Time | Source |
|---|---|---|---|---|---|
| $d=2$ | Disks | Counting | $m$ | $(n^{4/3}/m^{2/3})\log(m/n)$ | [8] |
| | Disks | Reporting | $n\log n$ | $\log n + k$ | [16] |
| | Triangles | Counting | $m$ | $\frac{n}{\sqrt{m}}\log^3 n$ | [10] |
| | Fat triangles | Reporting | $n\log^2 n$ | $\log^3 n + k$ | [182] |
| | Tarski cells | Counting | $n^{2+\varepsilon}$ | $\log n$ | [71] |
| $d=3$ | Functions | Reporting | $n^{1+\varepsilon}$ | $\log n + k$ | [6] |
| | Fat tetrahedra | Reporting | $m$ | $\frac{n^{1+\varepsilon}}{\sqrt{m}} + k$ | [115] |
| $d\geq 3$ | Simplices | Counting | $m$ | $\frac{n}{m^{1/d}}\log^{d+1} n$ | |
| | Balls | Counting | $n^{d+\varepsilon}$ | $\log n$ | [8] |
| | Balls | Reporting | $m$ | $\frac{n}{m^{1/\lceil d/2\rceil}}\operatorname{polylog} n + k$ | [199] |
| | Tarski cells | Counting | $n^{2d-3+\varepsilon}$ | $\log n$ | [71] |
| | | | $n^{\gamma+\varepsilon}$ | $\log n$ | [8] |

**Table 7.** Asymptotic upper bounds for point intersection searching.

Agarwal *et al.* [6] extended the approach for dynamic halfspace range searching to answer point-intersection queries amid the graphs of bivariate algebraic functions, each of bounded degree. Let $\mathbf{F}$ be an infinite family of bivariate polynomials, each of bounded degree, and let $\Lambda(m)$ denote the maximum size of the lower envelope of a subset of $\mathbf{F}$ of size $m$. Their techniques maintains an $n$-element subset $\mathcal{F} \subseteq \mathbf{F}$ in a data structure of size $O(\Lambda(n) \cdot n^{\varepsilon})$, so that a polynomial $f \in \mathbf{F}$ can be inserted into or deleted from $\mathcal{F}$ in $O(n^{\varepsilon})$ time and, for a query point $p$, all functions of $\mathcal{F}$ whose graphs lie below $p$ can be reported in time $O(\log n + k)$.

Besides the motion-planning application discussed above, point location in an arrangement of surfaces, especially determining whether a query point lies above a given set of regions of the form $x_{d+1} \geq f(x_1, \ldots, x_d)$, has many other applications in computational geometry; see [13, 71, 72] for examples. However, most of these applications call for an offline data structure because the query points are known in advance.

## 6.2 Segment intersection searching

Preprocess a set of objects in $\mathbb{R}^d$ into a data structure so that all the objects of $S$ intersected by a query segment can be reported (or counted) efficiently. We have already given an example of segment intersection-searching in the beginning of this section. See Table 8 for some of the known results on segment intersection searching. For the sake of clarity, we have omitted polylogarithmic factors from the query-search time whenever it is of the form $n/m^{\alpha}$.

If we are interested in just determining whether a query segment intersects any of the input objects, better bounds can be achieved in some cases. For example, a segment intersection-detection query for a set of balls in $\mathbb{R}^d$, where $d \leq 3$, can be answered in $O(\log n)$ time using $O(n^{d+\varepsilon})$ storage [4].

| $d$ | Objects | Problem | Size | Query Time | Source |
|---|---|---|---|---|---|
| | Simple polygons | Reporting | $n$ | $(k+1)\log n$ | [164] |
| | Lines | Reporting | $m$ | $n/\sqrt{m}+k$ | [10, 85] |
| $d=2$ | Segments | Counting | $m$ | $n/\sqrt{m}$ | [10, 85] |
| | Circles | Counting | $n^{2+\varepsilon}$ | $\log n$ | [15] |
| | Circular arcs | Counting | $m$ | $n/m^{1/3}$ | [15] |
| | Planes | Counting | $m$ | $n/m^{1/3}$ | [7] |
| | Halfplanes | Reporting | $m$ | $n/m^{1/3}+k$ | [7] |
| $d=3$ | Triangles | Counting | $m$ | $n/m^{1/4}$ | [8] |
| | Spheres | Counting | $m$ | $n/m^{1/4}$ | [8] |
| | Spheres | Reporting | $n^{3+\varepsilon}$ | $(k+1)\log^2 n$ | [4] |
| | Hyperplanes | Counting | $m$ | $n/m^{1/d}$ | [7] |

**Table 8.** Asymptotic upper bounds for segment intersection searching, with polylogarithmic factors omitted.

A special case of segment intersection searching, in which the objects are horizontal segments in the plane and query ranges are vertical segments, has been widely studied. In this case a query can be answered in time $O(\log n + k)$ using $O(n \log n)$ space and preprocessing [273]. If we also allow insertions and deletions, the query and update time are respectively $O(\log n \log\log n + k)$ and $O(\log n \log\log n)$ [212], or $O(\log^2 n + k)$ and $O(\log n)$ using only linear space [84]; if we allow only insertions, the query and update time become $O(\log n + k)$ and $O(\log n)$ [173].

A problem related to segment intersection searching is the *stabbing problem.* Given a set $S$ of objects in $\mathbb{R}^d$, determine whether a query $k$-flat $(0 < k < d)$ intersects all objects of $S$. Such queries can also be answered efficiently using semialgebraic range-searching data structures. A line-stabbing query amid a set of triangles in $\mathbb{R}^3$ can be answered in $O(\log n)$ time using $O(n^{2+\varepsilon})$ storage [240]. The paper by Goodman *et al.* [142] is an excellent survey of this topic.

## 6.3 Rectangle intersection searching

Given a set $S$ of polygons in the plane, preprocess them into a data structure so that all objects intersecting a query rectangle can be reported efficiently. This problem, also known as the *windowing query* problem, arises in a variety of applications. In many situations, the query output is required to be clipped within the query rectangle. In practice, each polygon in $S$ is approximated by its smallest enclosing rectangle and the resulting rectangles are preprocessed for rectangle-rectangle intersection searching, as discussed in Section 3.6. If the polygons in $S$ are large, then this scheme is not efficient, especially if we want to clip the query output within the query rectangle. A few data structures, for example, strip trees [32] and V-trees [209], have been proposed that store each polygon hierarchically. We can use these data structures to store each polygon and then construct an R-tree or any other orthogonal range-searching data structure on the smallest enclosing rectangles of the polygons. Nievergelt and Widmayer [225] describe another data structure, called a *guard file*, which is suitable if the polygons are fat (have bounded aspect ratio). They place a set of well-chosen points, called *guards*, and associate a subset of polygons with each guard that either contain the guard or lie "near" the guard. For a query rectangle $\gamma$, they determine the set of guards that lie inside $\gamma$; the lists of polygons associated with these guards give the candidates that intersect $\gamma$.

### 6.4 Colored intersection searching

Preprocess a given set $S$ of colored objects in $\mathbb{R}^d$ (*i.e.*, each object in $S$ is assigned a color) so that the we can report (or count) the colors of the objects that intersect the query range. This problem arises in many contexts where one wants to answer intersection-searching queries for input objects of non-constant size. For example, given a set $P = \{P_1, \ldots, P_m\}$ of $m$ simple polygons, one may wish to report all the simple polygons that intersect a query segment; the goal is to return the index, and not the description, of these polygons. If we color the edges of $P_i$ by the color $i$, the problem reduces to colored segment intersection searching in a set of segments.

If an intersection-detection query for $S$ with respect to a range $\gamma$ can be answered in $Q(n)$ time, then a colored intersection-reporting query with $\gamma$ can be answered in time $O((k \log(n/k) + 1)Q(n))$. Thus, logarithmic query-time intersection-searching data structures can easily be modified for colored intersection reporting, but very little is known about linear-size colored intersection-searching data structures, except in some special cases [14, 50, 153, 154, 155, 176].

Gupta *et al.* [153] have shown that the colored halfplane-reporting queries in the plane can be answered in $O(\log^2 n + k)$ using $O(n \log n)$ space. Agarwal and van Kreveld [14] presented a linear-size data structure with $O(n^{1/2+\varepsilon} + k)$ query time for colored segment intersection-reporting queries amid a set of segments in the plane, assuming that the segments of the same color form a connected planar graph, or if they form the boundary of a simple polygon; these data structures can also handle insertions of new segments. Gupta *et al.* [153, 155] present segment intersection-reporting structures for many other special cases.

## 7 Optimization Queries

The goal of an optimization query is to return an object that satisfies a certain condition with respect to a query range. Ray-shooting queries are perhaps the most common example of optimization queries. Other examples include segment-dragging and linear-programming queries.

### 7.1 Ray-shooting queries

Preprocess a set $S$ of objects in $\mathbb{R}^d$ into a data structure so that the first object (if any) hit by a query ray can be reported efficiently. This problem arises in ray tracing, hidden-surface removal, radiosity, and other graphics problems. Recently, efficient solutions to many other geometric problems have also been developed using ray-shooting data structures.

A general approach to the ray-shooting problem, using segment intersection-detection structures and Megiddo's parametric searching technique [210], was proposed by Agarwal and Matoušek [7]. Suppose we have a segment intersection-detection data structure for $S$. Let $\rho$ be a query ray. Their algorithm maintains a segment $\overline{ab} \subseteq \rho$ such that the first intersection point of $\overline{ab}$ with $S$ is the same as that of $\rho$. If $a$ lies on an object of $S$, it returns $a$. Otherwise, it picks a point $c \in \overline{ab}$ and determines, using the segment intersection-detection data structure, whether the interior of the segment $\overline{ac}$ intersects any object of $S$. If the answer is yes, it recursively finds the first intersection point of $\overline{ac}$ with $S$; otherwise, it recursively finds the first intersection point of $\overline{cb}$ with $S$. Using parametric searching, the points $c$ at each stage can be chosen in such a way that the algorithm terminates after $O(\log n)$ steps. In some cases, by using a more direct approach, we can improve the query time by a polylogarithmic factor. For example, by exploiting some additional properties of input objects and of partition trees, we can modify a segment intersection-searching data structure in some cases to answer ray shooting queries [3, 85, 149].

Another approach for answering ray-shooting queries is based on visibility maps. A ray in $\mathbb{R}^d$ can be represented as a point in $\mathbb{R}^d \times \mathbb{S}^{d-1}$. Given a set $S$ of objects, we can partition the parametric space $\mathbb{R}^d \times \mathbb{S}^{d-1}$ into cells so that all points within each cell correspond to rays that hit the same object first; this partition is called the *visibility map* of $S$. Using this approach and some other techniques, Chazelle and Guibas [77] showed that a ray-shooting query in a simple polygon can be answered in $O(\log n)$ time using $O(n)$ space. Simpler data structures were subsequently proposed by Chazelle *et al.* [69] and Hershberger and Suri [164]. Following a similar approach, Pocchiola and Vegter [241] showed that a ray-shooting query amid a set $\mathcal{P}$ of $s$ disjoint convex polygons, with a total of $n$ vertices, can be answered in $O(\log n)$ time, using $O(n+m)$ space, where $m = O(s^2)$ is the size of the visibility graph of $\mathcal{P}$.[8]

Table 9 gives a summary of known ray-shooting results. For the sake of clarity, we have omitted polylogarithmic factors from query times of the form $n/m^\alpha$. The ray-shooting structures for $d$-dimensional convex polyhedra by Matoušek and Schwarzkopf [205] assume that the source point of the query ray lies inside the polytope. All the ray-shooting data structures mentioned in Table 9 can be dynamized at a cost of polylogarithmic or $n^\varepsilon$ factor

[8] The vertices of the *visibility graph* are the vertices of the polygons. Besides the polygon edges, there is an edge in the graph between two vertices $v_i, v_j$ of convex polygons $P_i$ and $P_j$ if the line segment $\overline{v_i v_j}$ does not intersect any other convex polygon and the line supporting the segment is tangent to both $P_i$ and $P_j$.

| $d$ | Objects | Size | Query Time | Source |
|---|---|---|---|---|
| $d=2$ | Simple polygon | $n$ | $\log n$ | [164] |
| | $s$ disjoint simple polygons | $n$ | $\sqrt{s}$ | [11, 164] |
| | $s$ disjoint simple polygons | $(s^2+n)\log s$ | $\log s \log n$ | [11] |
| | $s$ disjoint convex polygons | $s^2+n$ | $\log n$ | [241] |
| | $s$ convex polygons | $sn\log s$ | $\log s\log n$ | [11] |
| | Segments | $m$ | $n/\sqrt{m}$ | [10, 85] |
| | Circular arcs | $m$ | $n/m^{1/3}$ | [15] |
| | Disjoint arcs | $n$ | $\sqrt{n}$ | [15] |
| $d=3$ | Convex polytope | $n$ | $\log n$ | [104] |
| | $c$-oriented polytopes | $n$ | $\log n$ | [100] |
| | $s$ convex polytopes | $s^2n^{2+\varepsilon}$ | $\log^2 n$ | [12] |
| | Halfplanes | $m$ | $n/m^{1/3}$ | [7] |
| | Terrain | $m$ | $n/\sqrt{m}$ | [7, 73] |
| | Triangles | $m$ | $n/m^{1/4}$ | [8] |
| | Spheres | $n^{3+\varepsilon}$ | $\log^2 n$ | [4] |
| $d>3$ | Hyperplanes | $m$ | $n/m^{1/d}$ | [7] |
| | Hyperplanes | $\dfrac{n^d}{\log^{d-\varepsilon} n}$ | $\log n$ | [75, 202] |
| | Convex polytope | $m$ | $n/m^{1/\lfloor d/2\rfloor}$ | [7, 205] |
| | Convex polytope | $\dfrac{n^{\lfloor d/2\rfloor}}{\log^{\lfloor d/2\rfloor-\varepsilon} n}$ | $\log n$ | [205] |

**Table 9.** Asymptotic upper bounds for ray shooting queries, with polylogarithmic factors omitted.

in the query time. Goodrich and Tamassia [143] have developed a dynamic ray-shooting data structure for connected planar subdivisions, with $O(\log^2 n)$ query and update time.

Like range searching, many practical data structures have been proposed that, despite having bad worst-case performance, work well in practice. The books by Foley *et al.* [128] and Glassner [140] describe several practical data structures for ray tracing that are used in computer graphics. One common approach is to construct a subdivision of $\mathbb{R}^d$ into constant-size cells so that the interior of each cell does not intersect any object of $S$. A ray-shooting query can be answered by traversing the query ray through the subdivision until we find an object that intersects the ray. The worst-case query time is proportional to the maximum number of cells intersected by a segment that does not intersect any object in $S$; we refer to this quantity as the *crossing number* of the triangulation. Hershberger and Suri [164] showed that if $S$ is the boundary of a simple polygon, then a triangulation (using Steiner points) with $O(\log n)$ crossing number can be constructed in $O(n\log n)$ time. See [5, 216, 105, 197, 283] and the references therein for other ray-shooting results using this approach. Agarwal *et al.* [5] proved worst-case bounds for many cases on the number of cells in the subdivision that a line can intersect. For example, they show that the crossing number for a set of $k$ disjoint convex polyhedra in $\mathbb{R}^3$ is $\Omega(k+\log n)$, and they present an algorithm that constructs a triangulation of size $O(nk\log n)$ with stabbing number $O(k\log n)$. Aronov and Fortune [22] prove a bound on the average crossing number of set of disjoint triangles in $\mathbb{R}^3$, and present a polynomial-time algorithm to construct a triangulation that achieves this bound. In practice, however, very simple decompositions, such as oct-trees and binary space partitions [139] are used to trace a ray.

### 7.2 Nearest-neighbor queries

The *nearest-neighbor query* problem is defined as follows: Preprocess a set $S$ of points in $\mathbb{R}^d$ into a data structure so that a point in $S$ closest to a query point $\xi$ can be reported quickly. This is one of the most widely studied problems in computational geometry because it arises in so many different areas, including pattern recognition [98, 107], data compression [25, 243], information retrieval [124, 249], CAD [213], molecular biology [261], image analysis [186, 188], data mining [123, 160], machine learning [97], and geographic information systems [246, 266]. Most applications use so-called *feature vectors* to map a complex object to a point in high dimensions. Examples of feature vectors include color histograms, shape descriptors, Fourier vectors, and text descriptors.

For simplicity, we assume that the distance between points is measured in the Euclidean metric, though a more complicated metric can be used depending on the application. For $d = 2$, one can construct the Voronoi diagram of $S$ and preprocess it for point-location queries in $O(n \log n)$ time [242]. For higher dimensions, Clarkson [89] presented a data structure of size $O(n^{\lceil d/2 \rceil + \varepsilon})$ that can answer a query in $2^{O(d)} \log n$ time. The query time can be improved to $O(d^3 \log n)$, using a technique of Meiser [214].

A nearest-neighbor query for a set of points under the Euclidean metric can be formulated as an instance of the ray-shooting problem in a convex polyhedron in $\mathbb{R}^{d+1}$, as follows. We map each point $p = (p_1, \ldots, p_d) \in S$ to a hyperplane $\hat{p}$ in $\mathbb{R}^{d+1}$, which is the graph of the function

$$f_p(x_1, \ldots, x_d) = 2p_1x_1 + \cdots + 2p_dx_d - (p_1^2 + \cdots + p_d^2).$$

Then $p$ is a closest neighbor of a point $\xi = (\xi_1, \ldots, \xi_d)$ if and only if

$$f_p(\xi_1, \ldots, \xi_d) = \max_{q \in S} f_q(\xi_1, \ldots, \xi_d).$$

That is, if and only if $f_p$ is the first hyperplane intersected by the vertical ray $\rho(\xi)$ emanating from the point $(\xi_1, \ldots, \xi_d, 0)$ in the negative $x_{d+1}$-direction. If we define $P = \bigcap_{p \in S} \{(x_1, \ldots, x_{d+1}) \mid x_{d+1} \geq f_p(x_1, \ldots, x_d)\}$, then $p$ is the nearest neighbor of $\xi$ if and only if the intersection point of $\rho(\xi)$ and $\partial P$ lies on the graph of $f_p$. Thus a nearest-neighbor query can be answered in time roughly $n/m^{1/\lceil d/2 \rceil}$ using $O(m)$ space. This approach can be extended to answer farthest-neighbor and $k$-nearest-neighbor queries also. In general, if we have an efficient data structure for answering disk-emptiness queries for disks under a given metric $\rho$, we can apply parametric searching to answer nearest-neighbor queries under the $\rho$-metric, provided the data structure satisfies certain mild assumptions [7].

Note that the query time of the above approach is exponential in $d$, so it is impractical even for moderate values of $d$ (say $d \approx 10$). This has lead to the development of algorithms for finding approximate nearest neighbors [26, 28, 29, 91, 185, 188] or for special cases, such as when the distribution of query points is known in advance [87, 296].

Because of wide applications of nearest-neighbor searching, many heuristics have been developed, especially in higher dimensions. These algorithms use practical data structures described in Section 3, including *kd*-trees, R-trees, R$^*$-trees, and Hilbert R-trees; see e.g. [138, 166, 188, 186, 123, 160, 246, 266]. White and Jain [284] described a variant of R-tree for answering nearest-neighbor queries in which they use spheres instead of rectangles as enclosing regions. This approach was further extended by Katayama and Satoh [181]. Berchtold *et al.* [45] present a parallel algorithm for nearest-neighbor searching. For large input sets, one desires an algorithm that minimizes the number of disk accesses. Many of the heuristics mentioned above try to optimize the I/O efficiency, though none of them gives any performance guarantee. A few recent papers [24, 46, 236, 93] analyze the efficiency of some of the heuristics, under certain assumptions on the input.

### 7.3 Linear programming queries

Let $S$ be a set of $n$ halfspaces in $\mathbb{R}^d$. We wish to preprocess $S$ into a data structure so that for a direction vector $\vec{v}$, we can determine the first point of $\bigcap_{h \in S} h$ in the direction $\vec{v}$. For $d \leq 3$, such a query can be answered in $O(\log n)$ time using $O(n)$ storage, by constructing the normal diagram of the convex polytope $\bigcap_{h \in S} h$ and preprocessing it for point-location queries. For higher dimensions, Matoušek [201] showed that, using multi-dimensional parametric searching and a data structure for answering halfspace emptiness queries, a linear-programming query can be answered in $O((n/m^{1/\lfloor d/2 \rfloor}) \operatorname{polylog} n)$ with $O(m)$ storage. Recently Chan [54] has described a randomized procedure whose expected query time is $n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$, using linear space.

### 7.4 Segment dragging queries

Preprocess a set $S$ of objects in the plane so that for a query segment $e$ and a ray $\rho$, the first position at which $e$ intersects any object of $S$ as it is translated (dragged) along $\rho$ can be determined quickly. This query can be answered in $O((n/\sqrt{m}) \operatorname{polylog} n)$ time, with $O(m)$ storage, using segment intersection-searching structures and parametric searching. Chazelle [57] gave a linear-size, $O(\log n)$ query-time data structure for the special case in which $S$ is a set of points, $e$ is a horizontal segment, and $\rho$ is the vertical direction. Instead of dragging a segment along a ray, one can ask the same question for dragging along a more complex trajectory (along a curve and allowing both translation and rotation). These problems arise quite often in motion planning and manufacturing. See [215, 253] for a few such examples.

## 8 Concluding Remarks

In this survey paper we reviewed both theoretical and practical data structures for range searching. Theoretically optimal or near-optimal data structures are known for most range searching problems. However, from a practical standpoint, range searching is still largely open; known data structures are rather complicated and do not perform well in practice, especially as the dimension increases. Lower bounds suggest that we cannot hope for data structures that do significantly better than the naïve algorithm in the worst case (and for some problems, even in the average case), but it is still an interesting open question to develop simple data structures that work well on typical inputs, especially in high dimensions.

As we saw in this survey, range-searching data structures are useful for other geometric-searching problems as well. In the quest for efficient range-searching data structures, researchers have discovered several elegant geometric techniques that have enriched computational geometry as a whole. It is impossible to describe in a survey paper all the known techniques and results on range searching and their applications to other geometric problems. We therefore chose a few of these techniques that we thought were most interesting. For further details, we refer the interested reader to the books by Mulmuley [223], Preparata and Shamos [242], and Samet [251], and the survey papers by Chazelle [63], Güting [156], Matoušek [200, 204], and Nievergelt and Widmayer [226].

## References

[1] D. J. Abel and D. Mark, A comparative analysis of some two-dimensional orderings, *Intl. J. Geographic Informations Systems*, 4 (1990), 21–31.

[2] P. K. Agarwal, Geometric partitioning and its applications, in: *Computational Geometry: Papers from the DIMACS special year* (J. E. Goodman, R. Pollack, and W. Steiger, eds.), American Mathematical Society, 1991.

[3] P. K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, *SIAM J. Comput.*, 21 (1992), 540–570.

[4] P. K. Agarwal, B. Aronov, and M. Sharir, Computing envelopes in four dimensions with applications, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 348–358.

[5] P. K. Agarwal, B. Aronov, and S. Suri, Line stabbing bounds in three dimensions, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 267–276.

[6] P. K. Agarwal, A. Efrat, and M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 39–50.

[7] P. K. Agarwal and J. Matoušek, Ray shooting and parametric search, *SIAM J. Comput.*, 22 (1993), 794–806.

[8] P. K. Agarwal and J. Matoušek, On range searching with semialgebraic sets, *Discrete Comput. Geom.*, 11 (1994), 393–418.

[9] P. K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, *Algorithmica*, 13 (1995), 325–345.

[10] P. K. Agarwal and M. Sharir, Applications of a new space partitioning technique, *Discrete Comput. Geom.*, 9 (1993), 11–38.

[11] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polygons in 2D, *J. Algorithms*, 21 (1996), 508–519.

[12] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions, *SIAM J. Comput.*, 25 (1996), 100–116.

[13] P. K. Agarwal, M. Sharir, and S. Toledo, Applications of parametric searching in geometric optimization, *J. Algorithms*, 17 (1994), 292–318.

[14] P. K. Agarwal and M. van Kreveld, Polygon and connected component intersection searching, *Algorithmica*, 15 (1996), 626–660.

[15] P. K. Agarwal, M. van Kreveld, and M. Overmars, Intersection queries in curved objects, *J. Algorithms*, 15 (1993), 229–266.

[16] A. Aggarwal, M. Hansen, and T. Leighton, Solving query-retrieval problems by compacting Voronoi diagrams, *Proc. 22nd Annu. ACM Sympos. Theory Comput.*, 1990, pp. 331–340.

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

[18] A. Andersson and K. Swanson, On the difficulty of range searching, *Proc. 4th Workshop Algorithms Data Struct.*, *Lecture Notes Comput. Sci.*, Vol. 955, Springer-Verlag, 1995, pp. 473–481.

[19] M. Anthony and N. Biggs, *Computational Learning Theory*, Cambridge University Press, Cambridge, 1992.

[20] L. Arge, The Buffer Tree: A new technique for optimal I/O-algorithms, *Proc. 4th Workshop Algorithms Data Struct.*, number 955 in Lecture Notes Comput. Sci., 1995, pp. 334–345.

[21] L. Arge and J. S. Vitter, Optimal interval management in external memory, *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, October 1996, pp. 560–569.

[22] B. Aronov and S. Fortune, Average-case ray shooting and minimum weight triangulation, *Proc. 23th Annu. Sympos. Comput. Geom.*, 1997, pp. 203–211.

[23] B. Aronov, M. Pellegrini, and M. Sharir, On the zone of a surface in a hyperplane arrangement, *Discrete Comput. Geom.*, 9 (1993), 177–186.

[24] S. Arya, D. Mount, and O. Narayan, Accounting for boundary effects in nearest neighbor searching, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 336–344.

[25] S. Arya and D. M. Mount, Algorithms for fast vector quantization, *Data Compression Conference*, IEEE Press, 1993, pp. 381–390.

[26] S. Arya and D. M. Mount, Approximate nearest neighbor queries in fixed dimensions, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 271–280.

[27] S. Arya and D. M. Mount, Approximate range searching, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 172–181.

[28] S. Arya, D. M. Mount, and O. Narayan, Accounting for boundary effects in nearest-neighbor searching, *Discrete Comput. Geom.*, 16 (1996), 155–176.

[29] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Wu, An optimal algorithm for approximate nearest neighbor searching, *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 1994, pp. 573–582.

[30] T. Asano, T. Roos, P. Widmayer, and E. Welzl, Space filling curves and their use in the design of geometric data structures, *Proc. 2nd Latin Amer. Sympos. Theoret. Informatics*, *Lecture Notes Comput. Sci.*, Vol. 911, Springer-Verlag, 1995, pp. 36–48.

[31] D. Avis, Non-partitionable point sets, *Inform. Process. Lett.*, 19 (1984), 125–129.

[32] D. H. Ballard, Strip trees: A hierarchical representation for curves, *Commun. ACM*, 24 (1981), 310–321.

[33] R. Bar-Yehuda and S. Fogel, Partitioning a sequence into few monotone subsequences, Technical Report 640, Technion IIT, Haifa, Israel, 1990.

[34] R. Bar-Yehuda and S. Fogel, Variations on ray shooting, *Algorithmica*, 11 (1994), 133–145.

[35] R. Bayer and McCreight, Organization of large ordered indexes, *Acta Inform.*, 1 (1972), 173–189.

[36] B. Becker, H. Six, and P. Widmayer, Spatial priority search: An access technique for scaleless maps, *Proc. ACM SIGMOD Conf. on Management of Data*, 1991, pp. 128–138.

[37] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, The R*-tree: An efficient and robust access method for points and rectangles, *Proc. ACM SIGMOD Conf. on Management of Data*, 1990, pp. 322–331.

[38] A. M. Ben-Amram, Lower bounds on algebraic random access machines, *Proc. 22nd Internat. Colloq. Automata Lang. Prog.*, *Lecture Notes Comput. Sci.*, Vol. 944, Springer-Verlag, 1995, pp. 360–371.

[39] A. M. Ben-Amram, What is a "pointer machine"?, *SIGACT News*, 26 (1995), 88–95.

[40] M. Ben-Or, Lower bounds for algebraic computation trees, *Proc. 15th Annu. ACM Sympos. Theory Comput.*, 1983, pp. 80–86.

[41] J. L. Bentley, Multidimensional binary search trees used for associative searching, *Commun. ACM*, 18 (1975), 509–517.

[42] J. L. Bentley, Multidimensional divide-and-conquer, *Commun. ACM*, 23 (1980), 214–229.

[43] J. L. Bentley and J. H. Friedman, Data structures for range searching, *ACM Comput. Surv.*, 11 (1979), 397–409.

[44] J. L. Bentley and J. B. Saxe, Decomposable searching problems I: Static-to-dynamic transformation, *J. Algorithms*, 1 (1980), 301–358.

[45] S. Berchtold, C. Böhm, B. Barunmüller, D. A. Keim, and H.-P. Kriegel, Fast parallel similarity search in multimedia databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1997, pp. 1–12.

[46] S. Berchtold, C. Böhm, D. A. Keim, and H.-P. Kriegel, A cost model for nearest neighbor search in high-dimensional data space, *Proc. ACM Sympos. Principles of Database Systems*, 1997, pp. 78–86.

[47] S. Berchtold, D. A. Keim, and H.-P. Kriegel, The X-tree: An index structure for higher dimensional data, *Proc. 22th VLDB Conference*, 1996, pp. 28–39.

[48] T. Bially, Space-filling curves: Ttheir generation and their application to bandwidth reduction, *IEEE Trans. Information Theory*, 15 (1969), 658–664.

[49] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth, Classifying learnable geometric concepts with the Vapnik-Chervonenkis dimension, *J. ACM*, 36 (1989), 929–965.

[50] P. Bozanis, N. Ktsios, C. Makris, and A. Tsakalidis, New upper bounds for generalized intersection searching problems, *Proc. 22nd Inter. Colloq. Auto. Lang. Program.*, *Lecture Notes in Computer Science*, Vol. 944, 1995, pp. 464–475.

[51] P. Bozanis, N. Ktsios, C. Makris, and A. Tsakalidis, New results on intersection query problems, unpublished manuscript, 1996.

[52] H. Brönnimann, B. Chazelle, and J. Pach, How hard is halfspace range searching, *Discrete Comput. Geom.*, 10 (1993), 143–155.

[53] P. Bürgisser, M. Clausen, and M. A. Shokrollahi, *Algebraic Complexity Theory*, Springer-Verlag, 1996.

[54] T. M. Chan, Fixed-dimensional linear programming queries made easy, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 284–290.

[55] B. Chazelle, Filtering search: A new approach to query-answering, *SIAM J. Comput.*, 15 (1986), 703–724.

[56] B. Chazelle, Computing on a free tree via complexity-preserving mappings, *Algorithmica*, 2 (1987), 337–361.

[57] B. Chazelle, An algorithm for segment-dragging and its implementation, *Algorithmica*, 3 (1988), 205–221.

[58] B. Chazelle, A functional approach to data structures and its use in multidimensional searching, *SIAM J. Comput.*, 17 (1988), 427–462.

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

[60] B. Chazelle, Lower bounds for orthogonal range searching, I: The reporting case, *J. ACM*, 37 (1990), 200–212.

[61] B. Chazelle, Lower bounds for orthogonal range searching, II: The arithmetic model, *J. ACM*, 37 (1990), 439–463.

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

[63] B. Chazelle, Computational geometry: A retrospective, *Proc. 26th Annu. ACM Sympos. Theory Comput.*, 1994, pp. 75–94.

[64] B. Chazelle, A spectral approach to lower bounds, *Proc. 35th Annu. IEEE Sympos. Found. Comput. Sci.*, 1994, pp. 674–682.

[65] B. Chazelle, Lower bounds for off-line range searching, *Proc. 27th Annu. ACM Sympos. Theory Comput.*, 1995, pp. 733–740.

[66] B. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, New upper bounds for neighbor searching, *Inform. Control*, 68 (1986), 105–124.

[67] B. Chazelle and H. Edelsbrunner, Optimal solutions for a class of point retrieval problems, *J. Symbolic Comput.*, 1 (1985), 47–56.

[68] B. Chazelle and H. Edelsbrunner, Linear space data structures for two types of range search, *Discrete Comput. Geom.*, 2 (1987), 113–126.

[69] B. Chazelle, H. Edelsbrunner, M. Grigni, L. Guibas, J. Hershberger, M. Sharir, and J. Snoeyink, Ray shooting in polygons using geodesic triangulations, *Algorithmica*, 12 (1994), 54–68.

[70] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Proc. 16th Internat. Colloq. Automata Lang. Program.*, *Lecture Notes Comput. Sci.*, Vol. 372, Springer-Verlag, 1989, pp. 179–192.

[71] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Theoret. Comput. Sci.*, 84 (1991), 77–105.

[72] B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir, Diameter, width, closest line pair and parametric searching, *Discrete Comput. Geom.*, 10 (1993), 183–196.

[73] B. Chazelle, H. Edelsbrunner, L. J. Guibas, M. Sharir, and J. Stolfi, Lines in space: Combinatorics and algorithms, *Algorithmica*, 15 (1996), 428–447.

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

[75] B. Chazelle and J. Friedman, Point location among hyperplanes and unidirectional ray-shooting, *Comput. Geom. Theory Appl.*, 4 (1994), 53–62.

[76] B. Chazelle and L. J. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica*, 1 (1986), 133–162.

[77] B. Chazelle and L. J. Guibas, Visibility and intersection problems in plane geometry, *Discrete Comput. Geom.*, 4 (1989), 551–581.

[78] B. Chazelle, L. J. Guibas, and D. T. Lee, The power of geometric duality, *BIT*, 25 (1985), 76–90.

[79] B. Chazelle and F. P. Preparata, Halfspace range search: An algorithmic application of $k$-sets, *Discrete Comput. Geom.*, 1 (1986), 83–93.

[80] B. Chazelle and B. Rosenberg, Computing partial sums in multidimensional arrays, *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, 1989, pp. 131–139.

[81] B. Chazelle and B. Rosenberg, Simplex range reporting on a pointer machine, *Comput. Geom. Theory Appl.*, 5 (1996), 237–247.

[82] B. Chazelle, M. Sharir, and E. Welzl, Quasi-optimal upper bounds for simplex range searching and new zone theorems, *Algorithmica*, 8 (1992), 407–429.

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

[84] S. W. Cheng and R. Janardan, Efficient dynamic algorithms for some geometric intersection problems, *Inform. Process. Lett.*, 36 (1990), 251–258.

[85] S. W. Cheng and R. Janardan, Algorithms for ray-shooting and intersection searching, *J. Algorithms*, 13 (1992), 670–692.

[86] Y.-J. Chiang and R. Tamassia, Dynamic algorithms in computational geometry, *Proc. IEEE*, 80 (1992), 1412–1434.

[87] K. Clarkson, Nearest neighbor queries in metric spaces, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 609–617.

[88] K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete Comput. Geom.*, 2 (1987), 195–222.

[89] K. L. Clarkson, A randomized algorithm for closest-point queries, *SIAM J. Comput.*, 17 (1988), 830–847.

[90] K. L. Clarkson, Randomized geometric algorithms, in: *Computing in Euclidean Geometry* (D.-Z. Du and F. K. Hwang, eds.), World Scientific, Singapore, 1992, pp. 117–162.

[91] K. L. Clarkson, An algorithm for approximate closest-point queries, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 160–164.

[92] K. L. Clarkson and P. W. Shor, Applications of random sampling in computational geometry, II, *Discrete Comput. Geom.*, 4 (1989), 387–421.

[93] J. G. Cleary, Analysis of an algorithm for finding nearest neighbors in Euclidean space, *ACM Trans. Math. Softw.*, 5 (1979), 183–192.

[94] R. Cole, Partitioning point sets in 4 dimensions, *Proc. 12th Internat. Colloq. Automata Lang. Program.*, *Lecture Notes Comput. Sci.*, Vol. 194, Springer-Verlag, 1985, pp. 111–119.

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

[96] D. Comer, The ubiquitous B-tree, *ACM Comput. Surv.*, 11 (1979), 121–137.

[97] S. Cost and S. Salzberg, A weighted nearest neighbor algorithm for learning with symbolic features, *Machine Learning*, 10 (1993), 57–67.

[98] T. Cover and P. Hart, Nearest neighbor pattern classification, *IEEE Trans. Information Theory*, 13 (1967), 21–27.

[99] M. de Berg, L. J. Guibas, and D. Halperin, Vertical decompositions for triangles in 3-space, *Discrete Comput. Geom.*, 15 (1996), 35–61.

[100] M. de Berg and M. Overmars, Hidden surface removal for $c$-oriented polyhedra, *Comput. Geom. Theory Appl.*, 1 (1992), 247–268.

[101] D. P. Dobkin and H. Edelsbrunner, Organizing point sets in two and three dimensions, Tech. Report F130, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1984.

[102] D. P. Dobkin and H. Edelsbrunner, Space searching for intersecting objects, *J. Algorithms*, 8 (1987), 348–361.

[103] D. P. Dobkin, J. Hershberger, D. Kirkpatrick, and S. Suri, Implicitly searching convolutions and computing depth of collision, *Proc. 1st Annu. SIGAL Internat. Sympos. Algorithms*, *Lecture Notes Comput. Sci.*, Vol. 450, Springer-Verlag, 1990, pp. 165–180.

[104] D. P. Dobkin and D. G. Kirkpatrick, A linear algorithm for determining the separation of convex polyhedra, *J. Algorithms*, 6 (1985), 381–392.

[105] D. P. Dobkin and D. G. Kirkpatrick, Determining the separation of preprocessed polyhedra – A unified approach, *Proc. 17th Internat. Colloq. Automata Lang. Program.*, *Lecture Notes Comput. Sci.*, Vol. 443, Springer-Verlag, 1990, pp. 400–413.

[106] D. P. Dobkin and S. Suri, Maintenance of geometric extrema, *J. ACM*, 38 (1991), 275–298.

[107] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley Interscience, New York, 1973.

[108] H. Edelsbrunner, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Heidelberg, 1987.

[109] H. Edelsbrunner, L. Guibas, J. Hershberger, R. Seidel, M. Sharir, J. Snoeyink, and E. Welzl, Implicitly representing arrangements of lines or segments, *Discrete Comput. Geom.*, 4 (1989), 433–466.

[110] H. Edelsbrunner and F. Huber, Dissecting sets of points in two and three dimensions, Report F138, Inst. Informationsverarb., Tech. Univ. Graz, Graz, Austria, 1984.

[111] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer, Polygonal intersection searching, *Inform. Process. Lett.*, 14 (1982), 74–79.

[112] H. Edelsbrunner and H. A. Maurer, A space-optimal solution of general region location, *Theoret. Comput. Sci.*, 16 (1981), 329–336.

[113] H. Edelsbrunner and M. H. Overmars, Batched dynamic solutions to decomposable searching problems, *J. Algorithms*, 6 (1985), 515–542.

[114] H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inform. Process. Lett.*, 23 (1986), 289–293.

[115] A. Efrat, M. Katz, F. Nielsen, and M. Sharir, Dynamic data structures for fat objects and their applications, *Proc. 5th Workshop Algorithms Data Struct.*, 1997. To appear.

[116] P. Erdős and G. Szekeres, A combinatorial problem in geometry, *Compositio Math.*, 2 (1935), 463–470.

[117] J. Erickson, New lower bounds for halfspace emptiness, *37th Annu. ACM Sympos. Found. Comput. Sci.*, 1996, pp. 472–481.

[118] J. Erickson, New lower bounds for Hopcroft's problem, *Discrete Comput. Geom.*, 16 (1996), 389–418.

[119] J. Erickson, Space-time tradeoffs for emptiness queries, *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, 1997. To appear.

[120] G. Evangelidis, D. Lomet, and B. Salzberg, The $\mathrm{hB}^{\Pi}$-tree: A multi-attribute index supporting concurrency, recovery and node consolidation, *VLDB Journal*, 6 (1997), 1–25.

[121] C. Faloutsos, Gray codes for partial match and range queries, *IEEE Trans. on Software Eng.*, 44 (1988), 1381–1393.

[122] C. Faloutsos and V. Gaede, Analysis of $n$-dimensional quadtrees using the Hausdorff fractal dimension, *Proc. 22nd VLDB Conference*, 1996, pp. 40–50.

[123] C. Faloutsos and K.-I. Lin, *FastMap*: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 163–173.

[124] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos, Fast subsequence matching in time-series databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1994, pp. 86–93.

[125] C. Faloutsos and Y. Rong, DOT: A spatial access method using fractals, *Proc. 7th IEEE Internat. Conf. on Data Engineering*, 1991, pp. 152–158.

[126] C. Faloutsos and S. Roseman, Fractals for secondary key retrieval, *Proc. ACM SIGMOD Conf. on Management of Data*, 1989, pp. 247–252.

[127] R. A. Finkel and J. L. Bentley, Quad trees: a data structure for retrieval on composite keys, *Acta Inform.*, 4 (1974), 1–9.

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

[129] F. W. Fredman and D. E. Willard, Surpassing the information theoretic bound with fusion trees, *J. Comput. Syst. Sci.*, 48 (1993), 424–436.

[130] M. L. Fredman, The complexity of maintaining an array and computing its partial sums, *J. ACM*, 29 (1979), 250–260.

[131] M. L. Fredman, Inherent complexity of range query problems, *Proc. 17th Allerton Conf. Commun. Control Comput.*, 1979, pp. 231–240.

[132] M. L. Fredman, The inherent complexity of dynamic data structures which accommodate range queries, *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, 1980, pp. 191–199.

[133] M. L. Fredman, A lower bound on the complexity of orthogonal range queries, *J. ACM*, 28 (1981), 696–705.

[134] M. L. Fredman, Lower bounds on the complexity of some optimal data structures, *SIAM J. Comput.*, 10 (1981), 1–10.

[135] M. L. Fredman and D. J. Volper, The complexity of partial match retrieval in a dynamic setting, *J. Algorithms*, 3 (1982), 68–78.

[136] H. Freeston, The BANG file: a new kind of grid file, *Proc. ACM SIGMOD Conf. on Management of Data*, 1987, pp. 260–269.

[137] M. Freestone, A general solution of the $n$-dimensional B-tree problem, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 80–91.

[138] J. H. Friedman, J. L. Bentley, and R. A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Trans. Math. Softw.*, 3 (1977), 209–226.

[139] H. Fuchs, Z. M. Kedem, and B. Naylor, On visible surface generation by a priori tree structures, *Comput. Graph.*, 14 (1980), 124–133. Proc. SIGGRAPH '80.

[140] A. S. Glassner, *Ray Tracing*, Academic Press, 1989.

[141] J. Goldstein, R. Ramakrishnan, U. Shaft, and J.-B. Yu, Processing queries by linear constraints, *Proc. ACM Sympos. Principles of Database Systems*, 1997, pp. 257–267.

[142] J. E. Goodman, R. Pollack, and R. Wenger, Geometric transversal theory, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, Heidelberg–New York–Berlin, 1993, pp. 163–198.

[143] M. T. Goodrich and R. Tamassia, Dynamic ray shooting and shortest paths via balanced geodesic triangulations, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 318–327.

[144] J. Gray, A. Bosworth, A. Layman, and H. Patel, Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals, *Proc. 12th IEEE Internat. Conf. on Data Engineering*, 1996, pp. 152–159.

[145] D. Greene, An implementation and performance analysis of spatial data access methods, *Proc. 5th IEEE Internat. Conf. on Data Engineering*, 1989, pp. 606–615.

[146] L. Greengard, *The Rapid Evaluation of Potential Fields in Particle Systems*, MIT Press, Cambridge, 1988.

[147] D. Grigoriev and N. Vorobjov, Complexity lower bounds for computation trees with elementary transcendental function gates, *Theoret. Comput. Sci.*, 157 (1996), 185–214.

[148] R. Grossi and G. F. Italiano, Efficient splitting and merging algorithms for order decomposable problems, *Proc. 24th Internat. Colloq. Automata, Lang. Prog.*, 1997, to appear.

[149] L. Guibas, M. Overmars, and M. Sharir, Ray shooting, implicit point location, and related queries in arrangements of segments, Report 433, Dept. Comput. Sci., New York Univ., New York, NY, March 1989.

[150] O. Günther, The design of the cell tree: An object oriented index structure for geometric data bases, *Proc. 5th IEEE Internat. Conf. on Data Engineering*, 1989, pp. 598–605.

[151] O. Günther and J. Bilmes, Tree based access methods for spatial databases: Implementation and performance evaluation, *IEEE Trans. Knowledge and Data Engineering*, 3 (1991), 342–356.

[152] H. Gupta, V. Harinarayan, A. Rajaraman, and J. D. Ullman, Index selection for OLAP, *Proc. ACM SIGMOD Conf. on Management of Data*, 1996, pp. 205–216.

[153] P. Gupta, R. Janardan, and M. Smid, Efficient algorithms for generalized intersection searching on non-iso-oriented objects, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 369–378.

[154] P. Gupta, R. Janardan, and M. Smid, On intersection searching problems involving curved objects, *Proc. 4th Scand. Workshop Algorithm Theory*, *Lecture Notes Comput. Sci.*, Vol. 824, Springer-Verlag, 1994, pp. 183–194.

[155] P. Gupta, R. Janardan, and M. Smid, Further results on generalized intersection searching problems: counting, reporting and dynamization, *J. Algorithms*, 19 (1995), 282–317.

[156] R. Güting, An introduction to spatial database systems, *VLDB Journal*, 4 (1994), 357–399.

[157] A. Guttman, R-trees: a dynamic index structure for spatial searching, *Proc. ACM SIGACT-SIGMOD Conf. Principles Database Systems*, 1984, pp. 569–592.

[158] D. Harel and R. E. Tarjan, Fast algorithms for finding nearest common ancestors, *SIAM J. Comput.*, 13 (1984), 338–355.

[159] V. Harinarayan, A. Rajaraman, and J. D. Ullman, Implementing data cubes efficiently, *Proc. 13th IEEE Internat. Conf. on Data Engineering*, 1997, pp. 208–219.

[160] T. Hastie and R. Tibshirani, Discriminant adaptive nearest neighbor classification, *IEEE Trans. Pattern Anal. Mach. Intell.*, 18 (1996), 607–616.

[161] D. Haussler and E. Welzl, Epsilon-nets and simplex range queries, *Discrete Comput. Geom.*, 2 (1987), 127–151.

[162] J. H. Hellerstein, E. Koustsoupias, and C. H. Papadimtriou, On the analysis of indexing schemes, *Proc. ACM Sympos. Principles of Database Systems*, 1997, pp. 249–256.

[163] A. Henrich, Improving the performance of multi-dimensional access structures based on *kd*-trees, *Proc. 12th IEEE Intl. Conf. on Data Engineering*, 1996, pp. 68–74.

[164] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *J. Algorithms*, 18 (1995), 403–431.

[165] K. H. Hinrichs, The grid file system: implementation and case studies of applications, Report Diss. ETH 7734, Swiss Federal Inst. Tech. Zürich, Zürich, Switzerland, 1985.

[166] G. R. Hjaltason and H. Samet, Ranking in spatial databases, *Advances in Spatial Databases - Fourth International Symposium* (M. J. Egenhofer and J. R. Herring, eds.), number 951 in Lecture Notes Comput. Sci., August 1995, pp. 83–95.

[167] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant, Range queries in OLAP data cubes, *Proc. ACM SIGMOD Conf. on Management of Data*, 1997, pp. 73–88.

[168] C.-T. Ho, J. Bruck, and R. Agrawal, Partial-sum queries in OLAP data cubes using covering codes, *Proc. ACM Sympos. Principles of Database Systems*, 1997, pp. 228–237.

[169] E. G. Hoel and H. Samet, A qualitative comparison study of data structures for large line segment databases, *Proc. ACM SIGMOD Conf. on Management of Data*, 1992, pp. 205–214.

[170] P. Houthuys, Box sort, a multidimensional binary sorting method for rectangular boxes, used for quick range searching, *Visual Comput.*, 3 (1987), 236–249.

[171] A. Hutflesz, H.-W. Six, and P. Widmayer, Globally order preserving multidimensional linear hashing, *Proc. 4th Intl. Conf. on Data Engineering*, 1988, pp. 572–579.

[172] A. Hutflesz, H.-W. Six, and P. Widmayer, Twin grid files: Space optimizing access schemes, *Proc. ACM SIGMOD Conf. on Management of Data*, 1988, pp. 183–190.

[173] H. Imai and T. Asano, Dynamic orthogonal segment intersection search, *J. Algorithms*, 8 (1987), 1–18.

[174] P. Indyk, R. Motwani, P. Raghavan, and S. Vempala, Locality-preserving hashing in multidimensional space, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 618–625.

[175] H. V. Jagdish, Linear clustering of objects with multiple attributes, *Proc. ACM SIGMOD Conf. on Management of Data*, 1990, pp. 332–342.

[176] R. Janardan and M. Lopez, Generalized intersection searching problems, *Internat. J. Comput. Geom. Appl.*, 3 (1993), 39–69.

[177] I. Kamel and C. Faloutsos, Parallel R-trees, *Proc. ACM SIGMOD Conf. on Management of Data*, 1992, pp. 195–204.

[178] I. Kamel and C. Faloutsos, On packing R-trees, *Proc. 2nd Internat. Conf. on Information and Knowledge Management*, 1993, pp. 490–499.

[179] I. Kamel and C. Faloutsos, Hilbert R-tree: An improved R-tree using fractals, *Proc. 20th VLDB Conference*, 1994, pp. 500–510.

[180] P. C. Kanellakis, S. Ramaswamy, D. E. Vengroff, and J. S. Vitter, Indexing for data models with constraints and classes, *Proc. 12th ACM SIGACT-SIGMOD-SIGART Conf. Princ. Database Sys.*, 1993, pp. 233–243.

[181] N. Katayama and S. Satoh, The SR-tree: An index structure for high-dimensional nearest-neighbor queries, *Proc. ACM SIGMOD Conf. on Management of Data*, 1997, pp. 369–380.

[182] M. Katz, 3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting amidst convex fat objects, Research Report 2583, INRIA, BP93, 06902 Sophia-Antipolis, France, 1995.

[183] M. D. Katz and D. J. Volper, Data structures for retrieval on square grids, *SIAM J. Comput.*, 15 (1986), 919–931.

[184] R. Klein, O. Nurmi, T. Ottmann, and D. Wood, A dynamic fixed windowing problem, *Algorithmica*, 4 (1989), 535–550.

[185] J. Kleinberg, Two algorithms for nearest-neighbor search in high dimension, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 599–608.

[186] V. Koivune and S. Kassam, Nearest neighbor filters for multivariate data, *IEEE Workshop on Nonlinear Signal and Image Processing*, 1995.

[187] J. Komlós, J. Pach, and G. Woeginger, Almost tight bounds for $\epsilon$-nets, *Discrete Comput. Geom.*, 7 (1992), 163–173.

[188] F. Korn, N. Sidiropoulos, C. Faloutsos, E. Siegel, and Z. Protopapa, Fast nearest neighbor search in medical image database, *Proc. 22nd VLDB Conference*, 1996, pp. 215–226.

[189] H.-P. Kriegel and B. Seeger, Multidimensional order preserving linear hashing with partial expansions, in: *Proc. Intl. Conf. on Database Theory*, *Lecture Notes Comput. Sci.*, Vol. 243, Springer-Verlag, 1986, pp. 203–220.

[190] R. Krishnamurthy and K.-Y. Wang, Multilevel grid files, Tech. report, IBM T. J. Watson Center, Yorktown Heights, NY, 1985.

[191] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[192] D. T. Lee and C. K. Wong, Finding intersections of rectangles by range search, *J. Algorithms*, 2 (1981), 337–347.

[193] S. Leutenegger, M. A. Lopez, and J. Edington, STR: A simple and efficient algorithm for R-tree packing, *Proc. 13th IEEE Internat. Conf. on Data Engineering*, 1997, pp. 497–506.

[194] K. Lin, H. Jagdish, and C. Faloutsos, The TV-tree: An index structure for higher dimensional data, *VLDB Journal*, 4 (1994), 517–542.

[195] D. Lomet and B. Salzberg, The hB-tree: A multiattribute indexing method with good guaranteed performance, *ACM Trans. Database systems*, 15 (1990), 625–658.

[196] G. S. Lueker, A data structure for orthogonal range queries, *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, 1978, pp. 28–34.

[197] J. MacDanold and K. Booth, Heuristics for ray tracing using space subdivision, *Visual Comput.*, 6 (1990), 153–166.

[198] J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.*, 8 (1992), 315–334.

[199] J. Matoušek, Reporting points in halfspaces, *Comput. Geom. Theory Appl.*, 2 (1992), 169–186.

[200] J. Matoušek, Epsilon-nets and computational geometry, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), *Algorithms and Combinatorics*, Vol. 10, Springer-Verlag, 1993, pp. 69–89.

[201] J. Matoušek, Linear optimization queries, *J. Algorithms*, 14 (1993), 432–448.

[202] J. Matoušek, On vertical ray shooting in arrangements, *Comput. Geom. Theory Appl.*, 2 (1993), 279–285.

[203] J. Matoušek, Range searching with efficient hierarchical cuttings, *Discrete Comput. Geom.*, 10 (1993), 157–182.

[204] J. Matoušek, Geometric range searching, *ACM Comput. Surv.*, 26 (1994), 421–461.

[205] J. Matoušek and O. Schwarzkopf, On ray shooting in convex polytopes, *Discrete Comput. Geom.*, 10 (1993), 215–232.

[206] J. Matoušek and E. Welzl, Good splitters for counting points in triangles, *J. Algorithms*, 13 (1992), 307–319.

[207] J. Matoušek, E. Welzl, and L. Wernisch, Discrepancy and $\varepsilon$-approximations for bounded VC-dimension, *Combinatorica*, 13 (1993), 455–466.

[208] E. M. McCreight, Priority search trees, *SIAM J. Comput.*, 14 (1985), 257–276.

[209] M. R. Mediano, M. A. Casanova, and M. Dreux, V-trees: A storage method for long vector data, *Proc. 20th VLDB Conference*, 1994, pp. 321–329.

[210] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM*, 30 (1983), 852–865.

[211] K. Mehlhorn, *Multi-dimensional Searching and Computational Geometry*, Springer-Verlag, Heidelberg, West Germany, 1984.

[212] K. Mehlhorn and S. Näher, Dynamic fractional cascading, *Algorithmica*, 5 (1990), 215–241.

[213] H. Mehrotra and J. E. Gary, Feature-based retrieval of similar shapes, *Proc. 9th IEEE Intl. Conf. on Data Engineering*, 1996, pp. 108–115.

[214] S. Meiser, Point location in arrangements of hyperplanes, *Inform. Comput.*, 106 (1993), 286–303.

[215] J. S. B. Mitchell, Shortest paths among obstacles in the plane, *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, 1993, pp. 308–317.

[216] J. S. B. Mitchell, D. M. Mount, and S. Suri, Query-sensitive ray shooting, *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, 1994, pp. 359–368.

[217] G. Morton, A computer oriented geodetic data base and a new technique in file sequencing, Tech. Rep., IBM Ltd., Ottawa, Canada, 1966.

[218] R. Motwani and P. Raghavan, *Randomized Algorithms*, Cambridge University Press, New York, NY, 1995.

[219] J. Mullin, Spiral storage: Efficient dynamic hashing with constant-performance, *The Computer Journal*, 28 (1985), 330–334.

[220] K. Mulmuley, A fast planar partition algorithm, I, *J. Symbolic Comput.*, 10 (1990), 253–280.

[221] K. Mulmuley, Randomized multidimensional search trees: Dynamic sampling, *Proc. 7th Annu. ACM Sympos. Comput. Geom.*, 1991, pp. 121–131.

[222] K. Mulmuley, Randomized multidimensional search trees: Further results in dynamic sampling, *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, 1991, pp. 216–227.

[223] K. Mulmuley, *Computational Geometry: An Introduction Through Randomized Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1994.

[224] J. Nievergelt, H. Hinterberger, and K. C. Sevcik, The grid file: An adaptable, symmetric multi-key file structure, *ACM Trans. Database Systems*, 9 (1984), 38–71.

[225] J. Nievergelt and P. Widmayer, Guard files: Stabbing and intersection queries on fat spatial objects, *The Computer Journal*, 36 (1993), 107–116.

[226] J. Nievergelt and P. Widmayer, Spatial data structures: Concepts and design choices, unpublished manuscript, 1996.

[227] J. Orenstein, A comparison of spatial query processing techniques for native and parameter spaces, *Proc. ACM SIGMOD Conf. on Management of Data*, 1990, pp. 343–352.

[228] J. Orenstein and T. Merrett, A class of data structures for associative searching, *Proc. ACM SIGMOD Conf. on Management of Data*, 1984, pp. 181–190.

[229] M. Ouksel, The interpolation-based grid file, *Proc. ACM Sympos. Principles of Database Systems*, 1985, pp. 20–27.

[230] M. H. Overmars, *The Design of Dynamic Data Structures*, Springer-Verlag, Heidelberg, West Germany, 1983.

[231] M. H. Overmars, Efficient data structures for range searching on a grid, *J. Algorithms*, 9 (1988), 254–275.

[232] M. H. Overmars, M. H. M. Smid, M. T. de Berg, and M. J. van Kreveld, Maintaining range trees in secondary memory, part I: Partitions, *Acta Inform.*, 27 (1990), 423–452.

[233] M. H. Overmars and A. F. van der Stappen, Range searching and point location among fat objects, *Algorithms – ESA'94* (J. van Leeuwen, ed.), *Lecture Notes Comput. Sci.*, Vol. 855, Springer-Verlag, 1994, pp. 240–253.

[234] J. Pach, Notes on geometric graph theory, in: *Discrete and Computational Geometry: Papers from the DIMACS Special Year* (J. E. Goodman, R. Pollack, and W. Steiger, eds.), American Mathematical Society, 1991, pp. 273–285.

[235] B.-U. Pagel, H.-W. Six, and H. Toben, The transformation technique for spatial objects revisited, in: *Proc. 3rd Intl. Symp. on Large Spatial Databases*, *Lecture Notes Comput. Sci.*, Vol. 692, Springer-Verlag, 1993, pp. 73–88.

[236] A. Papadpoulos and Y. Manolopoulos, Performance of nearest neighbor queries in R-trees, in: *Proc. 6th Intl. Conf. Database Theory, Lecture Notes Comput. Sci.*, Vol. 1186, 1997, pp. 394–408.

[237] M. S. Paterson and F. F. Yao, Efficient binary space partitions for hidden-surface removal and solid modeling, *Discrete Comput. Geom.*, 5 (1990), 485–503.

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

[239] M. Pellegrini, On point location and motion planning among simplices, *Proc. 25th Annu. ACM Sympos. Theory Comput.*, 1994, pp. 95–104.

[240] M. Pellegrini and P. Shor, Finding stabbing lines in 3-space, *Discrete Comput. Geom.*, 8 (1992), 191–208.

[241] M. Pocchiola and G. Vegter, Pseudo-triangulations: Theory and applications, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 291–300.

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

[243] V. Ramasubramanian and K. K. Paliwal, Fast $k$-dimensional tree algorithms for nearest neighbor search with applications to vector quantization encoding, *IEEE Trans. Signal Processing*, 40 (1992), 518–531.

[244] S. Ramaswamy and S. Subramanian, Path caching: A technique for optimal external searching, *Proc. 13th Annu. ACM Sympos. Principles Database Syst.*, 1994, pp. 25–35.

[245] J. T. Robinson, The $k$-d-b-tree: a search structure for large multidimensional dynamic indexes, *Proc. ACM SIGACT-SIGMOD Conf. Principles Database Systems*, 1981, pp. 10–18.

[246] N. Roussopoulos, S. Kelley, and F. Vincent, Nearest neighbor queries, *Proc. ACM SIGMOD Conf. on Management of Data*, 1995, pp. 71–79.

[247] N. Roussopoulos, Y. Kotidis, and M. Roussopoulos, Cubetree: Organization of and bulk incremental updates on the data cube, *Proc. ACM SIGMOD Conf. on Management of Data*, 1997, pp. 89–99.

[248] H. Sagan, *Space-Filling Curves*, Springer-Verlag, New York, 1994.

[249] G. Salton, *Automatic Text Processing*, Addison-Wesley, Reading, MA, 1989.

[250] H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS*, Addison-Wesley, Reading, MA, 1990.

[251] H. Samet, *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[252] H. W. Scholten and M. H. Overmars, General methods for adding range restrictions to decomposable searching problems, *J. Symbolic Comput.*, 7 (1989), 1–10.

[253] E. Schömer and C. Thiel, Efficient collision detection for moving polyhedra, *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995, pp. 51–60.

[254] O. Schwarzkopf, Dynamic maintenance of geometric structures made easy, *Proc. 32nd Annu. IEEE Sympos. Found. Comput. Sci.*, 1991, pp. 197–206.

[255] B. Seeger and H.-P. Kriegel, The buddy-tree: An efficient and robust access method for spatial data base systems, *Proc. 16th VLDB Conference*, 1990, pp. 590–601.

[256] R. Seidel, Backwards analysis of randomized geometric algorithms, in: *New Trends in Discrete and Computational Geometry* (J. Pach, ed.), Springer-Verlag, Heidelberg, Germany, 1993, pp. 37–68.

[257] T. Sellis, N. Roussopoulos, and C. Faloutsos, The $R^+$-tree: A dynamic index for multi-dimensional objects, *Proc. 13th VLDB Conference*, 1987, pp. 507–517.

[258] K. Sevcik and N. Koudas, Filter trees for managing spatial data over a range of size granularities, *Proc. 22nd VLDB Conference*, 1996, pp. 16–27.

[259] M. Sharir and P. K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, New York, 1995.

[260] K. Shim, R. Srikant, and R. Agrawarl, High-dimensional similarity joins, *Proc. 13th IEEE Internat. Conf. on Data Engineering*, 1997, pp. 301–311.

[261] B. K. Shoichet, D. L. Bodian, and I. D. Kuntz, Molecular docking using shape descriptors, *J. Computational Chemistry*, 13 (1992), 380–397.

[262] A. Shoshani, OLAP and statistical databases: Similarities and differences, *Proc. ACM Sympos. Principles of Database Systems*, 1997, pp. 185–196.

[263] M. Smid, Algorithms for semi-online updates on decomposable problems, *Proc. 2nd Canad. Conf. Comput. Geom.*, 1990, pp. 347–350.

[264] M. Smid, Maintaining the minimal distance of a point set in less than linear time, *Algorithms Rev.*, 2 (1991), 33–44.

[265] D. M. H. Sommerville, *Analytical Geometry in Three Dimensions*, Cambridge University Press, Cambridge, 1951.

[266] R. F. Sproull, Refinements to nearest-neighbor searching, *Algorithmica*, 6 (1991), 579–589.

[267] J. M. Steele and A. C. Yao, Lower bounds for algebraic decision trees, *J. Algorithms*, 3 (1982), 1–8.

[268] J. Stolfi, *Oriented Projective Geometry: A Framework for Geometric Computations*, Academic Press, New York, NY, 1991.

[269] V. Strassen, Algebraic complexity theory, in: *Algorithms and Complexity* (J. van Leeuwen, ed.), *Handbook of Theoretical Computer Science*, Vol. A, MIT Press, 1990, chapter 11, pp. 633–672.

[270] S. Subramanian and S. Ramaswamy, The $P$-range tree: A new data structure for range searching in secondary memory, *Proc. 6th ACM-SIAM Sympos. Discrete Algorithms*, 1995, pp. 378–387.

[271] R. E. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[272] P. M. Vaidya, Space-time tradeoffs for orthogonal range queries, *SIAM J. Comput.*, 18 (1989), 748–758.

[273] V. K. Vaishnavi and D. Wood, Rectilinear line segment intersection, layered segment trees and dynamization, *J. Algorithms*, 3 (1982), 160–176.

[274] M. J. van Kreveld, *New Results on Data Structures in Computational Geometry*, Ph.D. Dissertation, Dept. Comput. Sci., Utrecht Univ., Utrecht, Netherlands, 1992.

[275] M. J. van Kreveld and M. H. Overmars, Divided $k$-d trees, *Algorithmica*, 6 (1991), 840–858.

[276] M. J. van Kreveld and M. H. Overmars, Concatenable structures for decomposable problems, *Inform. Comput.*, 110 (1994), 130–148.

[277] J. S. Vitter and D. E. Vengroff, Efficient 3-d range searching in external memory, *Proc. 28th Annu. ACM Sympos. Theory Comput.*, 1996, pp. 192–201.

[278] J. von zur Gathen, Algebraic complexity theory, in: *Annual Review of Computer Science*, Vol. 3, Annual Reviews, Palo Alto, CA, 1988, pp. 317–347.

[279] J. Vuillemin, A unifying look at data structures, *Commun. ACM*, 23 (1980), 229–239.

[280] E. Welzl, Partition trees for triangle counting and other range searching problems, *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, 1988, pp. 23–33.

[281] E. Welzl, On spanning trees with low crossing numbers, in: *Data Structures and Efficient Algorithms, Final Report on the DFG Special Joint Initiative*, *Lecture Notes Comput. Sci.*, Vol. 594, Springer-Verlag, 1992, pp. 233–249.

[282] H. Weyl, Randbemerkungen zu Hauptproblemen der Mathematik, II, Fundamentalsatz der Alegbra und Grundlagen der Mathematik, *Math. Z.*, 20 (1924), 131–151.

[283] K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Cho, C. M. Park, and I. Y. Song, Octree-R: An adaptible octree for efficient ray tracing, *IEEE Trans. Visual. and Comp. Graphics*, 1 (1995), 343–349.

[284] D. A. White and R. Jain, Similarity indexing with the SS-tree, *Proc. 12th IEEE Intl. Conf. on Data Engineering*, 1996, pp. 516–523.

[285] M. White, N-trees: Large ordered indexes for multidimensional space, Tech. report, US Bureau of the Census, Statistical Research Division, Washington, DC, 1982.

[286] D. E. Willard, Polygon retrieval, *SIAM J. Comput.*, 11 (1982), 149–165.

[287] D. E. Willard, Lower bounds for the addition-subtraction operations in orthogonal range queries and related problems, *Inform. Comput.*, 82 (1989), 45–64.

[288] D. E. Willard, Applications of the fusion tree method to computational geometry and searching, *Proc. 3rd ACM-SIAM Sympos. Discrete Algorithms*, 1992, pp. 286–296.

[289] D. E. Willard, Applications of range query theory to relational data base join and selection operations, *J. Comput. Syst. Sci.*, 52 (1996), 157–169.

[290] D. E. Willard and G. S. Lueker, Adding range restriction capability to dynamic data structures, *J. ACM*, 32 (1985), 597–617.

[291] A. C. Yao, Space-time trade-off for answering range queries, *Proc. 14th Annu. ACM Sympos. Theory Comput.*, 1982, pp. 128–136.

[292] A. C. Yao, On the complexity of maintaining partial sums, *SIAM J. Comput.*, 14 (1985), 277–288.

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

[294] F. F. Yao, A 3-space partition and its applications, *Proc. 15th Annu. ACM Sympos. Theory Comput.*, 1983, pp. 258–263.

[295] F. F. Yao, D. P. Dobkin, H. Edelsbrunner, and M. S. Paterson, Partitioning space for range queries, *SIAM J. Comput.*, 18 (1989), 371–384.

[296] P. N. Yianilos, Data structures and algorithms for nearest neighbor search in general metric spaces, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 311–321.