Range Searching:
Emptiness, Reporting, and Approximate Counting

Thesis submitted for the degree “Doctor of Philosophy”
by
Hayim Shaul

This work was carried out under the supervision of
Professor Micha Sharir

Submitted to the Senate of Tel Aviv University
September 2011
To my wife, Iris, with much love.
To my children Noa, Yael and Omer, who taught me the joy of approximate counting.

Less is more. More’s less.
Some is more than emptiness.
Count them, more or less.

(Haiku version of this dissertation)
Abstract

In this thesis we study several range searching problems in computational geometry concentrating on range emptiness problems.

Informally, in a range searching problem we are given a set $P$ of points and are asked to preprocess them into a data structure, such that, given some query range $\gamma$ of constant description complexity, we can quickly count or report the points of $P \cap \gamma$. Emptiness searching queries, in which we only want to test whether $P \cap \gamma = \emptyset$, are intuitively easier to answer.

The thesis consists of two major parts. In the first part we discuss several problems related to emptiness searching, involving ray shooting and visibility over terrains. In the second part we present a general data structure for answering range emptiness queries with semialgebraic ranges, and describe several applications of this structure.

Applications of emptiness searching

Ray shooting amid fat triangles in 3-space. Given a set $\mathcal{O}$ of $n$ objects, the problem of ray shooting in $\mathcal{O}$ is to preprocess $\mathcal{O}$ into a data structure such that, given a query ray $\rho$ (specified by the point it emanates from, and by its orientation), the first object $o \in \mathcal{O}$ hit by $\rho$ can be found efficiently. This problem has received considerable attention due to its application in many geometric problems.

In the most general case, assuming that the objects in $\mathcal{O}$ have constant description complexity, Agarwal and Matoušek [5] showed that with linear storage and near linear preprocessing, ray shooting in $\mathcal{O}$ can be answered in $O^*(n^{3/4})$ time. This is also the best known bound for arbitrary triangles. In some special cases better bounds are known. In Chapter 2 we consider the case of ray shooting amid fat triangles (where all their angles are at least some fixed constant), and improve the query time bound to $O^*(n^{2/3})$.

We also consider a problem where the ray follows a vertical parabolic trajectory, which we refer to as stone throwing. Here we consider only the case of stone throwing amid general triangles. Surprisingly, the query time bound for stone throwing is the same as the query time bound for ray shooting (amid general triangles), namely $O^*(n^{3/4})$ even though a vertical parabola has more degrees of freedom than a line.

\footnote{The $O^*(\cdot)$ notation hides factors of the form $n^\varepsilon$, for any $\varepsilon > 0$.}
Inter-point visibility over terrains in $\mathbb{R}^2$ and $\mathbb{R}^3$. Given a set $P$ of $n$ points in $\mathbb{R}^2$ or $\mathbb{R}^3$, and a polygonal or polyhedral terrain $T$ with $m$ edges or faces, we say that two points $p_1, p_2 \in P$ are mutually visible from one another if they lie above $T$ and the segment $p_1p_2$ does not intersect $T$. The visibility graph $G$ is a graph whose vertices are the points of $P$, and two vertices are connected by an edge if the points they represent are visible from each other.

In Chapter 3 we study the problem of determining whether the visibility graph is a clique, i.e., whether every pair of points are visible. We present efficient algorithms for solving the problem in two and three dimensions, with respective running times $O(n \log n + m)$ and $O(nm \log^2 n)$.

Emptiness searching in semialgebraic settings

The main contribution of this thesis is the extension of Matoušek’s emptiness data structure for half-spaces bounded by hyperplanes [57], to any class of “simply-shaped” semialgebraic ranges. Here we seek data structures with linear (or near linear) storage and near linear preprocessing time, and sub-linear query time, where the goal is to make the query time as small as possible. This extension is similar to the extension of the range searching machinery for half-spaces bounded by hyperplanes [58] to semialgebraic ranges [5], but it faces additional technical problems, detailed later in the thesis.

Of course one can linearize the given ranges into a higher-dimensional space, i.e., map the original points to points lying on on some algebraic surface in higher dimension, as suggested in [5]. However, the query time depends on the dimension of the embedding space, and deteriorates as this dimension grows. It is therefore often desirable to construct an emptiness data structure on the points in the original space, in the hope that such a structure yields better performance.

In Chapter 4 we describe our extension of emptiness searching to the semialgebraic case, and apply it to several problems: (i) ray-shooting amid balls in 3-space, with $O^*(n^{2/3})$ query time, (ii) finding the farthest point from a convex shape in 3-space with $O^*(n^{1/2})$ query time, (iii) finding the closest point to a query point above a query line in the plane, with $O^*(1)$ query time, and (iv) fat-triangle emptiness queries in the plane. For the latter problem we give a data structure that requires linear storage and answers a query in $O^*(1)$ time, which is comparable with the simple solution that requires $O(n \log^2 n)$ storage and $O(\log^3 n)$ query time.

In Chapter 5 we show how to extend the emptiness data structure into an approximate counting data structure. Here we propose two data structures. The first structure applies the technique of Aronov and Har-Peled [15] and uses the emptiness data structure as a black box. The second data structure is an extension of the data structure of Aronov and Sharir [17] to semialgebraic ranges. We also present a solution to the problem of $k$-heaviest range searching, where the data points have weights associated with them, and, given a query range $\gamma$ and a parameter $k$, we want to report the $k$ heaviest points in $P \cap \gamma$. 
Acknowledgments

First and foremost, I wish to thank my advisor Micha Sharir for his endless guidance and help, and especially for his dedication, patience and tolerance.

I would like to thank Sariel Har-Peled, Esther Ezra, Shakhar Smorodinsky for many helpful discussions and good advice.
Chapter 1

Introduction

1.1 Overview and Background

In this thesis we study several problems in computational geometry. Specifically: (i) We derive improved solutions for ray shooting amid several restricted classes of triangles in three dimensions, and derive an efficient algorithm for stone throwing (along a parabolic trajectory) amid arbitrary triangles. (ii) We present efficient algorithms for testing for visibility of all pairs in a given set of points over a polyhedral terrain, in two and three dimensions. (iii) We study the general problems of range emptiness and range reporting with semialgebraic ranges, and obtain efficient solutions for a wide variety of applications. (iv) Finally, we apply the results in (iii) to obtain efficient data structures for approximate range counting with semialgebraic ranges.

In this introductory chapter we introduce the needed terminology and notations, review earlier works, and describe our results.

1.1.1 Range searching: overview

A range space is a pair \((X, \Gamma)\), where \(X\) is a set and \(\Gamma \subseteq 2^X\) is a collection of subsets of \(X\), called ranges. In our applications, \(X = \mathbb{R}^d\), and \(\Gamma\) is a collection of semialgebraic sets of some specific type, each having constant description complexity \([69]\). That is, each set in \(\Gamma\) is given as a Boolean combination of a constant number of polynomial equalities and inequalities of constant maximum degree. Typical examples of ranges of this kind are half-spaces, balls, cylinders, simplices, hypercubes, etc. To simplify the analysis, we assume\(^1\) that all the ranges in \(\Gamma\) are defined by a single Boolean combination, so that each polynomial \(p\) in this combination is \((d + t)\)-variate, and each range \(\gamma \in \Gamma\) has \(t\) degrees of freedom, so that if we substitute the values of these \(t\) parameters into the last \(t\) variables of each \(p\), the resulting Boolean combination defines the range \(\gamma\). This allows

\(^1\)This assumption is not essential, and is only made to simplify the presentation.
us to represent the ranges of $\Gamma$ as points in an appropriate $t$-dimensional parametric space. This assumption holds in practically all of the standard applications.

Under these special assumptions, the range space $(X, \Gamma)$ has finite VC-dimension, a property formally defined in [49]. Informally, it ensures that, for any finite subset $P$ of $X$, the number of distinct ranges of $P$, i.e., the number of distinct sets of the form $P \cap \gamma$, for $\gamma \in \Gamma$, is $O(|P|^\delta)$, where $\delta$ is (upper bounded by) the VC-dimension.

As a matter of fact, the actual range spaces that we will consider will be of the form $(P, \Gamma_P)$, where $P \subset \mathbb{R}^d$ is a finite point set, and each range in $\Gamma_P$ is the intersection of $P$ with a range in $\Gamma$. Given such a range space $(P, \Gamma_P)$, with $|P| = n$ and with $\Gamma_P$ as the underlying set of ranges, the range searching problem for $(P, \Gamma_P)$ is to build a data structure such that, for a query range $\gamma \in \Gamma$, the subset $P \cap \gamma$ can be counted, reported, or tested for emptiness efficiently.

In the performance of a range searching data structure, there is usually a trade-off between the storage and preprocessing costs on one hand, and the query time on the other hand. At one end of the spectrum, one seeks a data structure that answers a query in (poly)logarithmic time. In that case the size of the data structure will be polynomial in $n$, and the goal is to minimize (the exponent in the bound on) the size. At the other end of the spectrum, one seeks a linear-size (or near linear-size) data structure. In that case, the query time will be sub-linear in $n$, and the goal is to minimize (the exponent in the bound on) the query time. In this thesis we will focus on range searching problems where the size of the data structure is linear (or near linear in some cases).

Given a set $P \subset \mathbb{R}^d$ of $n$ points and a parameter $r > 0$, Matoušek [58] showed how to build a partition $\Pi$ of $P$, of the form $\Pi = \{(P_1, \sigma_1), \ldots, (P_m, \sigma_m)\}$, where, for each $i = 1, \ldots, m$, we have $n/r \leq |P_i| \leq 2n/r$, $P_i \subset P$, and $\sigma_i$ is a simplex containing $P_i$ such that each hyperplane crosses at most $O(r^{1-1/d})$ simplices of $\Pi$. Matoušek used this machinery to construct a partition tree for $P$, where each node $v$ of the tree stores a subset $P_v$ of $P$, and the children of each internal node $v$ correspond to the subsets of the partition of $P_v$. Then, given any range $\gamma$ which is a half-space bounded by a hyperplane, we can search the tree with $\gamma$ and report $P \cap \gamma$ as the disjoint union of $O^*(n^{1-1/d})$ “canonical” subsets $P_v$. In particular, this yields a data structure of linear size which supports range counting queries on $P$ where the query cost is $O^*(n^{1-1/d})$. Here, as in the abstract, the notation $O^*(\cdot)$ hides subpolynomial factors, which are either polylogarithmic or of the form $O(n^\varepsilon)$, for any $\varepsilon > 0$, with a constant of proportionality which depends on $\varepsilon$.

The technique in [58] relies on the linearity of the hyperplanes bounding the ranges and does not work, as is, for more general ranges. For semialgebraic ranges, Agarwal and Matoušek [5] showed how to use a lifting scheme to linearize the ranges. For example, if $P$ is a planar point set and the ranges are disks, then $P$ can be lifted to 3-space by mapping each point $(x, y)$ to $(x, y, x^2 + y^2)$. Then a disk, given by $(x - x_0)^2 + (y - y_0)^2 \leq r^2$, is mapped to the half-space $z \leq 2x_0x + 2y_0y + (r^2 - x_0^2 - y_0^2)$. See [38] for more details. Hence, querying the original set $P$ with disks is equivalent to querying the lifted set $P^*$ by
1.1 Overview and Background

Half-spaces in $\mathbb{R}^3$, which can be done with near-linear storage and $O^*(n^{2/3})$ query time (for range counting queries). In general, if the linearization transforms the input set $P$ to a set $P^*$ in $\mathbb{R}^d$ then one can perform range searching on $P$ (or rather on $P^*$) using near-linear storage and $O^*(n^{1-1/d})$ query time. Agarwal and Matoušek also present an alternative mechanism, which does not use linearization but extends directly the partition machinery to semialgebraic sets. If the input set lies in $\mathbb{R}^d$ then, with near-linear storage, the query time is $O^*(n^{1-1/b})$, where $b = d$ for $d = 2, 3, 4$, and $b = 2d - 4$ for $d \geq 5$. See [5] for more details.

If we are interested only in querying whether $P \cap \gamma$ is empty or not, then intuitively one would expect better query time. Indeed, Matoušek [57] showed that answering emptiness queries for half-spaces bounded by hyperplanes can be done in $O^*(n^{1-1/\lfloor d/2 \rfloor})$ time, with near-linear storage. The technique, however, relies on the linearity of hyperplane and does not extend, as is, to semialgebraic ranges. As a matter of fact, the improved machinery in [57] can also be used to obtain a data structure for half-space reporting queries, so that (with near-linear storage) the points of $P$ in a query half-space $h$ can be reported in time $O^*(n^{1-1/\lfloor d/2 \rfloor}) + O(|P \cap h|)$.

1.1.2 Multi-level data structures

Multi-level data structures are a useful tool in computational geometry, in which several data-structures, each testing for some specific condition, are combined into a single data structure that tests for the conjunction of these conditions (or for more general Boolean combinations). In general, given a set of objects $P$, the $i$-th level in the data structure finds a subset of objects $P_i \subset P$ that comply with the $i$-th condition. Then $P_i$ is passed to the $(i + 1)$-st level as its input, so as to filter out of $P_i$ the subset satisfying the further conditions of the subsequent levels. Usually, the $i$-th level does not build $P_i$ explicitly. Instead, it stores canonical precomputed subsets $s_1, s_2, \ldots$, such that $P_i$ can be expressed as the disjoint union of some (not too many) of them. Each canonical subset $s$ of the $i$-th level is the input to an instance of the $(i + 1)$-st level (and of its descendants).

In most applications, an appropriate construction of a multi-level data structure ensures that its performance parameters are comparable, more or less, with those of its most expensive level. For example, suppose that the data structure has $t$ levels, that each level is a partition tree, constructed for half-space ranges over some canonical subset of the preceding level, and that the $i$-th level caters to half-space ranges in $d_i$ dimensions, for $i = 1, \ldots, t$. Put $d = \max_i d_i$, $c_i = 1 - 1/d_i$, and $c = \max_i c_i = 1 - 1/d$. Then the cost $Q_i(n)$ of a query starting at the $i$-th level of the data structure satisfies the recurrence $Q_i(n) = O(r)Q_{i+1}(n/r) + O(r^{1-1/d_i})Q_i(n/r)$, where $r$ is the partition parameter used in the construction (assume for simplicity that a fixed constant parameter is used for all levels). It is then easy to check that $Q_1(n) = O^*(n^c)$. Similar reasoning applies to the storage and preprocessing costs, showing both of them to be $O^*(n)$. The same considerations apply to more general kinds of multi-level structures.
1.1.3 Parametric searching

Parametric searching is an optimization technique, originally proposed by Megiddo [61], for finding the optimum value \( x^* \) of some objective function \( \Gamma(x) \), for which there exists a decision procedure \( D(x_0) \) which, given any concrete value \( x_0 \) of \( x \), can efficiently determine whether \( x_0 < x^* \), \( x_0 = x^* \) or \( x_0 > x^* \). Typically, there is a finite set \( X \) of candidate values for \( x^* \), so, in principle, we can simply enumerate \( X \) and run binary search on it, using the decision procedure to guide the search, to find \( x^* \). The problem is that \(|X|\) is in general too large so that it is too expensive to enumerate \( X \).

The main idea in parametric searching is to run \( D \) on the unknown optimum value \( x^* \). We assume that this generic execution proceeds by computing various expressions \( f(x^*) \) of \( x^* \), and that the branchings that it takes are determined by comparing such an expression \( f(x^*) \) to 0 and proceeding according to the outcome of the comparison. To implement such a comparison, we find the roots \( x_1 < x_2 < \cdots < x_k \) of \( f(x) \), and locate \( x^* \) amid these roots by calling \( D \) with each \( x_i \) as input, which determines whether \( x^* < x_i \), \( x^* = x_i \) or \( x^* > x_i \). After testing all roots we find two consecutive roots \( x_i, x_{i+1} \), such that \( x_i < x^* < x_{i+1} \) (if we have not already identified that \( x^* \) is equal to a root), and this determines the sign of \( f(x^*) \), allowing us to resolve the comparison and to continue with the generic execution of \( D \). In most applications, this execution will terminate at some comparison for which one of the corresponding roots is the desired \( x^* \). A naive implementation of this technique results in an algorithm with running time \( O(CT) = O(T^2) \), where \( C \) is the number of comparisons made by \( D \) and \( T \) is the running time of \( D \). Megiddo suggested to improve this running time by replacing \( D \) with a (serial execution of a) parallel version \( D_p \) running on \( u \) processors, whose parallel running time is \( T_p \). Then at each step there are (at most) \( u \) independent comparisons carried out by the \( u \) processors. We collect the roots of the polynomials of these \( u \) comparisons, and run binary search through the sequence of all these \( O(u) \) roots, to obtain two consecutive roots \( x_i, x_{i+1} \) such that \( x_i < x^* < x_{i+1} \). This allows us to resolve all the comparisons in this parallel step and we can proceed to the next parallel step. Each parallel step takes only \( O(u + T \log u) \) time, for a total of \( O(uT_p + TT_p \log u) \) time.

There are many variants and enhancements of this technique; see [73] for details.

An important application of parametric searching is to the ray shooting problem reviewed in the next section. Agarwal and Matoušek [4] proposed a general approach to reduce ray shooting amid a set of objects \( O \) into segment emptiness testing, using parametric searching. Specifically, let \( \rho \) be a given query ray, emanating from a point \( q \) in direction \( v \). Our goal is to find the smallest positive number \( t^* \) such that \( q + t^*v \) lies on (the boundary of) an object of \( O \). To apply parametric searching, we need a decision procedure \( D \) which, for a given \( t > 0 \), determines whether the segment \([q, q + tv]\) is empty, i.e., does not intersect any object of \( O \). If so, \( t^* > t \) and otherwise \( t^* \leq t \). The algorithm then applies parametric searching to a parallel version of \( D \) with the unknown \( t^* \) as input, which results in a ray shooting procedure whose query time is close to that for segment emptiness testing.
1.1 Overview and Background

1.1.4 Ray shooting

As just mentioned, the ray shooting problem is to preprocess a set of objects such that the first object hit by a query ray can be determined efficiently. The ray shooting problem has received considerable attention in the past because of its applications in computer graphics and other geometric problems. The planar case has been studied thoroughly. Optimal solutions, which answer a ray shooting query in $O(\log n)$ time using $O(n)$ space, have been proposed for some special cases [22, 26, 50]. For an arbitrary collection of segments in the plane, the best known algorithms answer a ray shooting query in time $O\left(\frac{n}{\sqrt{s}} \log^{O(1)} n\right)$ using $O^*(s)$ space and preprocessing [2, 9, 19], where $s$ is a parameter that can vary between $n$ and $n^2$.

The three-dimensional ray shooting problem seems much harder and it is still far from being fully solved. Most studies of this problem consider the case where the given set is a collection of triangles. If these triangles are the faces of a convex polyhedron, then an optimal algorithm (with $O(n)$ storage and $O(\log n)$ query time) can be obtained using the hierarchical decomposition scheme of Dobkin and Kirkpatrick [37]. If the triangles form a polyhedral terrain (an $xy$-monotone piecewise-linear surface), then the technique of Chazelle et al. [24] yields an algorithm that requires $O^*(n^2)$ space and answers ray shooting queries in $O(\log n)$ time. The best known algorithm for the general ray shooting problem (involving triangles) is due to Agarwal and Matoušek [5]; it answers a ray shooting query in time $O^*(\frac{n}{s^{1/3}})$, with $O^*(s)$ space and preprocessing, where the parameter $s$ can range between $n$ and $n^4$. See [5, 9] for more details. A variant of this technique was presented in [10] for the case of ray shooting amid a collection of convex polyhedra; see [52] for a more recent treatment of this case.

On the other hand, there are certain special cases of the 3-dimensional ray shooting problem which can be solved more efficiently. For example, if the objects are planes or half-planes, ray shooting amid them can be performed in time $O^*(\frac{n}{s^{1/3}})$, with $O^*(s)$ space and preprocessing; see [4] for details. If the objects are horizontal fat triangles or axis-parallel polyhedra, ray shooting can be performed in time $O(\log n)$ using $O^*(n^2)$ space; see [35] for details. If the objects are spheres, ray shooting can be performed in time $O^*(1)$ with $O^*(n^3)$ space; see [63]. In both cases involving ray shooting in terrains and ray shooting in axis-parallel polyhedra, one can also get the standard trade-off between query time and storage—see, e.g., [33].

As noted in the preceding subsection, the problems of ray shooting and segment emptiness searching are closely related. As a matter of fact, many of the results reviewed above are achieved by this parametric searching machinery, and the main problem that each of them solves is that of designing an efficient data structure for answering segment emptiness queries for the specific type of obstacles under consideration.
1.1.5 Cuttings

Many of the results reviewed so far, and many of the techniques that we will later develop in the thesis, are based on a general-purpose space decomposition technique known as *cutting*.

Given a finite collection $\Gamma$ of $n$ semialgebraic surfaces in $\mathbb{R}^d$, as in Section 1.1.1, and a parameter $r < n$, a *(1/r)*-cutting for $\Gamma$ is a partition $\Xi$ of $\mathbb{R}^d$ (or of some portion of $\mathbb{R}^d$) into a finite number of relatively open pairwise disjoint and simply shaped cells of dimensions $0, 1, \ldots, d$, so that each cell is crossed by at most $n/r$ ranges of $\Gamma$, where a range $\gamma \in \Gamma$ is said to cross a cell $\sigma$ if $\gamma \cap \sigma \neq \emptyset$, but $\gamma$ does not fully contain $\sigma$. Technically, each cell of $\Xi$ is required to be homeomorphic to a ball, and to have constant description complexity. In many cases we further require that each cell $\sigma$ be defined by at most $c = O(1)$ surfaces of $\Gamma$ itself, in the sense that they form a minimal family which specifies the boundary of $\sigma$ (see below for details) and none of these surfaces crosses $\sigma$. We will also need to consider weighted *(1/r)*-cuttings, where each range $\gamma \in \Gamma$ has a positive weight $w(\gamma)$, and each cell of $\Xi$ is crossed by ranges whose total weight is at most $W/r$, where $W = \sum_{\gamma \in \Gamma} w(\gamma)$ is the overall weight of all the ranges in $\Gamma$.

To illustrate this definition, consider the case when $\Gamma$ consists of hyperplanes. As shown in [25, 56], in this case $\Gamma$ admits a *(1/r)*-cutting consisting of $O(r^d)$ simplices. (A simple counting argument shows that this bound is tight.) The construction proceeds by taking a random sample $R$ of $O(r)$ hyperplanes of $\Gamma$, by forming its arrangement $A(R)$, and by triangulating each cell into simplices, using bottom-vertex triangulation [28]. This results in $O(r^d)$ simplices, and (with an appropriate choice of the constant of proportionality) each of them is expected to be crossed by at most $n/r$ hyperplanes. However, some simplices might be crossed by more hyperplanes. We “fix” each such simplex $\sigma$ by taking a second random sample from the hyperplanes that cross $\sigma$, triangulate the arrangement of this sample, and clip its cells to within $\sigma$. The analysis in [25, 56] shows that the expected overall number of new simplices is still $O(r^d)$ and, with constant probability, each of them is crossed by at most $n/r$ hyperplanes of $\Gamma$.

This approach works for any collection $\Gamma$ of semialgebraic surfaces of constant description complexity, but the bottleneck is in decomposing each cell of the arrangement $A(R)$ of the random sample $R$ into ball-like subcells of constant description complexity. We refer to such subcells as elementary cells; see below for details. This is done using the general-purpose technique of vertical decomposition, as introduced in [23]; see also [69] for details. The current best upper bounds on the size of such a decomposition, for an arrangement of $r$ surfaces as above, is $O(r^2)$ for $d = 2$, $O^*(r^3)$ for $d = 3$ [23], and $O^*(r^{2d-4})$ for $d \geq 4$ [53]. This leads to the same asymptotic bounds on the size of the *(1/r)*-cuttings for such a collection of surfaces. The bounds are probably not tight for $d \geq 5$, but improving them has been a major open problem for almost two decades.
1.1.6 Shallow ranges

A range $\gamma \in \Gamma$ is called $k$-shallow with respect to a set $P$ of points in $\mathbb{R}^d$ if $|\gamma \cap P| \leq k$. There is a close connection between the running time of range searching algorithms for shallow ranges and the complexity of the decomposition of the shallow levels of an arrangement of the surfaces bounding a subcollection of the ranges, where the level of a point in such an arrangement is the number of regions it is contained in. More precisely, we want to take each cell of the arrangement at level $\leq k$ and decompose it into subcells of constant description complexity. There are several techniques for obtaining such a decomposition, such as the bottom-vertex triangulation in the case of hyperplanes [28], but the only known general-purpose technique is vertical decomposition [23], already mentioned at the end of the preceding subsection. In this technique, which applies to arrangements of general semi-algebraic ranges (of constant description complexity), each cell is partitioned into subcells by erecting vertical walls from lower-dimensional features on its boundary, and by further decomposing the resulting “prisms” by recursing on the dimension. See [64, 65] for more details.

In some cases the bounds for the complexity of decomposition of the $k$ shallow levels of the arrangement, are known to be better than the complexity of the decomposition of the entire arrangement. For the case of an arrangement of $n$ hyperplanes in $\mathbb{R}^d$ (where the corresponding ranges are, say, the upper half-spaces bound by these hyperplanes), Clarkson and Shor [29] proved that the number of vertices of level at most $k$ is $O(n^{\lfloor d/2 \rfloor}k^{\lceil d/2 \rceil})$. Using the bottom vertex triangulation, we get a decomposition for this region of the arrangement that has the same description complexity. In the general case, however, it is not known how to decompose the shallow level into a small number of cells. Of course, one can always decompose the entire arrangement, for example by vertical decomposition, which leads to $O^*(n^d)$ cells for $d \leq 4$, and $O^*(n^{2d-4})$ cells [23] (with the improvement of [53]) otherwise. It is generally believed, though, that when $k$ is small, the actual number of cells is much smaller.

The area below the lower envelope (which can also be thought of as level 0) is especially interesting when dealing with emptiness problems. To decompose the area below the lower envelope one can simply decompose the arrangement induced on the lower envelope and then extend each cell downwards into a semi-unbounded prism-like cell. For the case of hyperplanes, the lower envelope (and the level 0) can be decomposed into $O(n^{\lceil d/2 \rceil})$ simplices. The case of general surfaces is much harder and little is known about decompositions of lower envelopes of general surfaces. The planar case is simple, as the decomposition of level 0 is proportional to the complexity of the lower envelope, which is near linear [69]. For general surfaces in three or higher dimensions the complexity of the lower envelope of $n$ surfaces in $\mathbb{R}^d$ (of constant description complexity) is $O^*(n^{d-1})$ [68]. In $\mathbb{R}^3$, the decomposition of the lower envelope is of complexity $O^*(n^2)$, but in higher dimension there is no better bound than the one obtained by decomposing the entire arrangement.
1.1.7 Elementary cells

We define, as in [5], an elementary cell in $\mathbb{R}^d$ to be a connected relatively open semialgebraic set of some dimension $k \leq d$, which is homeomorphic to a ball and has constant description complexity. As above, we assume, for simplicity, that the elementary cells are defined by a single Boolean combination involving $t$ free variables, and each cell is determined by fixing the values of these $t$ parameters.

Elementary cells are the building blocks of the cuttings discussed in the preceding subsections. Typically, such a cutting is obtained by taking a random sample $R$ of $r$ input surfaces, and by forming their arrangement $A(R)$. The complexity of a single cell of $A(R)$ might be very large, in fact it might even be (much) more than linear in the number of the surfaces (for example, a cell in an arrangement of hyperplanes), and even not simply connected. Informally, one would like to argue that a cell $\sigma$ of $A(R)$ is not crossed by any surface of the sample $R$, and therefore it cannot be crossed by too many of the original surfaces. However, the random sampling theory justifies this claim only when $\sigma$ has constant description complexity, or, alternatively, is defined by only a constant number of input surfaces; see [28, 49] for more details. Since elementary cells have these properties, refining the cells of an arrangement into elementary cells is therefore a fundamental step in the construction of a cutting. As already described in the preceding subsection, this is generally done using vertical decomposition. There are also other decomposition schemes that are less general than the vertical decomposition. For example, bottom-vertex triangulation [28] can be used to partition the convex cells of an arrangement of hyperplanes into simplices. Cylindrical algebraic decomposition [30] is an older technique which produces elementary cells similar to those in the vertical decomposition, but their number is typically much larger; it has the advantage that its output is a cell complex, which is not the case for vertical decomposition.

1.1.8 Elementary cell partitions

Let $P$ be a set of $n$ points in $\mathbb{R}^d$ and let $\Gamma$ be a class of semialgebraic ranges (of constant description complexity). An elementary cell partition of $P$ is a collection $\Pi = \{(P_1, s_1), \ldots, (P_m, s_m)\}$, for some integer $m$, such that (i) $\{P_1, \ldots, P_m\}$ is a partition of $P$ (into pairwise disjoint subsets), and (ii) each $s_i$ is an elementary cell that contains the respective subset $P_i$. In general, the cells $s_i$ need not be disjoint. Usually, one also specifies a parameter $r \leq n$, and requires that $n/r \leq |P_i| \leq 2n/r$ for each $i$, so $m = O(r)$.

Elementary cell partitions are basic building blocks of range searching data structures. Given a query range $\gamma$, we compare it to each elementary cell $s_i$. If $\gamma$ fully contains $s_i$ then it must contain all the points of $P_i$. If $\gamma$ does not intersect $s_i$ then it cannot contain any of the points in $P_i$. Otherwise ($\gamma$ intersects $s_i$ but does not contain it) we process $\gamma$ recursively within a similar elementary cell partition constructed for $P_i$. An efficient cell partition is one for which, for each range $\gamma \in \Gamma$, the number of subsets $P_i$ that have to be
1.1 Overview and Background

processed recursively is relatively small.

Given such an efficient partition scheme, it can be turned into an efficient partition tree. Each node of the tree stores some subset of \( P \). The root stores the entire \( P \), together with an elementary cell partition of \( P \). Each subset \( P_i \) of the partition is stored at a corresponding child of the root and the process continues recursively until we reach subsets of sufficiently small size. The efficiency of a range searching query with some \( \gamma \in \Gamma \), using the partition tree, depends on the number of nodes that \( \gamma \) visits, which in turn depends on the maximum number of cells in any single partition that \( \gamma \) crosses (intersects but does not fully contain).

Matoušek [58] has shown how to build an efficient partition tree for hyperplanes in \( d \) dimensions, where the tree requires \( O(n) \) storage, can be built in \( O(n \log n) \) time and a half-space range query can be answered in time \( O^*(n^{1-1/d}) \). This is based on an elementary cell partition scheme for hyperplanes, which, given a set \( P \) of \( n \) points in \( \mathbb{R}^d \) and a parameter \( r \leq n \), produces a partition \( \Pi = \{(P_1, s_1), \ldots, (P_m, s_m)\} \), where \( n/r \leq |P_i| \leq 2n/r \) for each \( i \), \( s_i \) is a simplex containing \( P_i \), and each hyperplane crosses at most \( O^*(r^{1-1/d}) \) of the \( O(r) \) simplices \( s_i \).

For ranges other than hyperplanes, Agarwal and Matoušek [5] presented an elementary cell partition scheme in which the number of cells crossed by a query range \( \gamma \) is \( O^*(r^{1-\delta}) \), where \( b = d \) for \( d = 2, 3, 4 \), and \( b = 2d - 4 \) for \( d \geq 5 \). (Their original bound was \( b = 2d - 3 \) and it was later improved by Koltun [53].) This in turn leads to the construction of an efficient partition tree with the same storage and preprocessing requirements, so that a range searching query with a range \( \gamma \in \Gamma \) can be answered in \( O^*(n^{1-1/b}) \) time. Agarwal and Matoušek also presented an alternative technique, in which they linearize the problem into a higher-dimensional space \( \mathbb{R}^\delta \), for \( \delta > d \), where each query range \( \gamma \) is transformed into a half-space. In this linearization each point \( p \in \mathbb{R}^d \) is mapped into a point \( p^* \in \mathbb{R}^\delta \) that lies on a \( d \)-dimensional surface \( S \). They then apply Matoušek’s partitioning scheme for half-spaces to the transformed point set in \( \mathbb{R}^\delta \). See [5] for more details.

Without going into details (which can be found in [5, 58]), we note that the exponent in the bound on the query time in such partition trees is closely related to the complexity of an elementary cell decomposition of an arrangement of surfaces bounding ranges in \( \Gamma \). Specifically, for a set \( R \) of \( r \) ranges of \( \Gamma \), let \( \zeta(r) \) denote an upper bound on the number of elementary cells in a decomposition of the cells of \( A(R) \). Then one can obtain an elementary cell partition of a point set \( P \), as above, so that each range \( \gamma \in \Gamma \) crosses (intersects but does not fully contain) at most \( O(r/\zeta^{-1}(r)) \) cells \( s_i \) of the partition. In particularly, if \( \zeta(r) = O(r^d) \), then this crossing number is \( O(r^{1-1/\delta}) \). This in turn leads to query time \( O^*(n^{1-1/\delta}) \) in the resulting partition tree. The specific bounds mentioned above are consequences of the bounds \( \zeta(r) = O(r^d) \) for hyperplanes (which is the number of simplices produced by the bottom-vertex triangulation technique), and \( \zeta(r) = O^*(r^d) \) for general semialgebraic ranges, as described above.

As described above, when dealing with emptiness or reporting queries better elementary
cell partitions can be obtained. Specifically, they are constructed via an elementary cell
decomposition of only the portion of $\mathbb{R}^d$ below the lower envelope of a sample $R$ of $r$
ranges from $\Gamma$. (More generally, what is needed is an elementary cell decomposition of the
complement of the union of the ranges in $R$; see Chapter 4 for details.) For example, for the
case of half-spaces bounded by hyperplanes in $\mathbb{R}^d$, the complexity of such a decomposition is
$O(r^{\lfloor d/2 \rfloor})$, and this leads to a partition tree that answers half-space range emptiness queries
in $O^*(n^{1-1/\lfloor d/2 \rfloor})$ time; see [57]. Sharper bounds of this kind are not always available
for general semialgebraic ranges. Moreover, adapting the machinery for emptiness (or
reporting) queries to the semialgebraic case faces several additional technical problems,
which we address and solve in Chapter 4.

1.1.9 $(\nu, \alpha)$-samples and shallow $\varepsilon$-nets.

For some of the range searching problems that we study in this thesis, we need the following
useful extension of the notion of $\varepsilon$-nets. For this, we recall the result of Li et al. [55], and
adapt it, similar to the recent observations in [48], to obtain the desired variant of $\varepsilon$-nets.

Let $(X, R)$ be a range space of finite VC-dimension $\delta$, and let $0 < \alpha, \nu < 1$ be two given
parameters. Consider the distance function (which is actually a metric)

$$d_{\nu}(r, s) = \frac{|r - s|}{r + s + \nu}, \quad \text{for } r, s \geq 0.$$

A subset $N \subseteq X$ is called a $(\nu, \alpha)$-sample if for each $R \in R$ we have

$$d_{\nu} \left( \frac{|X \cap R|}{|X|}, \frac{|N \cap R|}{|N|} \right) < \alpha.$$

**Theorem 1.1.1 (Li et al. [55])** A random sample $N$ of

$$O \left( \frac{1}{\alpha^2 \nu} \left( \delta \log \frac{1}{\nu} + \log \frac{1}{q} \right) \right)$$

elements of $X$ is a $(\nu, \alpha)$-sample with probability at least $1 - q$, for an appropriate constant
of proportionality.

Har-Peled and Sharir [48] (see also Har-Peled [47]) show that, by appropriately choosing
$\alpha$ and $\nu$, various standard constructs, such as $\varepsilon$-nets and $\varepsilon$-approximations, are special cases
of $(\nu, \alpha)$-samples. Here we follow a similar approach, and show the existence of small-size
shallow $\varepsilon$-nets, a new notation defined as follows.

Let $(X, R)$ be a range space of finite VC-dimension $\delta$, and let $0 < \varepsilon < 1$ be a given
parameter. A subset $N \subseteq X$ is a shallow $\varepsilon$-net if it satisfies the following two properties,
for some absolute constant $c$. 

1. For all $r, s \in X$ we have

$$d_{\nu}(r, s) = \frac{|r - s|}{r + s + \nu}, \quad \text{for } r, s \geq 0.$$

2. For all $R \in R$ we have

$$d_{\nu} \left( \frac{|X \cap R|}{|X|}, \frac{|N \cap R|}{|N|} \right) < \alpha.$$
(i) For each \( R \in \mathcal{R} \) and for any parameter \( t \geq 0 \), if \(|N \cap R| \leq t \log \frac{1}{\varepsilon} \) then \(|X \cap R| \leq c(t + 1)\varepsilon |X|\).
(ii) For each \( R \in \mathcal{R} \) and for any parameter \( t \geq 0 \), if \(|X \cap R| \leq t\varepsilon |X| \) then \(|N \cap R| \leq c(t + 1)\log \frac{1}{\varepsilon} \).

Note the difference between shallow and standard \( \varepsilon \)-nets: Property (i) (with \( t = 0 \)) implies that a shallow \( \varepsilon \)-net is also a standard \( \varepsilon \)-net (possibly with a recalibration of \( \varepsilon \)). Property (ii) has no parallel in the case of standard \( \varepsilon \)-nets — there is no guarantee how a standard net interacts with small ranges.

**Theorem 1.1.2** A random sample \( N \) of

\[
O \left( \frac{1}{\varepsilon} \left( \delta \log \frac{1}{\varepsilon} + \log \frac{1}{q} \right) \right)
\]

elements of \( X \) is a shallow \( \varepsilon \)-net with probability at least \( 1 - q \), for an appropriate constant of proportionality.

**Proof:** Take \( \alpha = 1/2 \), say, and calibrate the constants in the size of \( N \) to guarantee, with probability \( 1 - q \), that \( N \) is an \((\varepsilon, 1/2)\)-sample. Assume that this is indeed the case. For a range \( R \in \mathcal{R} \), put \( X_R = \frac{|X \cap R|}{|X|} \) and \( N_R = \frac{|N \cap R|}{|N|} \). We have

\[
d_\varepsilon(X_R, N_R) = \frac{|X_R - N_R|}{X_R + N_R + \varepsilon} < \frac{1}{2}.
\]

That is,

\[
|X_R - N_R| < \frac{1}{2}(X_R + N_R + \varepsilon),
\]

or

\[
X_R < 3N_R + \varepsilon, \quad \text{and, symmetrically,} \quad N_R < 3X_R + \varepsilon.
\]

Assuming that \( \varepsilon < q^c \), for some constant \( c > 0 \), this is easily seen to imply properties (i) and (ii). For (i), let \( R \) be a range for which \(|N \cap R| \leq t \log \frac{1}{\varepsilon} \); that is, \( N_R \leq \beta t \varepsilon \), for some absolute constant \( \beta \) (proportional to the VC-dimension and the parameter \( c \)). Then

\[
|X \cap R| = |X| \cdot X_R < |X|(3N_R + \varepsilon) \leq (3\beta t + 1)\varepsilon |X|.
\]

For (ii), let \( R \) be a range for which \(|X \cap R| \leq t\varepsilon |X| \); that is, \( X_R \leq t\varepsilon \). Then

\[
|N \cap R| = |N| \cdot N_R < |N|(3X_R + \varepsilon) \leq (3t + 1)\varepsilon |N| \leq (3t + 1)\gamma \log \frac{1}{\varepsilon},
\]

for another absolute constant \( \gamma \) (again, proportional to the VC-dimension and the parameter \( c \)).
1.2 Our Results

1.2.1 Ray shooting amid fat triangles and stone throwing

In Chapter 2 we consider several special cases of the ray shooting problem, including the cases of ray shooting in three dimensions amid a collection of arbitrary fat triangles (i.e., triangles whose angles are at least some fixed constant $\alpha > 0$) and amid a collection of triangles stabbed by a common line. We present an improved solution for the case where only near-linear storage is allowed. Specifically, we improve the query time from the best known general bound of $O^*(n^{3/4})$ [4] to $O^*(n^{2/3})$, using $O^*(n)$ space and preprocessing. Curiously, at the other end of the trade-off, we did not manage to improve upon the general case, and so $O^*(n^4)$ storage is still required for logarithmic-time queries. These two extreme bounds lead to a different trade-off, which is also presented in Chapter 2.

Next we study another problem, of shooting along arcs amid triangles in three dimensions, which we refer to as stone throwing. In this problem we are given a set $T$ of $n$ triangles in $\mathbb{R}^3$, and we wish to preprocess them into a data structure that can answer efficiently stone throwing queries, where each query specifies a point $p \in \mathbb{R}^3$ and an initial velocity vector $v \in \mathbb{R}^3$; these parameters define a vertical parabolic trajectory traced by a stone thrown from $p$ with initial velocity $v$ under gravity (which we assume to be exerted in the negative $z$-direction), and the query asks for the first triangle of $T$ to be hit by this trajectory. The query has six degrees of freedom, but the parabola $\pi$ that contains the stone trajectory has only five degrees of freedom, which is one more than the number of degrees of freedom for lines in space.

Unlike the special case of ray shooting studied earlier in Chapter 2, we consider here the general case where the triangles of $T$ are arbitrary, and present a solution that uses near-linear storage and answers stone-throwing queries in time $O^*(n^{3/4})$. These performance bounds are interesting, since they are identical to the best bounds known for the general ray-shooting problem, even though the stone-throwing problem appears to be harder since it involves one additional degree of freedom. At present we do not know whether the stone throwing problem admits a faster solution for the special classes of triangles considered in the first part of Chapter 2. Moreover, at the other extreme end of the trade-off, where we wish to answer stone-throwing queries in $O(\log n)$ time, the best solution that we have requires $O^*(n^5)$ storage, which is larger, by a factor of $n$, than the best known solution for the ray-shooting problem. (This latter solution is not presented in Chapter 2.)

The method can be easily extended to answer shooting queries along other types of trajectories, with similar performance bounds (i.e., near linear storage and preprocessing and near $n^{3/4}$ query time). In fact, this holds for shooting along the graph of any univariate algebraic function of constant degree that lies in any vertical plane.

The results of Chapter 2 appeared in [70].
1.2 Our Results

1.2.2 Inter-point visibility over terrains in two and three dimensions

Given a set $\mathcal{O}$ of obstacles in $\mathbb{R}^d$, we say that two points $p_1$ and $p_2$ are **mutually visible** if the segment $p_1p_2$ does not intersect any object $o \in \mathcal{O}$. We say that two objects $o_1$ and $o_2$ are mutually visible if there exist two points $p_1 \in o_1$ and $p_2 \in o_2$ such that $p_1$ and $p_2$ are mutually visible. Given a set $P$ of objects and a set $\mathcal{O}$ of obstacles, the associated **visibility graph** is a graph whose vertices are the objects of $P$, and two vertices are connected by an edge if the objects that they represent are mutually visible.

The problem of computing the visibility graph of an input scene was studied mainly in two dimensions [18, 54, 74]. Nilsson [66] showed a connection between visibility graphs and robot motion planning. Ghosh and Mount [45] considered the case where the obstacles are polygons in the plane with a total of $n$ vertices, and where the vertices of the visibility graph are the vertices of the polygons; they showed how to compute the visibility graph in $O(n \log n + k)$ time, where $k$ is the number of edges in the graph. Moet et al. [62] also studied several variations of this problem.

Given a set $D$ of data objects and a set $\mathcal{O}$ of obstacles in $\mathbb{R}^d$, the **mutual visibility** problem is to determine whether every pair of objects are mutually visible. This is a somewhat easier problem than constructing the whole visibility graph; we just want to determine whether this graph is complete.

In Chapter 3 we consider the case where the set of obstacles form a polygonal or polyhedral terrain in two or three dimensions, with $n$ edges and where $P$ is a set of $m$ points, all lying above the terrain. We present efficient algorithms for determining whether all pairs of points in $P$ are mutually visible. In two dimensions our algorithm runs in $O(m \log m + n)$ time. In three dimensions we show that the problem is $\text{3SUM}$-hard [44] and present an algorithm that solves this problem in $O(nm \log^2 m)$ time.

1.2.3 Emptiness queries with semialgebraic ranges

In range emptiness searching problems, we are given a set $P$ of $n$ points in $\mathbb{R}^d$, and wish to preprocess it into a data structure that supports efficient range emptiness queries, in which we specify a range $\sigma$, which is a semialgebraic set in $\mathbb{R}^d$ of constant description complexity, taken from some fixed class, and wish to determine whether $P \cap \sigma = \emptyset$. Range emptiness searching (with semialgebraic ranges) arises in many applications, as will be demonstrated in Chapter 4 of the thesis. The special case where the ranges are half-spaces bounded by hyperplanes has been treated by Matoušek [57] who has presented a solution which is more efficient than the best known techniques for standard range searching queries. Specifically, as already mentioned, with near-linear storage, half-space range queries on a set of $n$ points in $\mathbb{R}^d$ can be answered in time $O^*(n^{1-1/d})$ [57]. In contrast, half-space range emptiness searching can be answered, with near-linear storage, in time $O^*(n^{1-1/[d/2]})$.

Actually, Matoušek has also considered in [57] half-space range-reporting queries, in
which, given a half-space query $\sigma$, we want to report the points of $P \cap \sigma$. As shown in [57], this problem too can be answered more efficiently, in time $O^*(n^{1-1/[d/2]}) + O(k)$, where $k = |P \cap \sigma|$. See also [6] for a dynamic version of the problem.

The main technical contribution of Chapter 4 is an extension of Matoušek’s range emptiness and reporting data structures to the case of general semialgebraic ranges (of constant description complexity). We present a general technique, as well as several ad-hoc solutions, to range emptiness and reporting problems of this kind, which are considerably more efficient than the solutions for standard range searching queries with semialgebraic ranges [5].

Ray shooting amid balls. A motivating application of this study is ray shooting amid balls in $\mathbb{R}^3$, where we want to construct a data structure of linear size with near-linear preprocessing, which supports ray shooting queries in sublinear time. Typically, in problems of this sort, the bound on the query time is some fractional power of $n$, the number of objects, and the goal is to make the exponent as small as possible. For example, as already mentioned in Section 1.2.1, ray shooting amid a collection of $n$ arbitrary triangles can be performed in $O^*(n^{3/4})$ time (with linear storage) [5]. Better solutions are known for various special cases, such as those mentioned in Section 1.2.1.

At the other end of the spectrum, one is interested in ray shooting algorithms and data structures where a ray shooting query can be performed in logarithmic or polylogarithmic time (or even $O(n^\varepsilon)$ time, for any $\varepsilon > 0$; this is $O^*(1)$ in our shorthand notation). In this case, the goal is to reduce the storage (and preprocessing) requirements as much as possible. For example, for arbitrary triangles (and even for the special case of fat triangles), the best known bound for the storage requirement (with logarithmic query time) is $O^*(n^4)$ [9, 5]. For balls, Mohaban and Sharir [63] gave an algorithm with $O^*(n^3)$ storage and $O^*(1)$ query time. However, when only linear storage is used, the previously best known query time (for balls) is $O^*(n^{3/4})$ (as in the case of general triangles). In Chapter 4 we show, as an application of our general range emptiness machinery, that this can be improved to $O^*(n^{2/3})$ time.

As mentioned in Section 1.1.3, when answering a ray-shooting query for a set $S$ of input objects, one can reduce the problem to that of answering segment emptiness queries, following the parametric searching scheme proposed by Agarwal and Matoušek [4] (see also Megiddo [61] for the original underlying technique).

A standard way of performing the latter kind of queries is to switch to a dual parametric space, where each object in the input set is represented by a point. A segment $e$ in $\mathbb{R}^3$ is mapped to a surface $\sigma_e$, which is the locus of all the points representing the objects that $e$ touches (without penetrating into their interior). Usually, $\sigma_e$ partitions the dual space into two portions, one, $\sigma_e^+$, consisting of points representing objects whose interior is intersected by $e$, and the other, $\sigma_e^-$, consisting of points representing objects that $e$ avoids. The segment-emptiness problem thus transforms into a range-emptiness query: Does $\sigma_e^+$
contain any point representing an input object?

**Range reporting and emptiness searching.** As reviewed above, range-emptiness queries of this kind have been studied by Matoušek [57] (see also Agarwal and Matoušek [6]), but only for the case where the ranges are half-spaces bounded by hyperplanes. For this case, Matoušek has established a so-called shallow-cutting lemma, that shows the existence of a \((1/s)\)-cutting that covers the complement of the union of any \(m\) given half-space ranges, whose size is significantly smaller than the size of a \((1/s)\)-cutting that covers the entire space. This lemma provides the basic tool for partitioning a point set \(P\), in the style of [58] (see Section 1.1.1), so that shallow hyperplanes (those containing at most \(n/r\) points of \(P\) below them, say, for some given parameter \(r\)) cross only a small number of cells of the partition (see below for more details). This in turn yields a data structure, known as a shallow partition tree, that stores a recursive partitioning of \(P\), which enables us to answer more efficiently half-space range reporting queries for shallow hyperplanes, and thus also half-space range emptiness queries. Using this approach, the query time (for emptiness) improves from the general half-space range searching query cost of \(O^*(n^{1-1/d})\) to \(O^*(n^{1-1\lfloor d/2 \rfloor})\), as already mentioned. Reporting takes \(O^*(n^{1-1\lfloor d/2 \rfloor} + k)\), where \(k\) is the output size.

Consequently, as mentioned in Section 1.1.1, one way of applying this machinery for more general semialgebraic ranges is to “lift” the set of points and the ranges into a higher-dimensional space. However, if the space in which the ranges are linearized has high dimension, the resulting range reporting or emptiness queries become significantly less efficient. Moreover, in many applications, the ranges are Boolean combinations of polynomial (equalities and) inequalities, which creates additional difficulties in linearizing the ranges, resulting in even worse running time.

An alternative technique is to give up linearization, and instead work in the original space. As follows from the machinery of [57] (and further elaborated later in Chapter 4), this requires, as a major tool, the (existence and) construction of a decomposition of the complement of the union of \(m\) given ranges (in the case of segment emptiness, these are the ranges \(\sigma^+_e\), for an appropriate collection of segments \(e\), into a small number of elementary cells (in the terminology of [5]—see also Section 1.1.7). Here we face, especially in higher dimensions, a scarcity of sharp bounds on the complexity of the union itself, to begin with, and then on the complexity of a decomposition of its complement. Often, the best one can do is to decompose the entire arrangement of the given ranges, which results in too many elementary cells, and consequently in an algorithm with poor performance.

To recap, in the key technical step in answering general semialgebraic range reporting or emptiness queries, the best current approaches are either to construct a cutting of the entire arrangement of the range-bounding surfaces in the original space, or to construct a shallow cutting in another higher-dimensional space into which the ranges can be linearized. For many natural problems (including the segment-emptiness problem for balls in \(\mathbb{R}^3\)), both
approaches yield relatively poor performance.

As we will shortly note, in handling general semialgebraic ranges, we face another major technical issue, having to do with the construction of efficient test sets of ranges (in the terminology of [5], elaborated below). Addressing this issue is a major component of the analysis in Chapter 4, and is discussed in detail in that chapter.

**Our results.** We propose a variant of the shallow-cutting machinery of [57] for the case of semialgebraic ranges, which avoids the need for linearization, and works in the original space (which, for the case of ray shooting amid balls, is a 4-dimensional parametric space in which the balls are represented as points). While the machinery used by our variant is similar in principle to that in [57], there are several new significant technical difficulties which require more careful treatment.

Matoušek’s technique [57], as well as ours, considers a finite set \( Q \) of shallow ranges (called a test set), and builds a data structure which caters only for ranges in \( Q \). Matoušek shows how to build, for any given parameter \( r \), a set of half-spaces of size polynomial in \( r \), which represents well all \( (n/r) \)-shallow ranges, in the following sense: For any simplicial partition \( \Pi \) with parameter \( r \), let \( \kappa \) denote the maximal number of cells of \( \Pi \) crossed by a half-space in \( Q \). Then each \( (n/r) \)-shallow half-space crosses at most \( c\kappa \) cells of \( \Pi \), where \( c \) is a constant that depends on the dimension. Unfortunately (for the present analysis), the linear nature of the ranges is crucially needed for the proof, which therefore fails for non-linear ranges.

Being a good representative of all shallow ranges, in the above sense, is only one of the requirements from a good test set \( Q \). The other requirements are that \( Q \) be small, so that, in particular, it can be constructed efficiently, and that the (decomposition of the) complement of the union of any subset of \( Q \) have small complexity. All these properties hold for the case of half-spaces bounded by hyperplanes, studied in [57].

As it turns out, and hinted above, obtaining a “good” test set \( Q \) for general semialgebraic ranges, with the above properties, is not an easy task. We give a simple general recipe for constructing such a set \( Q \), but it consists of more complex ranges than those in the original setup (albeit still of constant description complexity). A major problem with this recipe is that since the members of \( Q \) have a more complex shape, it becomes harder to establish good bounds on the complexity of (the decomposition of) the complement of the union of any subset of these generalized ranges.

Nevertheless, once a good test set has been shown to exist, and to be efficiently computable, it leads to a construction of an efficient elementary-cell partition with a small crossing number for any empty or shallow original range. Using this construction recursively, one obtains a partition tree, of linear size, so that any shallow original range \( \gamma \) visits only a small number of its nodes (where \( \gamma \) visits a node if it crosses the elementary cell enclosing the subset of that node, meaning, as above, that it intersects this cell but does not fully contain it), which in turn leads to an efficient range reporting or emptiness-testing
1.2 Our Results

procedure. This part, of constructing and searching the tree, is almost identical to its counterparts in the earlier works [5, 57, 58], as reviewed above, and we will not elaborate on it in Chapter 4, focusing only on the technicalities in the construction of a single “shallow” elementary-cell partition.

Developing all this machinery, and then putting it into action, we obtain efficient data structures for the following applications, improving previous results or obtaining the first nontrivial solutions. These instances are:

Ray shooting amid balls in 3-space. Given a set $S$ of $n$ balls in $\mathbb{R}^3$, we construct, in $O^*(n)$ time, a data structure of $O(n)$ size, which can determine, for a given query segment $e$, whether $e$ is empty (avoids all balls), in $O^*(n^{2/3})$ time. Plugging this data structure into the parametric searching technique of Agarwal and Matoušek [4], we obtain a data structure for answering ray shooting queries amid the balls of $S$, which has similar performance bounds.

We represent balls in 3-space as points in $\mathbb{R}^4$, where a ball with center $(a, b, c)$ and radius $r$ is mapped to the point $(a, b, c, r)$, and each object $K \subset \mathbb{R}^3$ is mapped to the surface $\sigma_K$, which is the locus of all (points representing) balls tangent to $K$ (i.e., balls that touch $K$, but do not penetrate into its interior). In this case, the range of an object $K$ is the upper half-space $\sigma_K^+$ consisting of all points lying above $\sigma_K$ (representing balls that intersect $K$). The complement of the union of a subfamily of these ranges is the region below the lower envelope of the corresponding surfaces $\sigma_K$. The minimization diagram of this envelope is the 3-dimensional Euclidean Voronoi diagram of the corresponding set of objects. Thus we reveal (what we regard as) a somewhat surprising connection between the problem of ray shooting amid balls and the problem of analyzing the complexity of Euclidean Voronoi diagrams of (simply-shaped) objects in 3-space. In fact this can be generalized to answer emptiness queries amid balls in $\mathbb{R}^3$ for any class of ranges of constant description complexity, with the same bounds. This, for example, leads to a data structure that answers stone throwing queries amid balls in $O^*(n^{2/3})$ query time (and near-linear storage).

Farthest point from a line (or from any convex set) in $\mathbb{R}^3$. Let $P$ be a set of $n$ points in $\mathbb{R}^3$. We wish to preprocess $P$ into a data structure of size $O(n)$, so that, for any query line $\ell$, we can efficiently find the point of $P$ farthest from $\ell$. This is a useful routine for approximating polygonal paths in three dimensions; see [32].

As in the ray shooting problem, we can reduce such a query to a range emptiness query of the form: Given a cylinder $C$, does it contain all the points of $P$? (That is, is the complement of the cylinder empty?) We prefer to regard this as an instance of the complementary range fullness problem, which seeks to determine whether a query range is full (i.e., contains all the input points).

\footnote{In our solution, we will use a test set of objects $K$ which are considerably more complex than just lines or segments, but are nevertheless still of constant description complexity.}
Our machinery can handle this problem. In fact, we can solve the range fullness problem for any family of convex ranges in 3-space, of constant description complexity. Our solution requires $O(n)$ storage and near linear preprocessing, and answers a range fullness query in $O^*(n^{1/2})$ time, improving the query time $O^*(n^{2/3})$ given by Agarwal and Matoušek [5].

We then apply this result to solve the problem of finding the largest-area triangle spanned by a set of $n$ points in 3-space. The resulting algorithm requires $O^*(n^{26/11})$ time, which improves a previous bound of $O^*(n^{13/5})$ due to Daescu and Serfling [32]. We also adapt our machinery to compute efficiently the largest-perimeter triangle and the largest-height triangle spanned by such a point set.

In both this, and the preceding ray-shooting applications, we use the general, more abstract recipe for constructing good test sets.

**Fat triangle and circular cap range emptiness searching and reporting.** Finally, we consider two planar instances of the range emptiness and reporting problems, in which we are given a planar set $P$ of $n$ points, and the ranges are either $\alpha$-fat triangles or sufficiently large circular caps (say, larger than a semidisk). The general technique of Agarwal and Matoušek [5] yields, for any class of planar ranges with constant description complexity, a data structure with near linear preprocessing and linear storage, which answers such queries in time $O^*(n^{1/2})$ (for emptiness) or $O^*(n^{1/2}) + O(k)$ (for reporting). We improve the query time to $O^*(1)$ and $O^*(1) + O(k)$, respectively, in both cases.

In these planar applications, we abandon the general recipe, and construct good test sets in an ad-hoc (and simpler) manner. For $\alpha$-fat triangles (i.e., triangles with the property that each of their angles is at least $\alpha$, which is some fixed positive constant), the test set consists of "canonical" $(\alpha/2)$-fat triangles, and the fast query performance is a consequence of the fact that the complexity of the (complement of the) union of $m$ $\alpha'$-fat triangles is $O^*(m)$; specifically, the best upper bound is $O(m \log^* m)$ [13] (see also [43, 60]). It is quite likely that our machinery can also be applied to other classes of fat objects in the plane, for which near-linear bounds on the complexity of their union are known [34, 39, 40, 41]. However, constructing a good test set for each of these classes is not an obvious step. We leave these extensions as open problems for further research. (For fat triangles, our solution competes with an alternative known solution, with similar performance parameters. See Chapter 4 for more details.)

For circular caps, the motivation for range emptiness searching comes from the problem of finding, for a query consisting of a point $q$ and a line $\ell$, the point of $P$ which lies above $\ell$ and is nearest to $q$ (we only consider the case where $q$ lies on or above $\ell$). Such a procedure was considered in [31]. Using parametric searching, the latter problem can be reduced to that of testing for emptiness of a circular cap centered at $q$ and bounded by $\ell$ (the assumption on the location of $q$ ensures that this cap is at least a semidisk). Here too we manage to construct a test set which consists of (possibly slightly smaller) circular caps, and we exploit the fact that the complexity of the union of $m$ such caps is $O^*(m)$,
as long as the caps are not too small (relative to their bounding circles), to obtain the fast performance stated above.

**Related work.** Our study was originally motivated by work by Daescu and others [31, 32] on path approximations and related problems. In these applications one needs to compute efficiently the vertex of a subpath which is farthest from a given segment (connecting the two endpoints of the subpath). These works used the standard range searching machinery of [5], based on linearization into a high-dimensional space and motivated us to look for faster implementations.

Aronov et al. [14] studied variants of these problems for the case where the points of $P$ are in convex position. They proposed an algorithm that builds a data structure that uses $O(n \log^3 n)$ storage and can efficiently answer queries that seek the nearest point above a line, or the farthest point above a line. The query time is $O(\log n)$. In contrast, we solve the case of nearest point queries for arbitrary point sets.

The results of Chapter 4 appeared in [71, 72].

### 1.2.4 Approximate range counting

Given a set $P$ of $n$ points in $\mathbb{R}^d$, a set $\Gamma$ of semialgebraic ranges of constant description complexity, and a parameter $\delta > 0$, the approximate range counting problem for $(P, \Gamma)$ is to preprocess $P$ into a data structure such that, for any query range $\gamma \in \Gamma$, we can efficiently compute an approximate count $t_\gamma$ which satisfies

$$(1 - \delta)|P \cap \gamma| \leq t_\gamma \leq (1 + \delta)|P \cap \gamma|.$$

As in most of the rest of the thesis, we will only consider the case where the size of the data structure is to be (almost) linear, and the goal is to find solutions with small query time.

The problem has been studied in several recent papers [15, 16, 17, 51], for the special case where $P$ is a set of points in $\mathbb{R}^d$ and $\Gamma$ is the collection of half-spaces (bounded by hyperplanes). A variety of solutions, with near-linear storage, were derived; in all of them, the dependence of the query cost on $n$ is close to $n^{1-1/\lfloor d/2 \rfloor}$, which, as reviewed earlier, is roughly the same as the cost of half-space range emptiness queries, or the overhead cost of half-space range reporting queries [57].

The fact that the approximate range counting problem is closely related to range emptiness comes as no surprise, because, when $P \cap \gamma = \emptyset$, the approximate count $t$ must be 0, so range emptiness is a special case of approximate range counting. The goal is therefore to derive solutions that are comparable, in their dependence on $n$, with those that solve emptiness (or reporting) queries. As just noted, this has been accomplished for the case of half-spaces. In Chapter 5 we extend this technique to the general semialgebraic case, drawing on our results for emptiness and reporting searching, presented in Chapter 4.
Adapting the recent techniques of [15, 16, 17], we can turn our solutions into efficient algorithms for approximate range counting (with small relative error) for the cases mentioned above. That is, for a specified \( \delta > 0 \), we can preprocess the input point set \( P \) into a data structure which can efficiently compute, for any query range \( \gamma \) in the appropriate class \( \Gamma \) of semialgebraic sets, an approximate count \( t_\gamma \), satisfying \( (1 - \delta)|P \cap \gamma| \leq t_\gamma \leq (1 + \delta)|P \cap \gamma| \).

The performance of the resulting algorithms is comparable with those of the corresponding emptiness and reporting algorithms, and is detailed in Chapter 5.

A simple way of turning the algorithms of Chapter 4 into procedures for approximate range counting is to use the algorithm of Aronov and Har-Peled [15], which performs approximate range counting by a binary search over \( |P \cap \gamma| \), where the search is guided by repeated calls to an emptiness testing routine on various random samples of \( P \). This algorithm uses emptiness searching as a black box, so, plugging our solutions for this latter problem into the algorithm in [15], we obtain efficient approximate range counting algorithms for the ranges considered in Chapter 4.

The second approach is to adapt the alternative machinery of Aronov and Sharir [17]. Informally, it uses, for the case of half-spaces, a shallow partition tree, in the style of [57], where each node of the tree is augmented with an additional structure (relative \((p, \epsilon)\)-approximations, to be precise; see [17]). When a query range \( \gamma \) is fed into the tree, it visits some of its nodes. If \( \gamma \) is shallow at a node \( v \), it visits only a small number of its children, and if \( \gamma \) is not shallow at \( v \), we use the auxiliary structure stored at \( v \) to approximate \( |\gamma \cap P_v| \), where \( P_v \) is the subset of points stored at \( v \). Overall, this yields the desired approximate count, at a cost comparable with that of range emptiness queries.

In Chapter 5 we show how to adapt this machinery to general semialgebraic ranges, using the tools developed in Chapter 4. The performance bounds of the resulting structure satisfy similar properties to the bounds derived in [17]. See the chapter for more details.

The first solution in Chapter 5 appeared in [72].
Chapter 2

Ray Shooting and Stone Throwing

“Hey, they’re shooting at us,” said Arthur, crouching in a tight ball, “I thought they said they didn’t want to do that.”

“Yeah, I thought they said that,” agreed Ford.

Zaphod stuck a head up for a dangerous moment.

“Hey,” he said, “I thought you said you didn’t want to shoot us!” and ducked again.

They waited.

After a moment a voice replied, “It isn’t easy being a cop!”
(The Hitchhiker’s Guide to the Galaxy)

2.1 Introduction

In this chapter we study two ray shooting problems. The first problem, studied in Section 2.2, considers shooting straight rays amid special classes of triangles in three dimensions. These include the class of fat triangles and the class of triangles stabbed by a common line. In these and similar cases, our technique requires near-linear preprocessing and storage, and answers a query in $O^*(n^{2/3})$ time. This improves the best known result of $O^*(n^{3/4})$ query time (with near-linear storage) for general triangles.

In the second problem, considered in Section 2.3, we study shooting along certain arcs amid arbitrary triangles in three dimensions. In the main special case that we consider, we are given a set $T$ of $n$ triangles in $\mathbb{R}^3$, and we wish to preprocess them into a data structure that can answer efficiently stone throwing queries, where each query specifies a point $p \in \mathbb{R}^3$ and a velocity vector $v \in \mathbb{R}^3$; these parameters define a parabolic trajectory traced by a stone thrown from $p$ with initial velocity $v$ under gravity (which we assume to be exerted in the negative $z$-direction), and the query asks for the first triangle of $T$ to be
hit by this trajectory. Nevertheless, our technique can also be applied to other classes of arcs; see below for details.

2.2 Ray Shooting amid Fat Triangles and Other Special Classes

2.2.1 Preliminaries

In this section we assume that the given triangles are all non-vertical (i.e., none of them is parallel to the $z$-axis). The case of vertical triangles is considerably simpler, and can be treated using a simplified variant of the method presented below.

A triangle $\Delta$ is $\alpha$-fat (or fat, in short) if all its internal angles are larger than some fixed angle $\alpha$.

A positive curtain (resp., negative curtain) is an unbounded polygon in space with three edges, two of which are parallel to the $z$-axis and extend to $z = +\infty$ (resp., $z = -\infty$). In the extreme case where these vertical edges are missing, the curtain is a vertical half-plane bounded by a single line. Curtains have been studied in [35], but as a class of input objects of its own, rather than as an aid for a general ray shooting problem, as studied here.

Given a segment $s$ in space, we denote by $C^+(s)$ (resp., $C^-(s)$) the positive (resp., negative) curtain that is defined by $s$, i.e., whose bounded edge is $s$.

We say that a point $p$ is above (below) a triangle $\Delta$ if the vertical projection of $p$ on the $xy$-plane lies inside the vertical projection of $\Delta$, and $p$ is above (below) the plane containing $\Delta$.

Claim 2.2.1 Given a non-vertical triangle $\Delta$ with three edges $e_1$, $e_2$ and $e_3$ and a non-vertical segment $pq$, all in $\mathbb{R}^3$, then the segment $pq$ intersects the triangle $\Delta$ if and only if (exactly) one of the following conditions holds:

(i) $pq$ intersects one positive curtain $C^+(e_i)$ and one negative curtain $C^-(e_j)$ of two distinct respective edges $e_i, e_j$ of $\Delta$.

(ii) One of the endpoints $p,q$ is below $\Delta$ and $pq$ intersects one positive curtain $C^+(e_i)$ of $\Delta$.

(iii) One of the endpoints is above $\Delta$ and $pq$ intersects one negative curtain $C^-(e_i)$ of $\Delta$.

(iv) $p$ is above $\Delta$ and $q$ is below $\Delta$ or vice versa.

Proof: Straightforward; see Figure 2.1.

2.2.2 Overview of the algorithm

We first sketch the outline of our algorithm and then describe it in detail.
2.2 Ray Shooting amid Fat Triangles and Other Special Classes

Reduction to segment emptiness. Given a set $T$ of triangles and a ray $\rho$, specified by a point $p$ and a direction $d$, we want to determine the first triangle $t^* \in T$ intersected by $\rho$. We use the parametric search technique, as in Agarwal and Matoušek [4] (see Section 1.1.3), to reduce this problem to the segment emptiness problem, that is, to the problem of determining whether a query segment $pq$, where $q$ is a variable point along $\rho$, intersects any triangle in $T$.

An algorithm that solves this latter segment-emptiness problem proceeds through the following steps.

Partitioning a triangle into semi-canonical triangles. We start by decomposing each triangle into $O(1)$ sub-triangles, where each sub-triangle has two semi-canonical edges. A semi-canonical edge is an edge whose containing line belongs to a family that has three degrees of freedom, i.e., it can be represented by three real parameters (as opposed to four degrees of freedom for arbitrary lines in space). Examples of triangles that can be partitioned into such semi-canonical triangles are fat triangles, triangles stabbed by a common curve of constant description complexity (e.g., a fixed line), and triangles tangent to a common surface of constant description complexity (e.g., a fixed sphere). See below for details.

Discarding triangles whose $xy$-projections do not intersect the $xy$-projection of the query segment. Denote the $xy$-projection of an object $a$ by $\overline{a}$. We project the triangles and the query segment $pq$ on the $xy$-plane, and obtain a compact representation of the set of all triangles whose $xy$-projections are intersected by $\overline{pq}$. This set will be the union of a small number of canonical, preprocessed sets, and we apply the subsequent stages of the algorithm to each such subset. Moreover, we construct these sets in such a way that allows us to know which pair of edges of each triangle $t$ in a canonical set are

---

Figure 2.1: Intersection of a ray or a segment and a triangle in three dimensions. The four situations asserted in Claim 2.2.1 are shown left to right.
intersected by the segment in the projection. At least one of these edges is necessarily semi-canonical; we call it $e^c_t$, and call the other edge $e^r_t$. We also collect in canonical sets triangles whose projections contain one or two endpoints of $pq$. Handling such triangles is somewhat simpler than triangles of the former kind. This is a fairly standard range searching task, and can be answered by constructing a multi-level structure, where each level checks for one of a conjunction of conditions. As mentioned in Section 1.1.2, the total complexity of the storage needed by the structure is comparable with that of its most space-consuming level, and the time needed to answer a query is comparable with the query cost of its costliest level. The storage requirement of this structure in our case is $O^*(n)$, and the query time is $O^*(n^{1/2})$. See [1] for examples of similar structures. In the remainder of this overview we only consider canonical sets of the former kind.

Checking for intersection with curtains. We next need to test whether there exists a triangle $t^*$ in a given canonical set such that the query segment intersects the positive curtain $C^+(e^c_t)$ and the negative curtain $C^-(e^r_t)$. The symmetric case, involving $C^-(e^c_t)$ and $C^+(e^r_t)$, is handled similarly.

We first collect all triangles $t$ in a canonical subset so that $pq$ intersects the positive curtain $C^+(e^c_t)$ erected from the semi-canonical edge $e^c_t$ of $t$. The output is again a union of a small number of canonical sets. The fact that the edges $e^c_t$ are semi-canonical allows us to represent these curtains as points in a 3-dimensional space (rather than 4-dimensional as in the case of general curtains or lines in space), and this property is crucial for obtaining the improved query time.

Finally, for each of the new canonical sets, we test whether the segment intersects the negative curtain $C^-(e^r_t)$, erected over the other edge $e^r_t$ of at least one triangle in the set. This step is done using the (standard) representation of lines in $\mathbb{R}^3$ as points or hyperplanes in the 5-dimensional Plücker space, and exploiting the linear structure of this representation [24]. Symmetrically, we test, over all canonical sets, whether $pq$ intersects the negative curtain $C^-(e^c_t)$ and the positive curtain $C^+(e^r_t)$ of at least one triangle $t$. Any successful test at this stage implies that $pq$ intersects a triangle in $\mathcal{T}$. We also test for intersections of the types $(ii)-(iv)$ listed in Claim 2.2.1 using similar techniques (described in more detail below).

As our analysis will show, this multi-level structure uses $O^*(n)$ space and preprocessing and can answer queries in $O^*(n^{2/3})$ time.

2.2.3 Partitioning a triangle into semi-canonical triangles

Fat Triangles

Assume that the triangles are not “too vertical”. This assumption is needed to assure that a fat triangle in 3-space has a fat projection on the $xy$-plane. More formally, we assume that the angle formed between the $xy$-plane and the plane that supports any triangle in $\mathcal{T}$ is
at most $\theta$, where $\cos \theta = \frac{1}{\sqrt{3}}$. Steeper triangles can (and will) be handled by an appropriate permutation of the coordinate axes. (Every triangle has this property with respect to at least one coordinate plane.)

It is easy to see that there exists a set of vertical planes $D$ of size $O(1/\alpha)$, such that, for each vertex $v$ of any $\alpha$-fat triangle $t$ which is not too vertical, it is possible to split $t$ into two (non-empty) triangles by a segment incident to $v$ which is parallel to some plane in $D$. We refer to such a segment as being semi-canonical.

Given a set $T$ of $\alpha$-fat, not-too-vertical triangles, we decompose each triangle $\Delta \in T$ into four triangles, such that each new triangle has at least two semi-canonical edges. This is easy to do, in the manner illustrated in Figure 2.2; a similar construction, for horizontal fat triangles, has been used by de Berg et al. [35]. We refer to the resulting sub-triangles as semi-canonical.

![Figure 2.2: Partitioning a triangle into four semi-canonical triangles.](image)

We partition $T$ into $O(1/\alpha^2)$ canonical families, where all triangles in the same family have two edges parallel to two fixed canonical planes. We preprocess each family separately. Let $F$ be a fixed canonical family. Let us assume, without loss of generality, that the two corresponding canonical planes are the $xz$-plane and the $yz$-plane. (Since our problem is affine-invariant, we can achieve this reduction by an appropriate affine transformation).

The negative curtain $C^-(e^t_c)$ extended from (the line containing) a semi-canonical edge, $e^t_c$ parallel to the $xz$-plane has three degrees of freedom, and can be represented as $C_{\zeta,\eta,\xi} = \{(x, y, z) \mid y = \xi, z \leq \zeta x + \eta\}$, for three real parameters $\zeta, \eta, \xi$. The query line $\ell$ that contains the query segment $pq$ can be represented as $\ell_{a, b, c, d} = \{(x, y, z) \mid z = ay + b, x = cy + d\}$, for four real parameters $a, b, c, d$. We represent (the line bounding) a negative curtain $C_{\zeta,\eta,\xi}$ as the point $(\zeta, \eta, \xi)$ in a 3-dimensional parametric space $\Pi$. A query line $\ell_{a, b, c, d}$ intersects the negative curtain $C_{\zeta,\eta,\xi}$ if and only if $a\xi + b \leq \zeta(c\xi + d) + \eta$, which defines a pseudo-half-space in $\Pi$ bounded by the hyperbolic paraboloid $\eta = -c\xi + a\xi - d\zeta + b$. If we regard $\eta$ as the third, vertical coordinate axis in $\Pi$, then the point $(\zeta, \eta, \xi)$ representing $C^-(e^t_c)$ lies above the paraboloid if and only if the line $\ell_{a, b, c, d}$ intersects $C^-(e^t_c)$. In other words, we transform the problem of computing the set of negative curtains that are intersected by a query line to the problem of computing the subset of the curtain-representing points that lie in the pseudo-half-space above a query hyperbolic paraboloid.
Triangles Stabbed by a Common Line

Observe that a triangle intersecting a fixed line $\ell$ can be covered by three triangles, each having two edges with an endpoint on $\ell$, as illustrated in Figure 2.3.

Figure 2.3: Partitioning a triangle stabbed by a line $\ell$ into three semi-canonical triangles.

Assume, without loss of generality, that the triangles are stabbed by the $z$-axis. In this case, the representation of a semi-canonical negative curtain becomes: $C_{\zeta,\eta,\xi} = \{(x, y, z) \mid z \leq \zeta x + \eta, y = \xi x\}$. Again, a curtain defined by such an edge can be represented as a point $(\zeta, \eta, \xi)$ in a 3-dimensional parametric space $\Pi$. A query line $\ell_{a,b,c,d} = \{(x, y, z) \mid z = ax + b, y = cx + d\}$ intersects $C_{\zeta,\eta,\xi}$ if and only if $(\eta \xi + d \zeta + bc - da - b \xi - \eta c)(\xi - c) \geq 0$, as is easily checked. The first factor is the equation of a pseudo-half-space bounded by a hyperbolic paraboloid in $\Pi$. Here too, we have thus transformed the problem of computing the set of negative curtains that are intersected by a query line to the problem of computing the subset of curtain-representing points that lie in a semialgebraic portion of 3-space that is bounded by a hyperbolic paraboloid and a vertical plane.

Other Extensions

As stated earlier, this algorithm can be extended to any set of triangles, each having two edges that can each be described by three parameters (or where each triangle can be covered by such triangles). For example, triangles stabbed by an algebraic curve of constant degree, triangles tangent to an algebraic surface of constant degree, etc. In all these cases, our analysis can be extended to yield an algorithm that requires $O^*(n)$ storage and preprocessing, and answers ray shooting queries in $O^*(n^{2/3})$ time.

2.2.4 Finding intersections with semi-canonical curtains

Let $T'$ be one of the canonical sets output in the previous stage, consisting of triangles with the property that $pq$ crosses two of their edges. For each $t \in T'$, let $e^t_c, e^t_r$ denote the two edges of $t$ whose $xy$-projections are crossed by $pq$, where $e^t_c$ is semi-canonical (each triangle in the canonical set is associated with such a pair of pre-computed edges). In the next stage, we preprocess $T'$ into additional levels of the data structure, which allow us to compute the subset of all those triangles for which $pq$ intersects $C^{-}(e^t_c)$. 
2.2 Ray Shooting amid Fat Triangles and Other Special Classes 27

We choose some parameter \( r \leq n \), and obtain a partition of the (set of points in \( \Pi \) representing the) semi-canonical lines into \( m = O(r) \) subsets \( L_1, \ldots, L_m \), such that each set consists of \( O(n/r) \) points, and the following property holds. For each query line \( \ell \), let \( \sigma_\ell \) denote the surface bounding the parametric region \( \sigma_\ell^+ \) of all negative semi-canonical curtain crossed by \( \ell \). Then \( \sigma_\ell \) separates the points of at most \( O^*(r^{2/3}) \) sets. This partitioning follows, in a routine manner, from the technique of Agarwal and Matoušek [5] for range searching with semialgebraic sets, which, in the special case at hand, uses the fact that the arrangement of any \( t \) of these surfaces can be decomposed into \( O^*(t^3) \) subcells, each of constant description complexity.

Given a query surface that represents the query line \( \ell \), every set \( L_i \) either lies entirely in the “crossing region” \( \sigma_i^+ \), lies entirely outside \( \sigma_i^+ \), or is separated by \( \sigma_\ell \). If \( L_i \cap \sigma_\ell^+ = \emptyset \), we ignore \( L_i \). If \( L_i \subset \sigma_\ell^+ \), we pass \( L_i \) to the next level of the data structure. Otherwise we recurse into \( L_i \).

Using this structure, we output the set of all triangles that have a negative semi-canonical curtain intersected by \( pq \) (and have another edge whose projection is also intersected by \( pq \)), as the union of pre-computed sets of triangles. Similarly, we can output, in the same fashion, the set of all triangles that have a positive semi-canonical curtain intersected by \( pq \).

Consider next the case where \( T' \) is a canonical set of triangles with the property that their \( xy \)-projections contain one endpoint of \( pq \), say \( p \), and have an edge whose projection intersects \( pq \). Rather than preprocessing curtain-representing points in \( \Pi \), we preprocess the set of planes in \( \mathbb{R}^3 \) that contain the triangles of \( T' \), so that we can efficiently represent the set of all triangles in \( T' \) whose planes pass above a query point \( p \) (or below such a point). To do so, we dualize 3-space, mapping these planes into points, and any query point into a query plane, where the task is to determine the set of all the dual points that lie above the dual query plane. This is a standard three-dimensional range-query task [58], which can be done in much the same way as above, using query planes instead of query surfaces, with the same asymptotic performance bounds.

Finally, consider the simplest case of all, where \( T' \) is a canonical set of triangles with the property that their \( xy \)-projections contain both \( p \) and \( q \). We preprocess \( T' \) exactly as in the preceding paragraph, and query it with the point \( p \). Let \( T'' \) be a canonical subset of the query output, consisting of triangles whose planes pass above \( p \). We then need to determine whether \( T'' \) contains a triangle whose plane passes below \( q \), and then report an intersection of the type \((iv)\) listed in Claim 2.2.1. This final task is easy to accomplish: we compute the lower envelope of the planes of the triangles in \( T'' \), which is a convex polyhedron, and preprocess it for efficient point location queries, which determine whether a query point \( q \) lies above or below the envelope. Note that \( q \) lies above the envelope if and only if it passes above a plane containing a triangle of \( T'' \). Putting \( n'' = |T''| \), this final task can be performed in \( O(n'' \log n'') \) preprocessing time, \( O(n'') \) storage, and \( O(\log n'') \) query time, see [67] for a survey.
2.2.5 Determining an intersection with ordinary curtains

We now describe the last level of our data structure, for the remaining types of canonical subsets, excluding those that correspond to case (iv) of Claim 2.2.1. The output of a query segment $pq$, as it passes through the preceding levels, is a collection of canonical precomputed sets of triangles. Each such subset $T''$ has the property that $pq$ intersects $C^-(e^t)$, for every triangle $t \in T''$, or that $p$ lies below every triangle $t \in T''$; moreover, the projection of $pq$ onto the $xy$-plane intersects the projection of a second, not necessarily semi-canonical, edge $e^t_r$ of $t$. It is therefore sufficient to check whether $pq$ intersects any of the corresponding positive curtains $C^+(e^t_r)$.

This task is equivalent to testing whether there exists a line in a given set of lines in 3-space (namely, the extensions of the edges $e^t_r$) which passes below a query line (which is the line containing $pq$). Equivalently, it suffices to test whether all the input lines pass above the query line.

This task can be accomplished by mapping the data lines to points, and the query line to a hyperplane, in Plücker’s 5-space [24], and by applying Matoušek’s algorithm for half-space emptiness detection [57]. This algorithm, which is based on the so-called shallow cutting lemma, runs in $O(n^{1/2}2^{O(\log^* n)})$ time, with linear space and $O(n \log n)$ preprocessing time.

The overall cost of the algorithm. The space requirement $\Sigma(n)$ of any level in our data structure (including all the subsequent levels below it), for a set of $n$ triangles, satisfies the recurrence:

$$\Sigma(n) = O(r)\Sigma'(\frac{n}{r}) + O(r)\Sigma(\frac{n}{r}),$$

where $\Sigma'(n)$ is the space requirement of the next levels, for a set of $n$ triangles. If $\Sigma'(n) = O(n^{1+\varepsilon})$, for any $\varepsilon > 0$, then, choosing $r$ to be a sufficiently large constant that depends on $\varepsilon$, one can show that $\Sigma(n) = O(n^{1+\varepsilon})$, for any $\varepsilon > 0$, as well. Since $\Sigma'(n) = O(n)$ at the bottom level of the structure, this implies that the overall storage required by the data structure is $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$. The preprocessing time obeys a similar recurrence whose solution is also $O(n^{1+\varepsilon})$. That is, both storage and preprocessing are $O^*(n)$.

Similarly, the query time $Q(n)$ of any level in our data structure, for a set of $n$ triangles, satisfies the recurrence:

$$Q(n) = O(r) + O(r)Q'(\frac{n}{r}) + O^*(r^{2/3})Q(\frac{n}{r}),$$

where $Q'(n)$ is the query time at the next levels (for $n$ triangles). If $Q'(n) = O(n^{2/3+\varepsilon})$ for any $\varepsilon > 0$ (it is $O(1)$ at the bottom level of the structure), then, choosing $r$ as above, and calibrating the factor $O^*(r^{2/3})$, it follows that $Q(n) = O(n^{2/3+\varepsilon})$, for any $\varepsilon > 0$, as well. That is, $Q(n) = O^*(n^{2/3})$. In conclusion, we thus obtain:
Theorem 2.2.2 The ray shooting problem amid $n$ fat triangles in $\mathbb{R}^3$ can be solved with a query time $O^*(n^{2/3})$, using a data structure of size $O^*(n)$, which can be constructed in $O^*(n)$ time. The same bounds apply to each of the other special classes of triangles listed above, such as triangles stabbed by a common line.

2.2.6 Trading space for query time

So far, we have only considered algorithms that use near-linear storage. At the other end of the spectrum, using $O^*(n^4)$ space and preprocessing, ray shooting queries for general triangles can be answered in logarithmic time [9, 35]; there are no known better bounds for the special cases that we discuss. By combining these two solutions, a trade-off between storage and query time can be obtained, as in [9, 27, 59]. The idea behind these techniques is to build a partition-tree-like data structure, as described in the preceding sections, up to a certain depth. At that depth we apply the more space-consuming construction to each of the subsets of triangles stored at the (new) leaves. For example, if we recurse up to a depth where there are $O(n^\alpha)$ triangles in each leaf of the partition tree, then we need $O^*(n^{1-\alpha}n^{4\alpha}) = O^*(n^{1+3\alpha})$ space, and the query time is $O^*((n^{1-\alpha})^{2/3} \log n) = O^*(n^{2/3-2/3\alpha})$. This two-way construction has to be applied to each level of our data structure, but, as shown in [9, 27, 59], this does not affect the asymptotic form of the overall trade-off just noted. Putting $m = n^{1+3\alpha}$, or $n^\alpha = (m/n)^{1/3}$, the storage becomes $O^*(m)$ and the query becomes $O^*(n^{2/3}(m/n)^{-2/9}) = O^*(n^{8/9}/m^{2/9})$. In summary, we have shown:

**Theorem 2.2.3** For any parameter $n \leq m \leq n^4$, the ray shooting problem for a set of $n$ fat triangles, or $n$ triangles stabbed by a common line, or any of the other special cases of Section 2.2.3, can be solved using $O^*(m)$ space and preprocessing, and $O^*(n^{8/9}/m^{2/9})$ query time.

2.3 Stone Throwing

*The ships hung in the sky in much the same way that bricks don't.*

(The Hitchhiker's Guide to the Galaxy)

Next we study another problem of shooting along arcs amid triangles in three dimensions, which we refer to as **stone throwing**. In this problem we are given a set $\mathcal{T}$ of $n$ (arbitrary) triangles in $\mathbb{R}^3$, and we wish to preprocess them into a data structure that can answer efficiently stone throwing queries, where each query specifies a point $p \in \mathbb{R}^3$ and a velocity vector $v \in \mathbb{R}^3$; these parameters define a parabolic trajectory traced by a stone thrown from $p$ with initial velocity $v$ under gravity (which we assume to be exerted in the negative $z$-direction), and the query asks for the first triangle of $\mathcal{T}$ to be hit by this trajectory.
The query has six degrees of freedom, but the parabola \( \pi \) that contains the stone trajectory has only five degrees of freedom. Two of them define the vertical plane \( V_\pi \) (parallel to the \( z \)-axis) in which \( \pi \) lies, and three define the planar equation of \( \pi \) within \( V_\pi \). That is, we can specify the parabola, e.g., by the quintuple \((a, b, c, d, e)\) that define the equations \( y = ax + b, \ z = cx^2 + dx + e \). (We ignore, for simplicity of presentation, the simpler case where \( V_\pi \) is parallel to the \( yz \)-plane.) Note that, under gravity, we have \( c < 0 \), i.e., \( \pi \) is concave.

Unlike our treatment of the case of ray shooting, we only consider here the basic case where the triangles of \( T \) are arbitrary, and present a solution that uses near-linear storage and preprocessing, and answers stone-throwing queries in time \( O^*(n^{3/4}) \). These bounds are interesting, since they are identical to the best bounds known for the general ray-shooting problem, even though the stone-throwing problem seems to be harder than ray shooting, since it involves one additional degree of freedom. At present we do not know whether the problem admits a faster solution for the special classes of triangles considered in the first part of the chapter. Moreover, at the other extreme end of the trade-off, where we wish to answer stone-throwing queries in \( O(\log n) \) time, the best solution that we have requires \( O^*(n^5) \) storage, which is larger, by a factor of \( n \), than the best known solution for the ray-shooting problem. We omit in this chapter the description of this alternative solution.

Using the parametric searching technique, as in [4] and the preceding section, we can reduce the problem to testing emptiness of concave vertical parabolic arcs, in which we wish to determine whether such a query arc intersects any triangle in \( T \).

**Claim 2.3.1** Given a non-vertical triangle \( \Delta \), contained in a plane \( h \), with three edges \( e_1, e_2 \) and \( e_3 \), and given a parabolic arc \( \tilde{pq} \) contained in some concave vertical parabola \( \pi \), and delimited by the points \( p, q, \) all in \( \mathbb{R}^3 \), then the arc \( \tilde{pq} \) intersects the triangle \( \Delta \) if and only if (exactly) one of the following conditions holds (in the first four conditions the intersection consists of a single point, and in the last three it consists of two points):

1. \( \tilde{pq} \) intersects one positive curtain \( C^+(e_i) \) and one negative curtain \( C^-(e_j) \) of \( \Delta \).
2. One endpoint, say \( p \), lies below \( \Delta \), and \( \tilde{pq} \) intersects one positive curtain \( C^+(e_i) \) of \( \Delta \).
3. One endpoint, say \( p \), lies above \( \Delta \), and \( \tilde{pq} \) intersects one negative curtain \( C^-(e_i) \) of \( \Delta \).
4. One endpoint lies above \( \Delta \), and the other endpoint lies below \( \Delta \).
5. The parabola \( \pi \) intersects the plane \( h \), \( \tilde{pq} \) intersects two negative curtains \( C^-(e_i) \) and \( C^-(e_j) \), at the respective intersection points \( p_1 \) and \( p_2 \), and \( S(p_1) \leq \text{slope}(h \cap V_\pi) \leq S(p_2) \) (or vice versa), where \( S(x) \) is the slope of the tangent to \( \pi \) at point \( x \), and \( \text{slope}(h \cap V_\pi) \) is the slope of this intersection line, where all slopes are defined within the vertical plane \( V_\pi \).
6. One endpoint, say \( p \), lies below \( \Delta \), \( \pi \) intersects the plane \( h \), \( \tilde{pq} \) intersects one negative curtain \( C^-(e_i) \) of \( \Delta \) at some point \( p_1 \), and \( S(p_1) \leq \text{slope}(h \cap V_\pi) \leq S(p) \), or \( S(p) \leq \text{slope}(h \cap V_\pi) \leq S(p_1) \).
(2.iii) The parabola \( \pi \) intersects the plane \( h \), \( p \) and \( q \) lie below \( \Delta \), and \( S(p) \leq \text{slope}(h \cap V_\pi) \leq S(q) \) (or vice versa).

Proof: Straightforward. The first four conditions are similar to the ones given in Claim 2.2.1. The fifth condition is depicted in Figure 2.4, and the last two conditions are similar to it.

![Figure 2.4: Intersection of a vertical concave parabolic arc and a triangle: Case (2.i) of Claim 2.3.1. (The condition on the slopes is implicit in the figure.)](image)

As in the case of ray shooting, we can use Claim 2.3.1 to design an algorithm that solves the parabolic arc emptiness problem by testing whether any of the conditions (1.i)–(2.iii) holds. In the interest of brevity, we only describe how to check for conditions (1.i) and (2.i). The other conditions can be handled in a similar (and somewhat simpler) manner.

### 2.3.1 Testing condition (1.i)

Let \( \gamma = \tilde{pq} \) be a concave vertical parabolic arc. The general approach is similar to that used in Section 2.2 for the segment emptiness problem, and uses a multi-level data structure. We use the top levels to determine the triangles of \( T \) whose \( xy \)-projections are crossed by the \( xy \)-projection \( \gamma \) of \( \gamma \), which is a line segment, and such that both endpoints of \( \gamma \) lie outside the projected triangle. This is done exactly as before. Let \( T' \) be a canonical set of triangles that is output for \( \gamma \) at these top levels, where, in addition, each triangle \( t \in T' \) is associated with two of its edges \( e_t^1, e_t^2 \) whose \( xy \)-projections are crossed by the projection of \( \gamma \). As above, we can replace these edges by the straight lines that contain them, and the query arc \( \gamma \) by the entire parabola \( \pi \) that contains it, and we need to determine whether \( \pi \) crosses the negative curtain of \( e_t^1 \) and the positive curtain of \( e_t^2 \), for some \( t \in T' \) (or vice versa).

Both steps are handled in a similar manner. Consider, for example, the case of the negative curtains of the edges \( e_t^1 \). We map the lines supporting these edges to points in an appropriate 4-dimensional parametric space, and denote by \( T^* \) the resulting set of points. We map each concave vertical parabola \( \pi \) into the surface \( \sigma_\pi \) consisting of all points that represent lines that touch \( \pi \). We apply to \( T^* \) the partitioning technique of Agarwal and
Matoušek [5], which partitions $T^*$ into $r$ subsets, each consisting of $O(n/r)$ points, so that any surface $\sigma_\pi$ separates the points in at most $O^*(r^{3/4})$ subsets. This result depends on the existence of a vertical decomposition of the four-dimensional arrangement of any $m$ surfaces $\sigma_\pi$ into $O^*(m^4)$ elementary cells (see [5] and Chapter 4 for details), which follows from the result of Koltun [53]. We apply this partitioning step recursively, and obtain a partition tree, similar to those constructed earlier. Similar structures are constructed at the next level, which handles positive curtains erected over the other edges $e'_2$ of triangles in each resulting canonical subset. Recall that at this stage we only need to determine whether there exists a triangle $t$ in the canonical subset $T$ whose edge $e'_t$ passes below $\pi$. With an appropriate parametrization this can be expressed as testing whether any point in $T^*$ lies below the surface $\sigma_\pi$. This test is simpler than the preceding one, but we make no attempt to optimize it, and instead use the same partition tree as in the preceding level. (See however Chapter 4 in which problems of this latter kind are studied in detail, and where improved solutions are derived in several special instances.) We omit the further straightforward details. Arguing as above, one can show that the overall data structure requires $O^*(n)$ storage and preprocessing, and tests for condition (1.i) in overall time $O^*(n^{3/4})$.

2.3.2 Testing condition (2.i)

This test is somewhat more involved than the preceding one. It also constructs a multi-level data structure, whose top levels are similar to the entire structure of the preceding data structure. Using them, we can collect, for a query arc $\gamma$, canonical subsets of triangles, so that, for each triangle $t$ in each such set $T'$, $\gamma$ intersects the negative curtains erected over two designated edges $e'_t, e'_2$ of $t$.

In the next level, we collect those triangles $t$ that also satisfy the condition that the parabola $\pi$ containing $\gamma$ intersects the plane containing $t$. This is done by mapping the planes containing the triangles $t \in T'$ to points in dual 3-space, and by mapping any query parabola $\pi$ to the surface $\sigma_\pi$, consisting of all points that are dual to planes tangent to $\pi$. With an appropriate parametrization, the query seeks all the dual points which lie below the query surface $\sigma_\pi$. Applying the range-searching technique of [5], we can output the subset of the desired triangles as the union of a collection of canonical sets, as above. Since this level deals with points in 3-space, its performance is better than that of the preceding curtain-intersection levels.

It remains to test, for each such output set $T'$, whether there exists $t \in T'$, such that the two slope conditions, over $C^-(e'_1)$ and $C^-(e'_2)$, are satisfied. The next levels test for the slope conditions. Consider the slope condition over $C^-(e'_1)$, for a triangle $t$ in a canonical subset $T''$ (that is, the condition $S(p_1) \leq \text{slope}(h \cap V_\pi)$). There are two slopes that need to be compared. The first is the slope $S(p_1)$ of the tangent to $\pi$ at the point $p_1$ where it crosses $C^-(e'_1)$. This slope depends only on two of the parameters that specify $t$, namely the coefficients of the equation of the $xy$-projection of $e'_1$. The second slope is that of the
2.3 Stone Throwing

line \( h \cap V \), which depends only on the equation of the plane \( h \) containing \( t \). Moreover, if the equation of this plane is \( z = \xi x + \eta y + \zeta \), then the slope is independent of \( \zeta \). In other words, the overall slope condition can be expressed as a semialgebraic condition that depends on only four parameters that specify \( t \). Hence, we can represent the triangles of \( T'' \) as points in an appropriate 4-dimensional parametric space \(^1\), and map each parabola \( \pi \) into a semialgebraic set of constant description complexity in that space, which represents all triangles \( t \) for which \( \pi \) satisfies the slope condition over \( C^-(e_1^t) \). We now apply the partitioning technique of [5] for the set of points representing the triangles of \( T'' \) and for the set of ranges corresponding to parabolas \( \pi \), as just defined. (Again, since we are in 4-space, the bound of [53] allows us to apply the technique of [5], with the same performance bounds as in the testing of condition (1.1).) The slope condition over the other negative curtains \( C^-(e_2^t) \) is handled in the next and final level of the data structure, in exactly the same way as just described. We omit the further technical but routine details of handling these levels, which are similar to those already discussed.

Since each level of the data structure deals with sets of points in some parametric space of dimension at most four, the preceding analysis implies that the overall query time is \( O^*(n^{3/4}) \), and the storage remains \( O^*(n) \).

Extending these techniques to the other remaining five conditions of Claim 2.3.1, and omitting all further (but routine) details, we thus obtain:

**Theorem 2.3.2** A set of \( n \) triangles in \( \mathbb{R}^3 \) can be preprocessed into a data structure of size \( O^*(n) \) in time \( O^*(n) \), so that any stone-throwing query can be answered in time \( O^*(n^{3/4}) \).

### 2.3.3 Shooting along bounded-degree algebraic arcs

As mentioned in Section 1.2.1 this result can be extended to shooting along arcs that are graphs of univariate algebraic functions of constant maximum degree that lie in any vertical plane. We simply break such a graph into its (constant number of) maximal convex and concave portions, at points where the second derivative vanishes. It is easily verified that Claim 2.3.1 holds for any vertical concave arc and its symmetric counterpart holds for any vertical convex arc. We can therefore apply a similar algorithm to detect whether a concave or convex vertical arc intersects any triangle, and apply it to each of the \( O(1) \) concave and convex portions of the query arc. This is done in essentially the same manner as in the stone throwing algorithm, and we omit the routine details. Note that the algebraic degree of the query arc does not affect the asymptotic performance bounds (but affects the constants of proportionality). This is because the query arc is mapped into a surface in some parametric space of dimension that depends only on the input triangles, and is therefore at most four, as above. We thus obtain:

\(^1\)With an appropriate general position assumption, all these points will be distinct, although coincident points can also be handled.
Theorem 2.3.3 A set of \( n \) triangles in \( \mathbb{R}^3 \) can be preprocessed into a data structure of size \( O^*(n) \) in time \( O^*(n) \), so that shooting along any vertical algebraic arc of constant maximum degree can be answered in time \( O^*(n^{3/4}) \).

2.4 Conclusions

We have presented algorithms and data structures for answering various ray shooting (and stone throwing) queries. All these structures require \( O^*(n) \) storage and preprocessing time, and answer a query in sub-linear time. Our results either improve upon previous bounds, or give the first published bounds for the problem. Our data structures are fairly standard multi-level range searching structures, which make use of the partition technique of Matoušek. In the most significant levels, the data is transformed into parametric space where every line (extension of some triangle edge) is transformed into a point, and the query line or curve (the extension of the query ray or arc) is transformed into a semialgebraic range in that space. The query (at that particular level) then becomes a semialgebraic range searching query, and is carried out using known techniques, such as that of [5].

The innovation of these structures comes from an observation that reduces the dimension of the parametric space used at the critical level(s) of the structure. In the case of ray shooting amid special classes of triangles, this space is three-dimensional (as opposed to the standard four dimensions), which results in \( O^*(n^{2/3}) \) query time. In the case of stone throwing amid general triangles, this space is four-dimensional, which results in \( O^*(n^{3/4}) \) query time.

We did not manage to extend these ideas to improve the storage (and preprocessing time) of data structures that answer queries in logarithmic time. In this other end of the spectrum, the standard approach is to represent a query ray (or its extension to a line) as a point in some parametric space. The triangle edges (or the lines that support them) are transformed to (surfaces bounding) certain semialgebraic ranges in that parametric space. The data structure then answers a query by locating the cell of the arrangement of these surfaces that contains the query point. The fact that the surfaces can be represented by only three parameters does not seem to help here—we still need to consider arrangements in the 4-dimensional parametric space of lines in space (since the query line is not restricted, and can be any line in space). It is an intriguing open question whether the special structure of the arrangement can be exploited to reduce the storage cost.
Chapter 3

Inter-point Visibility over Terrains in Two and Three Dimensions

... wrap it round your head to ward off noxious fumes or to avoid the gaze of the Ravenous Bugblatter Beast of Traal (a mindbogglingly stupid animal, it assumes that if you can’t see it, it can’t see you — daft as a bush, but very ravenous).
(The Hitchhiker’s Guide to the Galaxy)

3.1 Introduction

In this chapter we consider the following mutual visibility problem. We are given a set $P$ of $m$ points and a polygonal or polyhedral terrain $T$ in two or three dimensions with $n$ vertices, so that all the points of $P$ lie above $T$. We present efficient algorithms that determine whether all pairs of points of $P$ are mutually visible, i.e., whether each pair $p, q \in P$ has the property that the segment $pq$ lies fully above $T$.

In Section 3.2 we present an algorithm for the planar case, which runs in $O(m \log m + n)$ time. In Section 3.3 we present an algorithm for the problem in three dimensions. We show that in this case the problem is 3SUM-hard, so our algorithm, which runs in $O(nm \log^2 m)$ time, is likely to be nearly worst-case optimal.

3.2 Terrains in Two Dimensions

Consider first the planar case. Here the terrain $T$ is simply an unbounded connected $x$-monotone polygonal path with $n$ vertices, and $P$ is a set of $m$ points, all lying above $T$. As above, two points $p, q$ are mutually visible if the segment $pq$ does not pass below any terrain vertex. A terrain vertex is a witness to the non-visibility of a pair $p, q \in P$ if it lies above the segment $pq$ (with one point lying to its right and one lying to its left).
Again, the problem is to determine whether all pairs of points of \( P \) are mutually visible, or equivalently, whether no vertex of \( T \) blocks any pair.

To solve this problem efficiently, we note that every pair of points of \( P \) are mutually visible if and only if the convex hull of \( P \) lies completely above \( T \). To test for this property, we compute the convex hull \( C \), in \( O(m \log m) \) time, and then sweep the plane from left to right and test, for each vertex \( q \) of \( T \) whether \( q \) lies above the current edge of the lower hull of \( P \); See Figure 3.1.

![Figure 3.1: The vertex \( q \) is a witness to the non-visibility of \( p_i \) and \( p_j \).](image)

Since the vertices of \( T \) are already given in \( x \)-sorted order, the total running time of the procedure is \( O(m \log m + n) \). That is, we have shown:

**Theorem 3.2.1** Given a polygonal terrain \( T \) in the plane with \( n \) vertices and a set \( P \) of \( m \) points, all lying above \( T \), we can determine, in \( O(m \log m + n) \) time, whether all pairs of points of \( P \) are mutually visible with respect to \( T \).

### 3.3 Terrains in Three Dimensions

In three dimensions our mutual visibility problem can be shown to be 3SUM-hard [44]. We show a reduction directly from the 3SUM problem itself, where we are given three sets of real numbers \( A, B, \) and \( C \), and we wish to determine whether there exist numbers \( a \in A, b \in B, \) and \( c \in C \), such that \( a + b = c \). Given such an instance to the 3SUM problem, we reduce it to the following instance of the 3-dimensional visibility problem. We map each \( a \in A \) to the data point \( a^* = (2a, -1, 0) \), and map each \( b \in B \) to the data point \( b^* = (2b, 1, 0) \). This produces an input set \( P \) of \(|A| + |B| \) points. Next, for each \( c \in C \), we create a sufficiently narrow peak of the terrain \( T \), whose apex is at \( c^* = (c, 0, 1) \), whose base lies on \( z = -1 \), and whose cross-section with the \( xy \)-plane is of diameter at most \( \varepsilon \), where \( \varepsilon \) is chosen sufficiently small, so that, for any \( a \in A, b \in B \), the segment \( a^*b^* \) meets the cross-section if and only if \( c = a + b \). The rest of the terrain lies below the \( xy \)-plane and is triangulated arbitrarily. Clearly, the complexity of \( T \) is \( O(|C|) \), and the entire construction takes nearly linear time.
By construction, \( a \in A, b \in B \) and \( c \in C \) satisfy \( a + b = c \) if and only if \( a^* \) and \( b^* \) are non-visible with respect to \( T \), with the peak at \( c^* \) blocking their mutual visibility. Hence a subquadratic solution to the 3-dimensional mutual visibility problem would yield a subquadratic solution to the 3SUM problem, so the problem is indeed 3SUM-hard, as claimed.

We therefore cannot hope for a subquadratic solution to the 3-dimensional mutual visibility problem. In what follows we present a solution with nearly quadratic running time.

The algorithm checks, for each edge \( e \) of \( T \), whether \( e \) blocks the visibility of any pair of points of \( P \). Consider then a fixed edge \( e \). Project \( P \) and \( e \) on the \( xy \)-plane, and assume, without loss of generality, that the image \( e^* \) of \( e \) lies on the \( y \)-axis; denote its endpoints by \( a \) and \( b \), so that \( a \) is its bottom endpoint and \( b \) is its top endpoint. Let \( L \) (resp., \( R \)) denote the set of points of \( P \) whose projections lie to the left (resp., to the right) of the \( y \)-axis. Denote by \( q^* \) the \( xy \)-projection of a point \( q \in \mathbb{R}^3 \), and let \( L^* \) and \( R^* \) denote the respective projections of \( L \) and \( R \) on the \( xy \)-plane, that is, \( L^* = \{ q^* \mid q \in L \} \) and \( R^* = \{ q^* \mid q \in R \} \). Let \( L_0 \subseteq L \) and \( R_0 \subseteq R \) be subsets with the property that for every \( p \in L_0 \), \( q \in R_0 \), the segment \( p^*q^* \) crosses \( e^* \). We can then test whether \( e \) blocks the visibility of any pair in \( L_0 \times R_0 \) as follows. Let \( \ell \) be the line containing \( e \) (back in 3-space) and let \( p_0 \in L_0 \) be the lowest point of \( L_0 \) as “seen” from \( \ell \). That is, the plane through \( \ell \) and \( p_0 \) passes below all the other points of \( L_0 \). Similarly, let \( q_0 \in R_0 \) be the lowest point of \( R_0 \) as seen from \( \ell \). Then, as is easily checked, \( e \) does not block the visibility of any pair in \( L_0 \times R_0 \) if and only if it does not block the visibility between \( p_0 \) and \( q_0 \). Hence, testing for this property can be done in \( O(|L_0| + |R_0|) \) time.

The algorithm then considers the graph \( H \subseteq L \times R \) whose edges connect those pairs \((p, q)\) for which \( p^*q^* \) intersects \( e^* \), decomposes \( H \) into a union of complete bipartite subgraphs, and applies the preceding procedure to each subgraph.

To obtain the decomposition of \( H \), let \( a \) and \( b \) denote the endpoints of \( e^* \); See Figure 3.2. We store the points of \( R \) in a 2-dimensional range tree, where the points are stored in the primary tree in the order of the slopes of the lines connecting their \( xy \)-projections with \( a \). At each node \( v \) of the primary tree, we store a secondary range tree of the points stored at the subtree rooted at \( v \), so that the points are stored at the secondary tree in the order of the slopes of the lines connecting their \( xy \)-projections with \( b \). Given a point \( l \in L \), we can query the two-level range tree with \( l \), to compute the set of all points \( r \in R \), for which \( l^*r^* \) crosses \( e^* \), as a disjoint union of \( O(\log^2 m) \) precomputed canonical subsets. Next, we associate with each canonical secondary subset \( R_i \subseteq R \) a subset \( L_i \subseteq L \), consisting of all the points \( l \in L \) for which \( R_i \) is one of the subsets of \( R \) whose union is the output to the query with \( l \). Finally, we test the visibility between each pair \( L_i \times R_i \) by the explicit procedure described above, in \( O(|L_i| + |R_i|) \) time. Hence, testing the visibility of \( L \times R \) across \( e \) can therefore be performed in time \( O(\sum_i |L_i| + \sum_i |R_i|) \). Since each point \( l \in L \) and \( r \in R \) appears in at most \( O(\log^2 |R|) \) pairs \( L_i \times R_i \), the overall cost is \( O((|L| + |R|) \log^2 |R|) \).
Figure 3.2: Handling a subproblem $L \times R$ for an edge $e$ of $T$. $L \times R_3$ is one of the complete bipartite graphs in the decomposition.
Repeating this procedure to each edge $e$ of $T$, we get a procedure, with $O(nm \log^2 m)$ running time, for solving the mutual visibility problem. We note that the storage used by the algorithm is $O(n)$ (to store the terrain) plus $O(m \log m)$ which is the storage used by a single 2-dimensional range tree. That is, we have shown:

**Theorem 3.3.1** Given a polyhedral terrain $T$ in 3-space with $n$ vertices and a set $P$ of $m$ points, all lying above $T$, we can determine, in $O(nm \log^2 m)$ time and $O(m \log m + n)$ space, whether all pairs of points of $P$ are mutually visible with respect to $T$. 
Chapter 4

Emptiness and Reporting Queries with Semialgebraic Ranges and Their Applications

For thousands more years the mighty ships tore across the empty wastes of space and finally dived screaming on to the first planet they came across — which happened to be the Earth — where due to a terrible miscalculation of scale the entire battle fleet was accidentally swallowed by a small dog. (The Hitchhiker’s Guide to the Galaxy)

4.1 Overview

In this chapter we extend Matoušek’s technique for half-space emptiness queries to more general semialgebraic ranges. In a typical range emptiness searching (resp., reporting) problem, we are given a set \( P \) of \( n \) points in \( \mathbb{R}^d \), and wish to preprocess it into a data structure that supports efficient range emptiness (resp., reporting) queries, in which we specify a range \( \sigma \), which, in general, is a semialgebraic set in \( \mathbb{R}^d \) of constant description complexity, and wish to determine whether \( P \cap \sigma = \emptyset \), or to report all the points in \( P \cap \sigma \). Range emptiness searching and reporting arise in many applications, and have been treated by Matoušek [57] in the special case where the ranges are half-spaces bounded by hyperplanes. As shown in [57], the two problems are closely related, and have solutions (for the case of half-spaces) with similar performance bounds. In this chapter we extend the analysis to general semialgebraic ranges, and show how to adapt Matoušek’s technique, without the need to linearize the ranges into a higher-dimensional space. This yields more efficient solutions to several useful problems, and we demonstrate the new technique in four applications, with the following results:

(i) We present an algorithm for ray shooting amid \( n \) balls in \( \mathbb{R}^3 \), which uses \( O(n) \) storage
and $O^*(n)$ preprocessing, and answers a query in $O^*(n^{2/3})$ time, improving the previous bound of $O^*(n^{3/4})$.

(ii) We present an algorithm that preprocesses, in $O^*(n)$ time, a set $P$ of $n$ points in $\mathbb{R}^3$ into a data structure with $O(n)$ storage, so that, for any query line $\ell$ (or, for that matter, any simply-shaped convex set), the point of $P$ farthest from $\ell$ can be computed in $O^*(n^{1/2})$ time. This improves the solution of [5] with query time $O^*(n^{2/3})$. This in turn yields an algorithm that computes the largest-area triangle spanned by $P$ in time $O^*(n^{26/11})$, as well as similar nontrivial algorithms for computing the largest-perimeter or largest-height triangle spanned by $P$.

(iii) We present an algorithm that preprocesses, in $O^*(n)$ time, a set $P$ of $n$ points in $\mathbb{R}^3$ into a data structure, so that, for any query $\alpha$-fat triangle $\Delta$ (i.e., a triangle all of whose angles are at least $\alpha$; see also Chapter 2), we can determine whether $\Delta \cap P$ is empty. The algorithm uses $O(n)$ storage and can answer a query in $O^*(1)$ time, that is, $O(n^\varepsilon)$ time for any $\varepsilon > 0$. The previous solution for this problem uses $O(n \log^2 n)$ storage and answers a query in $O(\log^3 n)$ time. Alternatively, we can report, in $O(n^{\varepsilon} + k)$ time, the points of $\Delta \cap P$, where $k = |\Delta \cap P|$ and $\varepsilon > 0$ is arbitrary.

(iv) Finally we present an algorithm that preprocesses, in $O^*(n)$ time, a set $P$ of $n$ points in $\mathbb{R}^2$ into a data structure, so that, given any query semidisk $c$, or a circular cap larger than a semidisk, we can determine, in $O^*(1)$ time, whether $c \cap P$ is empty, or report the $k$ points in $c \cap P$ in $O^*(1) + O(k)$ time.

Adapting the recent techniques of [15, 16, 17], we can turn our solutions into efficient algorithms for approximate range counting (with small relative error) for the cases mentioned above. Since the technical details of this extension are somewhat involved, we dedicate a separate chapter (Chapter 5) to this extension.

Most of the special notation used in this chapter has been introduced in Chapter 1 (see in particular Section 1.1).

Our technique is closely related to the notions of nearest- or farthest-neighbor generalized Voronoi diagrams, and of the union or intersection of geometric objects, where sharper bounds on the combinatorial complexity of these structures (or, more precisely, on the size of an elementary-cell decomposition of the corresponding portion of space; see below) yield faster range emptiness searching or reporting algorithms.

The function $\zeta(r)$. In Lemma 4.2.1 and Theorem 4.2.2, we use a function $\zeta(r)$ that bounds the number of elementary cells in a decomposition of the complement of the union of any $r$ ranges of $\Gamma$. (See Section 1.1.7.) We assume that $\zeta(r)$ is “well behaved”, in the sense that for each $c > 0$ there exists $c' > 0$ such that $\zeta(cr) \leq c' \zeta(r)$ for every $r$. We also assume that $\zeta(r) = \Omega(r)$. Both assumptions are natural and hold in practically all applications.

1Actually, it suffices to assume that the query caps are such that their central angles are at least some constant positive angle; see Section 4.5.
4.2 Semialgebraic Range Reporting or Emptiness Searching

Shallow cutting in the semialgebraic case. We begin by extending the shallow cutting lemma of Matoušek [57] to the more general setting of semialgebraic ranges. This extension is fairly straightforward, although it involves several technical steps that deserve to be highlighted. The analysis makes use of the notations introduced in Chapter 1, such as elementary cells and elementary cell partition.

Lemma 4.2.1 (Extended Shallow Cutting Lemma) Let $\Gamma$ be a collection of $n$ semialgebraic ranges in $\mathbb{R}^d$. Assume that the complement of the union of any subset of $m$ ranges in $\Gamma$ can be decomposed into at most $\zeta(m)$ elementary cells, for a well-behaved function $\zeta$ as above. Then, for any $r \leq n$, there exists a $\left(1 - \frac{1}{r}\right)$-cutting $\Xi$ with the following properties:

(i) The union of the cells of $\Xi$ contains the complement of the union of $\Gamma$.

(ii) $\Xi$ consists of $O(\zeta(r))$ elementary cells.

(iii) The complement of the union of the cells of $\Xi$ is contained in a union of $O(r)$ ranges in $\Gamma$.

See Figure 4.1 for an illustration.

Proof: The proof is a fairly routine adaptation of the proof in [57]. We employ a variant of the method of Chazelle and Friedman [25] for constructing the cutting. Let $\Gamma'$ be a random sample of $O(r)$ ranges of $\Gamma$, and let $E'$ denote the complement of the union of $\Gamma'$. By assumption, $E'$ can be decomposed into at most $O(\zeta(r))$ elementary cells. The resulting collection $\Xi$ of these cells is such that their union clearly contains the complement of the union of $\Gamma$. Moreover, the complement of the union of $\Xi$ is the union of the $O(r)$ ranges of $\Gamma'$. Hence, $\Xi$ satisfies all three conditions (i)–(iii), but it may fail to be a $\left(1 - \frac{1}{r}\right)$-cutting. This latter property is enforced as in [25], by further decomposing each cell $\tau$ of $\Xi$ that is crossed by more than $n/r$ ranges of $\Gamma$, using additional subsamples from the surfaces that cross $\tau$. Specifically, for each cell $\tau$ of $\Xi$, let $\Gamma_\tau$ denote the subset of those ranges in $\Gamma$ that cross $\tau$, and put $\xi_\tau = |\Gamma_\tau| r/n$. If $\xi_\tau > 1$, we sample $q = O(\xi_\tau \log \xi_\tau)$ ranges from $\Gamma_\tau$, construct the complement of the union of these ranges, decompose it into at most $\zeta(q)$ elementary cells, and clip them to within $\tau$. The resulting collection $\Xi'$ of subcells, over all cells $\tau$ of the original $\Xi$, clearly satisfies (i). The analysis of [25] (see also [7]) establishes an exponential decay property on the number of cells of $\Xi$ that are crossed by more than $\xi n/r$ ranges, as a function of $\xi$. Specifically, as in [7], the expected number of such cells is $O(2^{-\xi}E\{\zeta(|\Gamma''|)\})$, where $\Gamma''$ is another random sample of $\Gamma$, where each member of $\Gamma$ is chosen with probability $\frac{r}{n\xi}$. This property implies, as usual [25], that $\Xi'$ is (with high probability) a $\left(1 - \frac{1}{r}\right)$-cutting, and it also implies that the size of $\Xi'$ is still $O(\zeta(r))$, assuming $\zeta$ to be well behaved. Since we have only refined the original cells of $\Xi$, the number of ranges that cover the complement of the union of the final cells is still $O(r)$.

Figure 4.1: A planar point set and a collection $\Gamma$ of upper half-planes. A random sample of the lines bounding these ranges is shown in bold, with a decomposition of the region below their lower envelope, which contains the region below the lower envelope of all the bounding lines, drawn shaded.

A special case that arises frequently is where each range in $\Gamma$ is an upper half-space bounded by the graph of some continuous $(d-1)$-variate function. In this case the complement $K$ of the union of $r$ ranges is the portion of space that lies below the lower envelope of the bounding graphs. In this case, it suffices to decompose the graph of the lower envelope itself into at most $\zeta(r)$ elementary cells. Indeed, having done that, we can extend each cell $\tau$ within the envelope into the cell $\tau^-$ consisting of all points that lie vertically below $\tau$. The new cells decompose $K$ and are also elementary. A symmetric situation arises when $\Gamma$ is a collection of lower half-spaces.

As already discussed in Section 1.1.7, obtaining tight or nearly tight bounds for $\zeta(r)$ is still a major open problem for many instances of the above setup. For example, decomposing a lower envelope of $r$ $(d-1)$-variate functions of constant description complexity into $O^*(r^{d-1})$ elementary cells is still open for any $d \geq 4$. This bound is best possible in the worst case, since it is the worst-case tight bound (up to the hidden subpolynomial factor) on the complexity of such an undecomposed envelope [69]. The cases $d = 2$ (upper envelope of curves in the plane) and $d = 3$ (upper envelope of 2-dimensional surfaces in 3-space) are easy. In these cases $\zeta(r)$ is proportional to the complexity of the envelope, which in the worst case is near-linear for $d = 2$ and near-quadratic for $d = 3$ [69]. In higher dimensions, the only general-purpose bound known to date is the upper bound obtained by computing the vertical decomposition of the entire arrangement of the given surfaces, and extracting from it the relevant cells that lie on or above the envelope. In particular (see also Section
4.2 Semialgebraic Range Reporting or Emptiness Searching

1.1.8), for $d = 4$ the bound is $\zeta(r) = O^*(r^4)$, as follows from the results of [53]. This leaves a gap of about a factor of $r$ between this bound and the bound $O^*(r^3)$ on the complexity of the undecomposed envelope. For arbitrary $d \geq 4$, the bound is $\zeta(r) = O^*(r^{2d-4})$, with progressively increasing gaps from the lower bound $\Omega^*(r^{d-1})$. Of course, in certain special cases, most notably the case of hyperplanes, as studied in [57], both the envelope and its decomposition have (considerably) smaller complexity, with asymptotically equal bound (namely $O(r^{\lfloor d/2 \rfloor})$).

The situation with the complexity of the union of geometric objects is even worse. While considerable progress was recently made on many special cases in two and three dimensions (see [8] for a recent comprehensive survey), there are only very few sharp bounds on the complexity of unions in higher dimensions. Worse still, even when a sharp bound on the complexity of the union is known, obtaining comparable bounds on the complexity of a decomposition of the complement of the union is a much harder problem (in $d \geq 3$ dimensions). As an example, the union of $n$ infinite cylinders in 3-space is known to have near-quadratic complexity [11, 42], but it is still an open problem whether its complement can be decomposed into a near-quadratic number of elementary cells.

**Partition theorem for shallow semialgebraic ranges.** We next apply the new shallow cutting lemma to construct an elementary cell partition of a given input point set $P$, with respect to a specific (finite) set $Q$ of ranges. This is done in a fairly similar way to that in [5] (see also [57, 58]). A major difference in handling the semialgebraic case is the construction of a set $Q$ of ranges (called a test-set, as in [5, 57, 58]) that will be (a) small enough, and (b) representative of all shallow (or empty) ranges, in a sense discussed in detail below. The method given in [57] does not work in the general semialgebraic case, and different, sometimes ad-hoc approaches need to be followed.

The following theorem summarizes the main part of the construction (except for the construction of $Q$).

**Theorem 4.2.2 (Extended Partition Theorem)** Let $P$ be a set of $n$ points in $\mathbb{R}^d$, let $\Gamma$ be a family of semialgebraic ranges of constant description complexity, and let $1 \leq r < n$ be fixed. Let $Q$ be another finite collection (not necessarily a subset of $\Gamma$) of semialgebraic ranges of constant description complexity with the following properties: (i) The ranges in $Q$ are all $(n/r)$-shallow. (ii) The complement of the union of any $m$ ranges of $Q$ can be decomposed into at most $\zeta(m)$ elementary cells, for any $m$. (iii) Any $(n/r)$-shallow range $\gamma \in \Gamma$ can be covered by the union of at most $\delta$ ranges of $Q$, where $\delta$ is a constant.

Then there exists an elementary cell partition $\Pi$ of $P$, of size $O(r)$, into subsets of size between $n/r$ and $2n/r$, such that the crossing number of any $(n/r)$-shallow range in $\Gamma$ with the cells of $\Pi$ is either $O(r/\zeta^{-1}(r) + \log r \log |Q|)$, if there exists a fixed $\varepsilon > 0$ such that $\zeta(r)/r^{1+\varepsilon}$ is monotone increasing, or $O(r \log r/\zeta^{-1}(r) + \log r \log |Q|)$, otherwise.

The proof, which, again, is similar to those in [5, 57, 58], proceeds through the following
Lemma 4.2.3 Let \( P \) be a set of \( n \) points in \( \mathbb{R}^d \), and \( 1 \leq r < n \) a parameter. Let \( Q \) be a set of \((n/r)\)-shallow ranges, with the property that the complement of the union of any subset of \( n \) ranges of \( Q \) can be decomposed into at most \( \zeta(m) \) elementary cells, for any \( m \). Then there exists a subset \( P' \subseteq P \) of at least \( n/2 \) points and an elementary cell partition \( \Pi = \{(P_1, s_1), \ldots, (P_m, s_m)\} \) for \( P' \) with \( n/r \leq \left| P_i \right| \leq 2n/r \) for all \( i \), such that each range of \( Q \) crosses at most \( O(r/\zeta^{-1}(r) + \log |Q|) \) cells \( s_i \) of \( \Pi \).

Proof: We will inductively construct disjoint sets \( P_1, \ldots, P_m \subseteq P \) of size \( n/r \) (to simplify the presentation we ignore the floor rounding) and elementary cells \( s_1, \ldots, s_m \) such that \( P_i \subseteq s_i \) for each \( i \). The construction terminates when \(|P_1 \cup \cdots \cup P_m| \geq n/2 \). Suppose that \( P_1, \ldots, P_{i-1} \) have already been constructed, and set \( P_i := P \setminus \bigcup_{j<i} P_j \). We construct \( P_i \) as follows: For a range \( \sigma \in Q \), let \( \kappa_i(\sigma) \) denote the number of cells among \( s_1, \ldots, s_{i-1} \) crossed by \( \sigma \). We define a weighted collection \((Q, w_i)\) of ranges, so that each range \( \sigma \in Q \) appears with weight (or multiplicity) \( w_i(\sigma) = 2^{\kappa_i(\sigma)} \). We put \( w_i(Q) = \sum_{\sigma \in Q} w_i(\sigma) \). By Lemma 4.2.1 and by our assumption that the function \( \zeta(r) \) is well behaved, there exists a \((1/t)\)-cutting \( \Xi_i \) for the weighted collection \((Q, w_i)\) of size at most \( r/4 \), for an appropriate choice of \( t = c_1 \zeta^{-1}(r) \), for some constant \( c_1 > 0 \), with the following properties: The union of \( \Xi_i \) contains the complement of the union of \( Q \), and the complement of the union of \( \Xi_i \) is contained in the union of \( O(t) = c_2 t \) ranges of \( Q \), for some constant \( c_2 > 0 \). Since all these ranges are \((n/r)\)-shallow, the number of points of \( P \) not in the union of \( \Xi_i \) is at most \( c_2 t \cdot (n/r) = n \cdot c_2 (c_1 \zeta^{-1}(r))/r \), and our assumption that \( \zeta(r) \geq r \) imply that this is smaller than \( n/4 \), if we choose \( t \) appropriately (i.e., choose \( c_1 \) such that \( c_2 c_1 < 1/4 \)). Since we assume that \( |P_i| \geq n/2 \), it follows that at least \( n/4 \) points of \( P_i' \) lie in the union of the at most \( r/4 \) cells of \( \Xi_i \). By the pigeonhole principle, there is a cell \( s_i \) of \( \Xi_i \) containing at least \( n/r \) points of \( P_i' \). We take \( P_i \) to be some subset of \( P_i' \cap s_i \) of size exactly \( n/r \), and take \( s_i \) to be the cell in the partition which contains \( P_i \).

We next establish the asserted bound on the crossing numbers between the ranges of \( Q \) and the elementary cells \( s_1, \ldots, s_m \), in the following standard manner. The final weight \( w_m(\sigma) \) of a range \( \sigma \in Q \) with crossing number \( \kappa(\sigma) \) (with respect to the final partition) is \( 2^{\kappa(\sigma)} \). On the other hand, each newly added cell \( s_i \) is crossed by ranges of \( Q \) of total weight \( O(w_i(Q)/t) = O(w_i(Q)/\zeta^{-1}(r)) \), because \( s_i \) is an elementary cell of the corresponding weighted \((1/t)\)-cutting \( \Xi_i \). The weight of each of these crossing ranges is doubled at the \( i \)-th step, and the weight of all the other ranges remains unchanged. Thus \( w_{i+1}(Q) \leq w_i(Q)(1 + O(1/\zeta^{-1}(r))) \). Hence, for each range \( \sigma \in Q \) we have

\[
w_m(\sigma) \leq w_m(Q) \leq |Q| \left(1 + O\left(1/\zeta^{-1}(r)\right)\right)^m = |Q| \left(1 + O\left(1/\zeta^{-1}(r)\right)\right)^{O(r)} \leq |Q| e^{O(r/\zeta^{-1}(r))},
\]

and thus \( \kappa(\sigma) = \log w_m(\sigma) = O(r/\zeta^{-1}(r) + \log |Q|) \).
4.2 Semialgebraic Range Reporting or Emptiness Searching

Discussion. The limitation of Lemma 4.2.3 is that the bound that it derives (a) applies only to ranges in $\mathcal{Q}$, and (b) includes the term $\log |\mathcal{Q}|$. An ingenious component of the analysis in [57] overcomes both problems, by choosing a test set $\mathcal{Q}$ of ranges whose size is only polynomial in $r$ (and, in particular, is independent of $n$), which is nevertheless sufficiently representative of all shallow ranges, in the sense that the crossing number of any $(n/r)$-shallow range is $O(\max\{\kappa(\sigma) \mid \sigma \in \mathcal{Q}\})$. This implies that Lemma 4.2.3 holds for all shallow ranges, with the stronger bound which does not involve $\log |\mathcal{Q}|$ (assuming that $r/\zeta^{-1}(r)$ dominates the polylogarithmic term $\log |\mathcal{Q}|$).

Unfortunately, the technique of [57] does not extend to the case of semialgebraic ranges, as it crucially relies on the linearity of the ranges.

The following lemma gives a sufficient condition for a test set $\mathcal{Q}$ to be representative of the relevant shallow ranges, in the sense that $\mathcal{Q}$ satisfies the assumptions made in Theorem 4.2.2. That is:

**Lemma 4.2.4** Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and let $\Gamma$ be a family of semialgebraic ranges with constant description complexity. Consider an elementary-cell partition $\Pi = \{(P_1, s_1), \ldots, (P_r, s_r)\}$ of $P$ such that $n/r \leq |P_i| \leq 2n/r$ for each $i$. Let $Q$ be a finite set of $(n/r)$-shallow ranges (not necessarily ranges of $\Gamma$), so that the maximal crossing number of a range $q \in Q$ with respect to $\Pi$ is $\kappa$. Then, for any range $\gamma \in \Gamma$ which is contained in the union of at most $\delta$ ranges of $Q$ (for some constant $\delta$), the crossing number of $\gamma$ is at most $(\kappa + 1)\delta$.

**Proof:** Let $\gamma \in \Gamma$ be a range for which there exist $\delta$ ranges $q_1, \ldots, q_\delta$ of $Q$ such that $\gamma \subseteq q_1 \cup \cdots \cup q_\delta$. Then, if $\gamma$ crosses a cell $s_i$ of $\Pi$, then at least one of the covering ranges $q_j$ must either cross $s_i$ or fully contain $s_i$. The number of cells of $\Pi$ that can be crossed by any single $q_j$ is at most $\kappa$, and each $q_j$ can fully contain at most one cell of $\Pi$ (because $q_j$ is $(n/r)$-shallow). Hence, the overall number of cells of $\Pi$ that $\gamma$ can cross is at most $(\kappa + 1)\delta$, as asserted.

**Proof of Theorem 4.2.2.** Apply Lemma 4.2.3 to the input set $P_0 = P$, with parameter $r_0 = r$. This yields an elementary-cell partition $\Pi_0$ for (at least) half of the points of $P_0$, which satisfies the properties of that lemma. Let $P_1$ denote the set of the remaining points of $P_0$, and set $r_1 = r_0/2$. Apply Lemma 4.2.3 again to $P_1$ with parameter $r_1$, obtaining an elementary cell partition $\Pi_1$ for (at least) half of the points of $P_1$. We iterate this process $k = O(\log r)$ times, until the set $P_k$ has fewer than $n/r$ points. We take $\Pi$ to be the union of all the elementary-cell partitions $\Pi_i$ formed so far, together with one large cell containing

---

2It uses point-hyperplane duality, and exploits the fact that a half-space (bounded by a hyperplane) intersects a simplex if and only if it contains a vertex of the simplex, which is false in the general semialgebraic case.

3By choosing a slightly smaller value for $r$ in the construction of the partition, we can even rule out the possibility that a range $q_j$ fully contains a cell of $\Pi$. This however has no effect on the asymptotic bounds that the analysis derives.
all the remaining points of $P_k$. The resulting elementary-cell partition of $P$ consists of at most $1 + r + r/2 + r/4 + \ldots \leq 2r$ subsets, each of size at most $2n/r$ (the sizes are all equal to $n/r$ except for the last set). The crossing number of a range in $Q$ is, by Lemma 4.2.3,

$$O \left( \sum_{i=1}^{\log r} \left( \frac{(r/2^i)}{\zeta^{-1}(r/2^i)} + \log |Q| \right) \right).$$

Our assumptions on $\zeta$ imply that if there exists an $\varepsilon > 0$ such that $\zeta(r)/r^{1+\varepsilon}$ is monotone increasing, then the first terms add up to $O(r/\zeta^{-1}(r))$; otherwise we can bound their sum by $O(r \log r/\zeta^{-1}(r))$. Hence, by the properties of $Q$ and by Lemma 4.2.4, the crossing number of any $(n/r)$-shallow range, and in particular any empty range, is also $O(r/\zeta^{-1}(r) + \log |Q| \log r)$ or $O(r \log r/\zeta^{-1}(r) + \log |Q| \log r)$, respectively.

**Partition trees and reporting or emptiness searching.** As in the classical works on range searching [5, 57, 58], we choose $r$ to be a sufficiently large constant and apply Theorem 4.2.2 recursively, and obtain a partition tree $T$, where each node $v$ of $T$ stores a subset $P_v$ of $P$ and an elementary cell $\sigma_v$ enclosing $P_v$. (The root stores the entire $P$ and some sufficient large enclosing cell.) The children of a node $v$ are obtained from an elementary cell partition of $P_v$—each of them stores one of the resulting subsets of $P_v$ and its enclosing cell. At each leaf, the size of the subset that it stores is $O(r)$.

Testing a range $\gamma$ for emptiness is done by searching with $\gamma$ in $T$. At each visited node $v$, where $\gamma \cap \sigma_v \neq \emptyset$, we test whether $\gamma \supseteq \sigma_v$, in which case $\gamma$ is not empty. Otherwise, we find the children of $v$ whose cells are intersected by $\gamma$. If there are too many of them (more than the bound prescribed in Theorem 4.2.2) we know that $\gamma$ is not empty. Otherwise, we recurse at each child of $v$ as above.

Reporting is performed in a similar manner. If $\sigma_v \subseteq \gamma$ we output all of $P_v$. Otherwise, we find the children of $v$ whose cells are intersected by $\gamma$. If there are too many of them we know that $\gamma$ is not $(n_v/r)$-shallow (with respect to $P_v$), so, if $r$ is a constant, we can afford to check every element of $P_v$ for containment in $\gamma$, and output those points that do lie in $\gamma$. If there are not too many children of this kind, we recurse in each of them.

The efficiency of the search depends on the function $\zeta(m)$ and on the size of the test set $Q$. If $\zeta(m) = O^*(m^k)$ and the size of $Q$ is negligible (typically, polynomial in $r$) then, as is easily verified (and similar to the case of half-spaces) an emptiness query takes $O^*(n^{1-1/k})$ time, and a reporting query takes $O^*(n^{1-1/k}) + O(t)$ time, where $t$ is the output size. Thus making $\zeta$ (i.e., $k$) small is the main challenge in this technique.

**A general recipe for constructing good test sets.** Let $\Gamma$ be the given collection of semialgebraic ranges of constant description complexity. As above, we assume that each range $\gamma \in \Gamma$ has $t$ degrees of freedom, for some constant parameter $t$, so it can be represented as a point $\gamma^*$ in a $t$-dimensional parametric space, which, for convenience, we
denote as \( \mathbb{R}^d \). Each input point \( p \in P \) is mapped to a region \( K_p \), which is the locus of all points representing ranges which contain \( p \).

We fix a parameter \( r \geq 1 \), and choose a random sample \( N \) of \( ar \log r \) points of \( P \), where \( a \) is a sufficiently large constant. We form the set \( N^* = \{ K_p \mid p \in N \} \), construct the arrangement \( A(N^*) \), and let \( V = A_{\leq k}(N^*) \) denote the region consisting of all points contained in at most \( k \) ranges of \( N^* \), where \( k = b \log r \) and \( b \) is an absolute constant that we will fix later. We decompose \( V \) into elementary cells, using, e.g., vertical decomposition [69]. In the worst case, we get \( O^*(r^{2t-4}) \) elementary cells (for \( t \geq 4 \)) [23, 53].

Let \( \tau \) be one of these cells. We associate with \( \tau \) a generalized range \( \gamma_\tau \) in \( \mathbb{R}^d \), which is the union \( \bigcup \{ \gamma \mid \gamma^* \in \tau \} \). Since \( \tau \) has constant description complexity, as do the ranges of \( \Gamma \), it is easy to show that \( \gamma_\tau \) is a semialgebraic set of constant description complexity (see [20]).

We define the test set \( Q \) to consist of all the generalized ranges \( \gamma_\tau \), over all cells \( \tau \) in the decomposition of \( V \), and claim that, with high probability (and with an appropriate choice of \( b \)), \( Q \) is a good test set, in the following three aspects.

(i) **Compactness.** \( |Q| = O^*(r^{2t-4}) \); that is, the size of \( Q \) is polynomial in \( r \) and independent of \( n \).

(ii) **Shallowness.** Each range \( \gamma_\tau \) in \( Q \) is \( \beta(n/r) \)-shallow with respect to \( P \), for some constant parameter \( \beta \).

(iii) **Containment.** Every \( (n/r) \)-shallow range \( \gamma \in \Gamma \) is contained in a single range \( \gamma_\tau \) of \( Q \).

Property (i) is obvious. Consider the range space \( (P, \Gamma^*) \), where \( \Gamma^* \) consists of all generalized ranges \( \gamma_\tau \), over all elementary cells \( \tau \) of the form arising in the above vertical decomposition. It is a fairly easy exercise to show that \((P, \Gamma^*)\) also has finite VC-dimension. See, e.g., [69]. By Theorem 1.1.2, if \( a \) is a sufficiently large constant (proportional to the VC-dimension of \((P, \Gamma^*)\)) then \( N \) is a shallow \((1/r)\)-net for both range spaces \((P, \Gamma)\) and \((P, \Gamma^*)\), with high probability, so we assume that \( N \) is indeed such a shallow \((1/r)\)-net. See Section 1.1.9 for details.

Let \( \gamma_\tau \in Q \). Note that any point \( p \in P \) in \( \gamma_\tau \) lies in a range \( \gamma \in \Gamma \) with \( \gamma^* \in \tau \). By definition, \( \gamma^* \) also belongs to \( K_p \), and so \( K_p \) crosses or fully contains \( \tau \). Since \( \tau \) is \((b \log r)\)-shallow in \( A(N^*) \), it is fully contained in at most \( b \log r \) regions \( K_p \), for \( p \in N \) (and is not crossed by any such region). Hence \( |\gamma_\tau \cap N| < b \log r \), so, since \( N \) is a shallow \((1/r)\)-net for \((P, \Gamma^*)\), we have \( |\gamma_\tau \cap P| < c(b+1)n/r \), for some constant \( c > 0 \), so \( \gamma_\tau \) is \((c(b+1)n/r)\)-shallow (see Section 1.1.9), which establishes (ii) with \( \beta = c(b+1) \).

For (iii), let \( \gamma \in \Gamma \) be an \((n/r)\)-shallow range. Since \( N \) is a shallow \((1/r)\)-net for \((P, \Gamma)\), and \( |\gamma \cap P| \leq |P|/r \), we have \( |\gamma \cap N| \leq 2c \log r \), for some constant \( c > 0 \). Hence, with \( b \geq 2c \), we have \( \gamma \in V \), so there is a cell \( \tau \) of the decomposition which contains \( \gamma \). This, by construction, implies that \( \gamma \subseteq \gamma_\tau \), thus establishing (iii).

\(^4\)Here, in this dual construction, we do not need any sharper bound; any bound polynomial in \( r \) is sufficient for our purpose.
To make $Q$ a really good test set, we also need the following fourth property:

(iv) **Efficiency.** There exists a good bound on the associated function $\zeta(m)$, bounding the size of a decomposition of the complement of the union of any $m$ ranges of $Q$.

The potentially rather complex shape of these generalized ranges makes it harder to obtain, in general, a good bound on $\zeta$.

In what follows, we manage to use this general recipe in two of our four applications (ray shooting amid balls and range fullness searching), with good bounds on the corresponding functions $\zeta(\cdot)$. In two other planar applications (range emptiness searching with fat triangles and with circular caps), we abandon the general technique, and construct ad hoc good test sets (satisfying all four criteria).

**Remark:** In the preceding construction, we wanted to make sure that every $\frac{n}{r}$-shallow range $\gamma \in \Gamma$ is covered by a range of $Q$. If we only need this property for empty ranges $\gamma$ (which is the case for emptiness testing), it suffices to consider only the $0$-level of $A(N^*)$, i.e., the complement of the union of $N^*$. Other than this simplification, the construction proceeds as above.

### 4.3 Fullness Searching and Reporting Outliers for Convex Ranges

Let $P$ be a set of $n$ points in 3-space, and let $\Gamma$ be a set of convex ranges of constant description complexity. We wish to preprocess $P$ in near-linear time into a data structure of linear size, so that, given a query range $\gamma \in \Gamma$, we can efficiently determine whether $\gamma$ contains all the points of $P$. Alternatively, we want to report all the points of $P$ that lie outside $\gamma$. (This is clearly a special case of range emptiness searching or range reporting, if one considers the complements of the ranges in $\Gamma$.) We first focus on the range fullness problem; the extension to reporting "outliers" is similar to the standard treatment of reporting queries, as described earlier, and we discuss it briefly later on.

We present a solution to this problem, with $O^*(n^{1/2})$ query time, thereby improving over the best known general bound of $O^*(n^{2/3})$, given in [5], which applies to any range searching (e.g., range counting) with semialgebraic sets (of constant description complexity) in $\mathbb{R}^3$.

To apply our technique to this problem we first need to build a good test set. Since fullness searching is complementary to emptiness searching, we need a property complementary to that assumed in Theorem 4.2.2 (see also Lemma 4.2.4). In fact, we will enforce the property that every full range $\gamma$ fully contains a single range of $Q$, which is "almost full" (contains at least $n - \frac{n}{r}$ points of $P$).

As above, assuming the ranges of $\Gamma$ to have $t$ degrees of freedom, we map each range $\gamma \in \Gamma$ to a point $\gamma^*$ in $\mathbb{R}^t$. A point $p \in \mathbb{R}^3$ is mapped to a region $K_p$ which is the locus of all the points $\gamma^*$ that correspond to ranges $\gamma$ which contain $p$. We fix $r < n$, take a random
sample \( N \) of \( O(r \log r) \) points of \( P \) (with a sufficiently large constant of proportionality), construct the intersection \( I = \bigcap_{p \in N} K_p \), and decompose it into elementary cells. For each resulting cell \( \sigma \), let \( \gamma_\sigma \) denote the intersection \( \bigcap_{\gamma \in \sigma} \gamma \). As above, since \( \sigma \) has constant description complexity, \( \gamma_\sigma \) is a semialgebraic set of constant description complexity. Note that, since the ranges in \( \Gamma \) are convex, each range \( \gamma_\sigma \) is also convex (albeit of potentially more complex shape than that of the original ranges).

Define the test set \( Q \) to consist of all the generalized ranges \( \gamma_\sigma \), over all cells \( \sigma \) in the decomposition of \( I \). We argue that, with high probability, \( Q \) satisfies all four properties required from a good test set: (i) Compactness: As above, the size of \( Q \) is polynomial in \( r \) (it is at most \( O^*(r^{2t-4}) \)). (ii) Shallowness (or, rather, “almost fullness”): For each cell \( \sigma \) and any \( \gamma \in \Gamma \) with \( \gamma^* \in \sigma \), \( \gamma^* \) lies in all the sets \( K_p \), for \( p \in N \), and thus \( N \subseteq \gamma \). By construction, we also have \( N \subseteq \gamma_\sigma \). Apply the \( \varepsilon \)-net theory \([49]\) to the range space \((P, \Gamma)\), where the ranges of \( \Gamma \) are complements of ranges of the same form as the ranges \( \gamma_\sigma \). Since \( \gamma_\sigma \cap N = \emptyset \) for each cell \( \sigma \) in the decomposition, we have, with high probability, the property that for each cell \( \sigma \) of \( I \), \( \gamma_\sigma \) contains at least \( n - n/r \) points of \( P \), so it is an “almost full” range. (iii) Containment: Let \( \gamma \in \Gamma \) be a full range. Then, in particular, \( N \subseteq \gamma \). Then \( \gamma^* \subseteq I \), and let \( \sigma \) be the cell of \( I \) containing \( \gamma^* \). Then, by construction, \( \gamma_\sigma \subseteq \gamma \). (iv) Efficiency: Finally, we show that the complexity of a decomposition of the intersection of any \( m \) ranges in \( Q \) is \( O^*(m^2) \), so \( \zeta(m) = O^*(m^2) \).

**Claim 4.3.1** Let \( Q \) be a set of convex ranges, in \( \mathbb{R}^3 \) of constant description complexity. The intersection, \( K \), of any \( m \) ranges \( q_1, \ldots, q_m \in Q \) can be decomposed into \( O^*(m^2) \) elementary cells.

**Proof:** Since all ranges in \( Q \) are convex, \( K \) is convex too. Assume, for simplicity of presentation, that \( K \) is nonempty and has nonempty interior, and fix a point \( o \) in that interior. We can regard the boundary of each \( q_i \) as the graph of a bivariate function \( \rho = F_i(\theta, \phi) \) in spherical coordinates about \( o \). Then \( \partial K \) is the graph of the lower envelope of these functions. Since the \( q_i \)'s have constant description complexity, (the graph of) each \( F_i \) is also a semialgebraic set of constant description complexity\(^5\). Hence the combinatorial complexity of \( \partial K \) is \( O^*(m^2) \) \([69]\). Moreover, since \( \partial K \) is 2-dimensional, we can partition it into \( O^*(m^2) \) trapezoidal-like elementary cells, using a variant of the planar vertical decomposition technique, and then extend each such cell \( \tau_0 \) to a 3-dimensional cone-like cell \( \tau \), which is the union of all segments connecting \( o \) to the points of \( \tau_0 \). The resulting cells \( \tau \) constitute a decomposition of \( K \) into \( O^*(m^2) \) elementary cells, as claimed.

Using the machinery developed in the preceding section, we therefore obtain the following result.

**Theorem 4.3.2** Let \( P \) be a set of \( n \) points in \( \mathbb{R}^3 \), and let \( \Gamma \) be a family of convex ranges of constant description complexity. Then one can construct, in near linear time, a data

\(^5\)With an appropriate algebraic re-parametrization of the spherical coordinates, of course.
structure of linear size so that, for any range $\gamma \in \Gamma$, it can determine, in $O^*(n^{1/2})$ time, whether $\gamma$ is full.

**Reporting outliers.** To extend the above approach to the problem of reporting outliers, we apply a construction similar to that in the “general recipe” presented above. That is, we take the $b \log r$ deepest levels of $\mathcal{A}(N)$, for an appropriate constant $b$, decompose them into elementary cells, and construct a generalized range $\gamma_\sigma$ for each of these cells $\sigma$. The decomposition is obtained by a variant of the approach used by Agarwal et al. [3] for vertical decompositions of shallow levels in arrangements in $\mathbb{R}^3$. It follows from the analysis in that paper that the complexity of the decomposition, for any $m$ objects of $Q$, is still $O^*(m^2)$. The general machinery given above therefore implies the following result:

**Theorem 4.3.3** Let $P$ be a set of $n$ points in $\mathbb{R}^3$, and let $\Gamma$ be a family of convex ranges of constant description complexity. Then one can construct, in near linear time, a data structure of linear size so that, for any range $\gamma \in \Gamma$, it can report the points of $P$ in the complement of $\gamma$, in $O^*(n^{1/2}) + O(k)$ time, where $k$ is the query output size.

### 4.3.1 Farthest point from a convex shape

A useful application of the data structure of Theorem 4.3.2 is to farthest point queries. In such a problem we are given a set $P$ of $n$ points in $\mathbb{R}^3$, and wish to preprocess it, in near-linear time, into a data structure of linear size, so that, given a convex query object $o$ (from some fixed class of objects with constant description complexity), we can efficiently find the point of $P$ farthest from $o$.

We solve this problem using parametric searching (see Section 1.1.3 and [61]). The corresponding decision problem is: Given the query object $o$ and a distance $\rho$, determine whether the Minkowski sum $o \oplus B_\rho$ is full, where $B_\rho$ is the ball of radius $\rho$ centered at the origin. The smallest $\rho$ with this property is the distance to the farthest point from $o$. With an appropriate small-depth parallel implementation of this decision problem, the parametric searching also takes time $O^*(n^{1/2})$. Reporting the $k$ farthest points from $o$, for any parameter $k$, can be done in $O^*(n^{1/2}) + O(k)$ time, using a simple variant of this technique. We omit here the fairly routine further details of this application.

### 4.3.2 Computing the largest-area, largest-perimeter, and largest-height triangles

Let $P$ be a set of $n$ points in $\mathbb{R}^d$. We wish to find the triangle whose vertices belong to $P$ and whose area (respectively, perimeter, largest height) is maximal. The largest-area problem is a useful subroutine in path approximation algorithms; see Daescu and Serfling [32]. That paper presents an $O^*(n^{13/5})$-time algorithm for the 3-dimensional largest-area triangle. In $d$
dimensions, the running time of the corresponding algorithm is \( O^*(n^{3-2/(d^2/2)+1}) \). (Clearly, any of these problems is trivial to solve, in any dimension, in \( O(n^3) \) time.)

Consider first the problem of computing the largest-area triangle. In \( \mathbb{R}^3 \), our technique, without any additional enhancements, yields the improved bound \( O^*(n^{5/2}) \), using the following straightforward procedure. For each pair of points \( p_1, p_2 \in P \), we find the farthest point \( q \in P \) from the line \( \overline{p_1p_2} \), compute the area of \( \Delta p_1p_2q \), and output the largest-area triangle among those triangles. The procedure performs farthest-point queries from \( O(n^2) \) lines, for a total cost of \( O^*(n^{5/2}) \), as claimed.

We can improve this solution, using the following standard decomposition technique, to an algorithm with running time \( O^*(n^{26/11}) \). First, the approach just described performs \( M \) farthest-point queries on a set of \( N \) points in time \( O^*(MN^{1/2} + N) \), where the second term bounds the preprocessing cost of preparing the data structure.

Before continuing, we note the following technical issue. Recall that we find the farthest point from a query line \( \ell \) by drawing a cylinder \( C_\rho \) around \( \ell \), whose radius \( \rho \) is the smallest (unknown) radius for which \( C_\rho \) contains \( P \). The concrete value of \( \rho \) is found using parametric searching. In the approach that we follow now, we will execute in parallel \( O(n^2) \) different queries, each with its own \( \rho \), so care has to be taken when running the parametric search with this multitude of different unknown values of \( \rho \).

While there are several alternative solutions to this problem, we use the following one, which seems the cleanest. Let \( A > 0 \) be a fixed parameter. For each pair \( p_1, p_2 \) of distinct points of \( P \), let \( C_A(p_1p_2) \) denote the cylinder whose axis passes through \( p_1 \) and \( p_2 \) and whose radius is \( 2A/|p_1p_2| \). In the decision procedure, we specify the value of \( A \), and perform \( O(n^2) \) range fullness queries with the cylinders \( C_A(p_1p_2) \). If all of them are found to be full, then \( A \geq A^* \), where \( A^* \) is the (unknown) maximal area of a triangle spanned by \( P \); otherwise \( A < A^* \). (With a somewhat finer implementation, we can also distinguish between the cases \( A > A^* \) and \( A = A^* \); we omit the details of this refinement.)

To implement the decision procedure, we apply a duality transform, where each cylinder \( C \) in 3-space is mapped to a point \( C^* = (a, b, c, d, \rho) \), where \( (a, b, c, d) \) is some parametrization of the axis of \( C \) and \( \rho \) is its radius. In this dual parametric 5-space, a point \( p \in \mathbb{R}^3 \) is mapped to a surface \( p^* \), which is the locus of all (points representing) cylinders which contain \( p \) on their boundary. Note that the portion of space below (resp., above) \( p^* \) in the \( \rho \)-direction, consists of points dual to cylinders which do not contain (resp., contain) \( p \).

Let \( P^* = \{p^* \mid p \in P\} \). Fix some sufficiently large but constant parameter \( r_0 \), and construct a \((1/r_0)\)-cutting \( \Xi \) of \( A(P^*) \), using the vertical decomposition of a random sample of \( O(r_0 \log r_0) \) surfaces of \( P^* \) (see, e.g., [69]). As follows from [23, 53], the combinatorial complexity of \( \Xi \) is \( O^*(r_0^d) \). We distribute the \( O(n^2) \) points dual to the query cylinders among the cells of \( \Xi \), in brute force, and also find, in equally brute force, for each cell \( \tau \) the subset \( P^*_\tau \) of surfaces which cross \( \tau \). We ignore cells which fully lie below some surface of \( P^* \), because cylinders whose dual points fall in such a cell cannot be full (the decision algorithm stops as soon as such a point (cylinder) is detected). For each of the remaining
cells $\tau$, we repeat this procedure with the subset of the points dual to the surfaces in $P^*_\tau$ and with the subset of cylinders whose dual points lie in $\tau$. We keep iterating in this manner until we reach cuttings whose cells are crossed by at most $n/r$ dual surfaces, where $r$ is some (non-constant) parameter that we will shortly fix. As is easily checked, the overall number of cells in these cuttings is $O^*(r^6)$.

We then run the preceding weaker procedure on each of the resulting cells $\tau$, with the set $P^*_\tau$ of points dual to the surfaces which cross $\tau$ and with the set $C^*_\tau$ of cylinders whose dual points lie in $\tau$. Letting $m_\tau$ denote the number of these cylinders, the overall cost of the second phase of the procedure is

$$\sum_\tau O^*(m_\tau(n/r)^{1/2} + n/r) = O^*(n^2(n/r)^{1/2} + nr^5).$$

Since $r_0$ is a constant, the overall cost of the first phase is easily seen to be proportional to the overall size of the resulting subproblems, which is $O^*(n^2 + nr^5)$. Overall, the cost is thus

$$O^*(n^{5/2}/r^{1/2} + nr^5).$$

Choosing $r = n^{3/11}$, this becomes $O^*(n^{26/11})$.

Running a generic version of this decision procedure in parallel is fairly straightforward. The cuttings themselves depend only on the dual surfaces, which do not depend on $A^*$, so we can construct them in a concrete, non-parametric fashion. Locating the points dual to the query cylinders can be done in parallel, and, since $r_0$ is a constant, this takes constant parallel depth for each of the logarithmically many levels of cuttings. The second phase can also be executed in parallel in an obvious manner. Omitting the further easy details, we conclude that the overall algorithm also takes $O^*(n^{26/11})$ time.

**Largest-perimeter triangle.** The above technique can be adapted to yield efficient solutions of several problems of a similar flavor. For example, consider the problem of computing the largest-perimeter triangle among those spanned by a set $P$ of $n$ points in $\mathbb{R}^3$. Here, for each pair $p_1, p_2$ of points of $P$, and for a specified perimeter $\pi$, we construct the ellipsoid of revolution $E_{\pi}(p_1, p_2)$, whose boundary is the locus of all points $q$ satisfying $|qp_1| + |qp_2| = \pi - |p_1p_2|$. (Here, of course, we only consider pairs $p_1, p_2$ with $|p_1p_2| < \pi/2$.)

We now run $O(n^2)$ range fullness queries with these ellipsoids, and report that $\pi^* > \pi$ if at least one of these ellipsoids in not full, or $\pi^* \leq \pi$ otherwise, where $\pi^*$ is the largest perimeter. (Here too one can discriminate between $\pi^* < \pi$ and $\pi^* = \pi$; we omit the details as above.)

The efficient implementation of this procedure is carried out similar to the preceding algorithm, except that here the dual representation of our ellipsoids require six degrees of freedom, to specify the foci $p_1$ and $p_2$. Unlike the previous case, the dual surfaces $p^*$ do depend on $\pi$, so, in the generic implementation of the decision procedure we also need to
construct the various \((1/r_0)\)-cuttings in a generic, parallel manner.\(^6\) However, since \(r_0\) is a constant, this is easy to do in constant parallel depth per cutting. A \((1/r_0)\)-cutting in \(\mathbb{R}^6\) has complexity \(O^*(r_0^8)\) [23, 53]. A modified version of the preceding analysis then yields:

**Theorem 4.3.4** The largest-perimeter triangle among those spanned by a set of \(n\) points in \(\mathbb{R}^3\) can be computed in \(O^*(n^{12/5})\) time.

**Largest-height triangle.** In this variant, we wish to compute the triangle with largest height among those determined by a set \(P\) of \(n\) points in \(\mathbb{R}^3\). (That is, for each candidate triangle we take the largest of its three heights, and seek the maximum value of these largest heights.) Here, for each pair \(p_1, p_2\) of points of \(P\), and for a specified height \(h\), we construct the cylinder \(C_h(p_1, p_2)\), whose axis passes through \(p_1\) and \(p_2\) and whose radius is \(h\). We run \(O(n^2)\) range fullness queries with these cylinders, and report that \(h^* > h\) if at least one of these cylinders is not full, or \(h^* \leq h\) otherwise, where \(h^*\) is the desired largest height. (Here too one can discriminate between the cases \(h^* < h\) and \(h^* = h\).)

The efficient implementation of this procedure is carried out as above, except that here the dual representation of these cylinders require only four degrees of freedom, once \(h\) is specified. As in the preceding case, here too the surfaces of \(P^*\) depend also on \(h\), so we need a generic parallel procedure for constructing \((1/r_0)\)-cuttings for these surfaces, which however is not difficult to achieve, since \(r_0\) is a constant. We omit the simple routine details. Since a \((1/r_0)\)-cutting in \(\mathbb{R}^4\) has complexity \(O^*(r_0^4)\) [53], a modified version of the preceding analysis then yields:

**Theorem 4.3.5** The largest-height triangle among those spanned by a set of \(n\) points in \(\mathbb{R}^3\) can be computed in \(O^*(n^{16/7})\) time.

**Further extensions.** We can extend this machinery to higher dimensions, although its performance deteriorates as the dimension grows. The range fullness problem in \(\mathbb{R}^d\), for \(d \geq 4\), can be handled in much the same way as in the 3-dimensional case. When extending Claim 4.3.1, we have an intersection of \(m\) convex sets of constant description complexity in \(\mathbb{R}^d\), and we can regard the boundary of the intersection as the lower envelope of \(m\) \((d-1)\)-variate functions of constant description complexity, each representing the boundary of one of the input convex sets, in spherical coordinates about some fixed point in the intersection. The complexity of the lower envelope is \(O^*(m^{d-1})\) [68]. However, we need to decompose the region below the envelope into elementary cells, and, as already noted, the only known general-purpose technique for doing so is to decompose the entire arrangement of the graphs of the \(m\) boundary functions, and select the cells below the lower envelope. The complexity of such a decomposition is \(O^*(m^{2d-4})\) [23, 53]. This implies that \(\zeta(r) = O^*(r^{2d-4})\). The

\(^6\)We can make these surfaces independent of \(\pi\) if we add \(\pi\) as a seventh degree of freedom, but then the overall performance of the algorithm deteriorates.
rest of the analysis, including the construction of a good test set, is done in essentially the same manner. Hence, using the machinery of the previous section, we obtain:

**Theorem 4.3.6** Let \( P \) be a set of \( n \) points in \( \mathbb{R}^d \), for \( d \geq 4 \), and let \( \Gamma \) be a family of convex ranges of constant description complexity. Then one can construct, in near linear time, a data structure of linear size so that, for any range \( \gamma \in \Gamma \), it can determine, in \( O^*(n^{1-1/(2d-4)}) \), whether \( \gamma \) is full.

**Finding the largest-area triangle in \( \mathbb{R}^d \).** Let \( P \) be a set of \( n \) points in \( \mathbb{R}^d \), for \( d \geq 4 \), and consider the problem of finding the largest-area triangle spanned by \( P \). We apply the same method as in the 3-dimensional case, whose main component is a decision procedure which tests \( O(n^2) \) cylinders for fullness. A cylinder (with a line as an axis) in \( \mathbb{R}^d \) has \( 2d - 1 \) degrees of freedom, so the dual representation of our \( O(n^2) \) cylinders is as points in \( \mathbb{R}^{2d-1} \). The best known bound on the complexity of a \( (1/r) \)-cutting in this space is \( O^*(r^{2(2d-1)-4}) = O^*(r^{4d-6}) \). Applying this bound and the bound in Theorem 4.3.6, the overall cost of the decision procedure is

\[
O^* \left( n^2 \left( \frac{n}{r} \right)^{1-1/(2d-4)} + nr^{4d-7} \right).
\]

Optimizing the value of \( r \), and applying parametric searching, we get an algorithm for the maximum-area triangle in \( \mathbb{R}^d \) with running time

\[
O^* \left( n^{1+\frac{(4d-9)(4d-7)}{(2d-9)(2d-4)}} \right).
\]

(The exponent in this bound is always smaller than 3, for \( d \geq 3 \); it approaches 3 as \( d \to \infty \). Moreover, for \( d \geq 4 \), this bound is smaller than the bound \( O^*(n^{3-2/(\lfloor d/2 \rfloor + 1)}) \) of [32].) We can extend the other problems (largest-perimeter or largest-height triangles) in a similar manner, and can also obtain algorithms for solving higher-dimensional variants, such as computing the largest-volume tetrahedron or higher-dimensional simplices. We omit the straightforward but tedious analysis, and the resulting cumbersome-looking but non-trivial bounds.

### 4.4 Ray Shooting Amid Balls in 3-Space

Let \( B \) be a set of \( n \) balls in 3-space. We show how to preprocess \( B \) in near-linear time into a data structure of linear size, so that, given a query ray \( \rho \), the first ball that \( \rho \) hits can be computed in \( O^*(n^{2/3}) \) time, improving the general bound \( O^*(n^{3/4}) \) mentioned in the introduction. As already noted, we use the parametric-searching technique of Agarwal and Matoušek [4], which reduces the problem to that of efficiently testing whether a query segment \( s = qz \subset \rho \) intersects any ball in \( B \), where \( q \) is the origin of \( \rho \) and \( z \) is a parametric point along \( \rho \).
Parametric representation of balls and segments. We use a parametric 4-dimensional space, in which balls in 3-space are represented by points, so that a ball with center at \((a, b, c)\) and radius \(r\) is mapped to the point \((a, b, c, r)\). A segment \(e\), or for that matter, any closed nonempty set \(K \subset \mathbb{R}^3\) of constant description complexity, is mapped to a surface \(\sigma_K\), which is the locus of all points representing balls that touch \(K\) but are openly disjoint from \(K\). By construction, \(\sigma_K\) is the graph of a totally defined continuous and non-negative trivariate function \(r = \sigma_K(a, b, c)\), which is semialgebraic of constant description complexity. Moreover, points below (resp., above) \(\sigma_K\) represent balls which are disjoint from \(K\) (resp., intersect \(K\)).

Moreover, for any such set \(K\), \(\sigma_K(q)\) is, by definition, the (Euclidean) distance of \(q\) from \(K\). Hence, given a collection \(K = \{K_1, K_2, \ldots, K_m\}\) of \(m\) sets, the minimization diagram of the surfaces \(\sigma_{K_1}, \ldots, \sigma_{K_m}\) (that is, the projection onto the 3-space \(r = 0\) of the lower envelope of these surfaces) is the nearest-neighbor Voronoi diagram of \(K\). We use this property later on, in deriving a sharp bound on the resulting function \(\zeta(\cdot)\).

Building a test set for segment emptiness. Here we use the general recipe for constructing good test sets, which covers each empty segment \(e\) by a fairly complex “canonical” empty region \(K\), which has nonetheless constant description complexity. In parametric 4-space, each such region \(K\) is mapped to the upper half-space above the corresponding surface \(\sigma_K\); this portion of space represents the set of all balls that intersect \(K\). The complement of the union of \(m\) such ranges is the portion of 4-space below the lower envelope of the corresponding surfaces \(\sigma_{K_i}\). Using the connection between this envelope and the Voronoi diagram of the \(K_i\)’s, we are able to decompose (the diagram and thus) the complement of the union into \(\zeta(m) = \Omega^*(m^3)\) elementary cells.

In more detail, the construction proceeds as follows. We start by choosing a random sample \(N\) of \(O(r \log r)\) balls of \(B\), and then construct a test set \(Q\) for empty segment ranges, with respect to the balls of \(N\). While we do not have a clean, explicit geometric definition of these ranges, they will satisfy, as above, all the four requirements from a good test set. Also, we spell out the adaptation of the general recipe to the present scenario, to help the reader see through one concrete application of the general recipe.

Specifically, we move to a dual space, in which segments in 3-space are represented as points. Segments in 3-space have six degrees of freedom; for example, we can represent a segment by the coordinates of its two endpoints. The dual space is therefore 6-dimensional. Each ball \(B \in N\) is mapped to a surface \(B^*\), which is the locus of all points representing segments which touch \(\partial B\) but do not penetrate into its interior; that is, either they are tangent to \(B\), at a point in their relative interior, or they have an endpoint on \(\partial B\) but are openly disjoint from \(B\).

Let \(N^*\) denote the collection of the surfaces dual to the balls of \(N\). Construct a \((1/r)\)-cutting of \(A(N^*)\), which consists of \(O^*(r^8)\) elementary cells \([23, 53]\). Each cell \(\tau\) has the property that all points in \(\tau\) represent segments which meet a fixed set of balls from among
the balls in $N$ and avoid all other balls of $N$; the set depends only on $\tau$.

For each cell $\tau$ whose corresponding set of balls is empty, we define $K_\tau$ to be the union, in 3-space, of all segments $e$ whose dual points lie in $\tau$. Since $\tau$ is an elementary cell, $K_\tau$ is a semialgebraic set of constant description complexity (see, e.g., [20]). Moreover, $K_\tau$ is an $N$-empty region, in the sense that it is openly disjoint from all the balls in $N$.

Since we have to work in parametric 4-space, we map each region $K_\tau$ of the above kind into a range $\gamma_\tau$ in 4-space, which is the locus of all (points representing) balls which intersect $K_\tau$. As discussed above, $\gamma_\tau$ is the upper half-space bounded by the graph $\sigma_{K_\tau}$ of the distance function from points in $R^3$ to $K_\tau$.

We define the desired test set $Q$ to consist of all the ranges $\gamma_\tau$, as just defined, and argue that, with high probability, $Q$ indeed satisfies all four properties required from a good test set: (a) Compactness: $|Q| = O^*(r^8)$, so its size is small. (b) Shallowness: With high probability, each range in $Q$ is $(n/r)$-shallow, since it does not contain any point representing a ball in $N$ (and we assume that the sample $N$ does indeed have this $\varepsilon$-net property, which makes all the ranges in $Q$ $(n/r)$-shallow). (c) Containment: Each empty segment is also $N$-empty, so its dual point lies in some cell $\tau$ of the cutting, whose associated subset of balls is empty. By construction, we have $e \subset K_\tau$. That is, any ball intersecting $e$ also intersects $K_\tau$, so the range in 4-space that $e$ defines is contained in $\gamma_\tau$, i.e., in a single range of $Q$. (d) Efficiency: As we argue next, the complement of the union of any $m$ ranges in $Q$ can be decomposed into $O^*(m^3)$ elementary cells.

The proof of (d) proceeds as follows. The complement of the union of $m$ ranges, $\gamma_{\tau_1}, \ldots, \gamma_{\tau_m}$, is the region below the lower envelope of the corresponding surfaces $\sigma_{K_{\tau_1}}, \ldots, \sigma_{K_{\tau_m}}$. To decompose this region, it suffices to produce a decomposition of the 3-dimensional minimization diagram of these surfaces, and extend each of the resulting cells into a semi-unbounded vertical prism, whose “ceiling” lies on the envelope.

The combinatorial complexity of the minimization diagram of a collection $\mathcal{K} = \{K_{\tau_1}, \ldots, K_{\tau_m}\}$ of $m$ trivariate functions of constant description complexity is $\approx O^*(m^3)$ [69]. Moreover, as noted above, the minimization diagram is the Euclidean nearest-neighbor Voronoi diagram of $\mathcal{K}$.

We can decompose each cell $V_i = V(K_{\tau_i})$ of the diagram (or, more precisely, the portion of the cell outside the union of the $K_{\tau_i}$’s) using its star-shapedness with respect to its “site” $K_{\tau_i}$; that is, for any point $p \in V(K_{\tau_i})$, the segment connecting $p$ to its nearest point on $K_{\tau_i}$ is fully contained in $V(K_{\tau_i})$. As is easy to verify, this property holds regardless of the shape, or intersection pattern, of the regions in $\mathcal{K}$. We first decompose the 2-dimensional faces bounding $V_i$ into elementary cells, using, e.g., an appropriate variant of 2-dimensional vertical decomposition, and then take each such cell $\phi_0$ and extend it to a cell $\phi$, which is the union of all segments, each connecting a point in $\phi_0$ to its nearest point on $K_{\tau_i}$. The resulting cells, obtained by applying this decomposition to all cells of the diagram,
form a decomposition of the portion of the diagram outside the union of the $K_{\tau_i}$’s, into a total of $O^*(m^3)$ elementary cells, as desired. The union of the $K_{\tau_i}$’s themselves, being a subcollection of cells of a 3-dimensional arrangement of $m$ regions of constant description complexity, can also be decomposed into $O^*(m^3)$ cells, using standard results on vertical decomposition in three dimensions [69].

Using Lemma 4.2.4 and the machinery of Section 4.2, in conjunction with the parametric searching technique of [4], we thus obtain the following theorem.

**Theorem 4.4.1** Ray shooting amid $n$ balls in 3-space can be performed in $O^*(n^{2/3})$ time, using a data structure of $O(n)$ size, which can be constructed in $O^*(n)$ time.

**Remark:** In the preceding description, we only considered *empty* ranges. If desired, we can extend the analysis to obtain a data structure which also supports “reporting queries”, in which we want to report the first $k$ balls hit by a query ray. We omit the details of this simple extension, which follows the same approach as for reporting outliers in Section 4.3. (As in Section 4.3, we only consider a small number of levels above the envelope, and the analysis can be extended to show that the number of cells in the resulting decomposition of any number $m$ of ranges is still $O^*(m^3)$.)

Note that the machinery described above answers general emptiness queries amid balls in 3-space, and does not require the ranges to be line segments. We can therefore consider shooting amid balls along other kinds of arcs. For example, using parametric searching with a similar construction, where the ranges are arcs of vertical parabolas, we can answer stone throwing queries amid balls in 3-space in $O^*(n^{2/3})$ time using linear storage. See Chapter 2 for a definition of the stone throwing problem and related results.

### 4.5 Range Emptiness Searching and Reporting in the Plane

#### 4.5.1 Fat triangle reporting and emptiness searching

Let $\alpha > 0$ be a fixed constant, and let $P$ be a set of $n$ points in the plane. We present a procedure which preprocesses $P$, in $O^*(n)$ time, into a data structure of linear size, so that, given an $\alpha$-fat query triangle $\Delta$ (which, as we recall, is a triangle all of whose angles are at least $\alpha$), we can determine in $O^*(1)$ time whether $\Delta \cap P = \emptyset$, or report in $O^*(1) + O(k)$ time the points of $P$ in $\Delta$, where $k = |P \cap \Delta|$.

To do so, we need to construct a good test set $Q$. We use the following “canonization” process (an ad-hoc process, not following the general recipe of Section 4.2). As above, we apply the construction to a random sample $N$ of $O(r \log r)$ points of $P$. For simplicity, we first show how to canonize *empty* triangles, and then extend the construction to shallow triangles. (As before, the first part suffices for emptiness searching, whereas the second
part is needed for reporting queries.) Let $\Delta$ be an $\alpha$-fat empty triangle, which is then also $N$-empty. As shown in [60], $\Delta$ is the union of three semi-canonical triangles, each of which has two edges of fixed orientations taken from the set of $O(1/\alpha)$ orientations \[\{ja/2 \mid j = 0, 1, \ldots, \lfloor 4\pi/\alpha \rfloor\},\] and its third edge is an edge of $\Delta$; see Figure 4.2. To answer an emptiness query with $\Delta$, we test for emptiness each of these three covering triangles. A reporting query is handled similarly. Each point is reported at most three times, and eliminating these repetitions is routine and easy to do.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.2}
\caption{Covering a triangle by three semi-canonical triangles.}
\end{figure}

Hence, in what follows we may assume that $\Delta$ is semi-canonical and $N$-empty. We expand $\Delta$ homothetically, keeping one vertex fixed and translating the opposite side away, until it hits a point $q_1$ of $N$. We then expand the new triangle homothetically from a second vertex, until the opposite side hits a second point $q_2$ of $N$, and then apply a similar expansion from the third vertex, making the third edge of the triangle touch a third point $q_3$ of $N$. We end up with an $N$-empty triangle $\Delta'$, homothetic to, and containing, $\Delta$, each of whose sides passes through one of the points $q_1, q_2, q_3 \in N$. See Figure 4.3. (It is possible that some of these expansions never hit a point of $N$, so we may end up with an unbounded wedge or half-plane instead of a triangle. Also, the points $q_1, q_2, q_3$ need not be distinct.)

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure4.3}
\caption{The first step in canonizing an empty triangle.}
\end{figure}

We turn the side containing $q_1$ (whose orientation is not fixed) clockwise and counterclockwise about $q_1$, keeping its endpoints on the lines containing the other two sides, until we reach an orientation in $D$, or until we hit another point of $N$ (which could also be one
of the points \( q_2, q_3 \), whichever comes first. Each of the new sides forms, with the two lines containing the two other sides, a new (openly) \( N \)-empty \((\alpha/2)\)-fat triangle; the union of these two triangles covers \( \Delta' \). See Figure 4.4.

Each of these new triangles is \((\alpha/2)\)-fat, openly \( N \)-empty, has two sides at fixed orientations, each passing through a point of \( N \), and a third side which either has a fixed orientation and passes through a point of \( N \) or passes through two points of \( N \). Since \(|D| = O(1/\alpha) = O(1)|\), it follows that the overall number of these canonical covering triangles is \( O((r \log r)^4) = O^*(r^4) \). (We omit the easy extensions of this step to handle unbounded wedges or half-planes, or the cases where the points \( q_i \), or some of the newly encountered points, lie at vertices of the respective triangles.)

We take \( Q \) to be the collection of these canonical triangles, and argue that \( Q \) indeed satisfies the properties of a good test set (with high probability): (a) **Compactness:** \(|Q| = O^*(r^4)|\), so its size is small. (b) **Shallowness:** With high probability, each range in \( Q \) is \((n/r)\)-shallow (and, as usual, we assume that this property does indeed hold for our choice of \( N \)). (c) **Containment:** By construction, each (semi-canonical) \( \alpha \)-fat empty triangle is contained in the union of at most six triangles in \( Q \). (d) **Efficiency:** Being \((\alpha/2)\)-fat, the union of any \( m \) triangles in \( Q \) has complexity \( O(m \log^* m) \) (this is a very recent improvement over the “classical” \( O(m \log \log m) \) bound of [60]; see [43] and [13]). Hence the associated function \( \zeta \) satisfies \( \zeta(m) = O(m \log^* m) \), and \( \zeta^{-1}(r) = O\left(\frac{r}{\log \log r}\right) \). Combined with Lemma 4.2.4 and the machinery of Section 4.2, we get the following recursion for the query time:

\[
Q(n) \leq D(r) + O(\log^* r)Q(n/r),
\]

where \( D(r) \) bounds the time needed to find which of the \( O(r) \) cells of the partition are crossed by \( \Delta \) and which are (fully) contained in \( \Delta \). Choosing \( r = O(1) \) and accepting a brute force solution with \( D(r) = O(r) \), we get the following theorem:
Theorem 4.5.1 One can preprocess a set $P$ of $n$ points in the plane, in near-linear time, into a data structure of linear size, so that, for any query $\alpha$-fat triangle $\Delta$, one can determine, in $O^*(1)$ time, whether $\Delta \cap P = \emptyset$.

Discussion. There is an alternative way of testing a semi-canonical fat triangle $\Delta$ (and thus any fat triangle) for emptiness. That is, we can construct a 3-level data structure, where the first two levels are range trees [36] on the points, sorted in the respective orders of their projections on lines orthogonal to the fixed orientations of two of the sides of $\Delta$. Querying the first two levels with $\Delta$, we obtain $O(\log^2 n)$ canonical subsets of $P$, each fully contained in the wedge spanned by the two fixed-orientation sides of $\Delta$. We then query each of these subsets $P_w$ with the line $\ell$ supporting the third edge of $\Delta$, to determine whether all the points of $P_w$ lie in the half-plane bounded by $\ell$ and disjoint from $\Delta$, which is easy to do in $O(\log n)$ time and $O(|P_w|)$ storage. Overall we obtain a data structure that uses $O(n \log^2 n)$ storage and can be constructed in $O(n \log^2 n)$ time, so that an emptiness query takes $O(\log^3 n)$ time.

Compared with the data structure of Theorem 4.5.1, this alternative solution has faster query time (choosing $r$ to be a constant yields $Q(n) = O(n^\varepsilon)$), where $\varepsilon > 0$ can be made arbitrarily small as we increase $r$) but it uses more storage.

Reporting points in fat triangles. We can extend the technique given above to solve the problem of reporting the points of $P$ that lie inside any query fat triangle. For this, we need to construct a test set that will be good for shallow ranges and not just for empty ones. Using Theorem 1.1.2, we construct (by random sampling) a shallow $(1/r)$-net $N \subseteq P$ of size $O(r \log r)$. We next canonize every $(n/r)$-shallow $\alpha$-fat triangle $\Delta$, by the same canonization process used above, with respect to the set $N$. Note that each of the resulting canonical triangles contains (in its interior) the same subset of $N$ as $\Delta$ does. By the properties of shallow $(1/r)$-nets, since $|\Delta \cap P| \leq n/r$, we have $|\Delta \cap N| = O(\log r)$, so all the resulting canonical triangles are $(c \log r)$-shallow with respect to $N$, for some absolute constant $c$. Again, since $N$ is a shallow $(1/r)$-net, all the canonical triangles are $(c' n/r)$-shallow with respect to $P$, for another absolute constant $c'$. Hence, the resulting collection $Q$ of canonical triangles is a good test set for all shallow fat triangles, and we can apply the machinery of Section 4.2 to obtain a data structure of linear size, which can be constructed in near-linear time, and which can perform reporting queries in fat triangles in time $O^*(1) + O(k)$, where $k$ is the output size of the query.

4.5.2 Range emptiness searching with semidisks and circular caps

The motivation for studying this problem comes from the following problem, addressed in [31]. We are given a set $P$ of $n$ points in the plane, and wish to preprocess it into a data structure of linear size, so that, given a query point $q$ and a query line $\ell$, one can
quickly find the point of $P$ closest to $q$ and lying, say, above $\ell$. In the original problem, as formulated in [31], one also assumes that $q$ lies on $\ell$, but we will consider, and solve, the more general version of the problem, where $q$ also lies above $\ell$.

The standard approach (e.g., as in [5]) yields a solution with linear storage and near-linear preprocessing, and query time $O^*(n^{1/2})$. We present a solution with query time $O^*(1)$.

Using parametric searching [61], the problem reduces to that of testing whether the intersection of a disk of radius $\rho$ centered at $q$ with the half-plane $\ell^+$ above $\ell$ is $P$-empty. The resulting range is a circular cap larger than a semidisk (or exactly a semidisk if $q$ lies on $\ell$). Again, the main task is to construct a good test set $Q$ for such ranges, which we do by using an ad-hoc canonization process, which covers each empty circular cap by $O(1)$ canonical caps, which satisfy the properties of a good test set; in particular, we will have $\zeta(m) = O^*(m)$. (As before, we consider here only the case of emptiness detection, and will consider the reporting problem later.)

To construct a test set $Q$ we choose a random sample $N$ of $O(r \log r)$ points of $P$, with a sufficiently large constant of proportionality, and build a set of canonical empty ranges with respect to $N$. Let $C = C_{c,\rho,\ell}$ be a given circular cap (larger than a semidisk) with center $c$, radius $\rho$, and chord supported by a line $\ell$. We first translate $\ell$ in the direction which enlarges the cap, until either its portion within the disk $D$ of the cap touches a point of $N$, or $\ell$ leaves $D$. See Figure 4.5(left). In the latter case, $C$ is contained in a complete $N$-empty disk, and it is fairly easy to show that such a disk is contained in the union of at most three canonical $N$-empty disks, each passing through three points of $N$ or through two diametrically opposite points of $N$; there are at most $O^*(r)$ such disks, since they are all Delaunay disks of $N$.

![Figure 4.5](image-url)  
Figure 4.5: The first steps in canonizing an empty cap.

Suppose then that the new chord (we continue to denote its line as $\ell$) passes through a point $q$ of $N$, as in Figure 4.5 (left). Let $D$ be a set of $O(1)$ canonical orientations, uniformly spaced and sufficiently dense along the unit circle. Rotate $\ell$ about $q$ in both clockwise and counterclockwise directions, until we reach one of the two following events: (i) the orientation of $\ell$ belongs to $D$; or (ii) the portion of $\ell$ within $D$ touches another point of $N$. In either case, the two new lines, call them $\ell_1, \ell_2$, become canonical—there are only
such possible lines. Note that our original cap $C$ is contained in the union $C_1 \cup C_2$, where $C_1 = C_{c,\rho,\ell_1}$ and $C_2 = C_{c,\rho,\ell_2}$. Moreover, although the new caps need no longer be larger than a semidisk, they are not much smaller—this is an easy exercise in elementary geometry. See Figure 4.5(right).

We next canonize the disk of $C$ (which is also the disk of $C_1$ and $C_2$). Fix one of the new caps, say $C_1$. Expand $C_1$ from the center $c$, keeping the line $\ell_1$ fixed, until we hit a point $q_1$ of $N$ (lying in $\ell_1^+$); see Figure 4.6(left). If the center $c$ lies in $\ell_1^+$ then we move $c$ parallel to $\ell_1$ in both directions, again keeping $\ell_1$ itself fixed and keeping the circle pass through $q_1$, until we obtain two circular caps, each passing through $q_1$ and through a second point of $N$ (if we do not hit a second point, we reach a quadrant, bounded by $\ell$ and by the line orthogonal to $\ell$ through $q_1$). The union of the two new circular caps (each larger than a semidisk) covers $C_1$. See Figure 4.6(right).

If $c$ lies in $\ell_1^-$, we move it along the two rays connecting it to the endpoints $u_0, v_0$ of the chord of $C_1$ (supported by $\ell_1$); see Figure 4.7. As before, each of the motions stops when the circle hits another point of $N$ in $\ell_1^+$, or when the motion reaches $u_0$ or $v_0$. We claim that $C_1$ is contained in the union of the two resulting caps. Indeed, let $u$ and $v$ denote the locations of the center at the two stopping placements. We need to show that, for any point $b \in C_1$ we have either $|bu| \leq |q_1u|$ or $|bv| \leq |q_1v|$. If both inequalities did not hold, then both $u$ and $v$ would have to lie on the side of the perpendicular bisector of $q_1b$ containing $q_1$. This is easily seen to imply that $c$ must also lie on that side, which however is impossible (because $|bc| \leq |q_1c|$).

Next, we take one of these latter caps, $C'$, whose bounding circle passes through $q_1$ and through a second point $q_2$ of $N \cap \ell_1^+$, and move its center along the bisector of $q_1q_2$ in both directions, keeping the bounding circle touch $q_1$ and $q_2$, and still keeping the line $\ell_1$ supporting the chord fixed. We stop when the first of these events takes place: (i) The center reaches $\ell_1$, in which case the cap becomes a semidisk (this can happen in only one of the moving directions). (ii) The center reaches the midpoint of $q_1q_2$. (iii) The bounding circle touches a third point of $N \cap \ell_1^+$. (iv) The central angle of the chord along $\ell_1$ is equal to some fixed positive angle $\beta > 0$. The union of the two new caps covers $C'$. (It is possible that during the motion the moving circle becomes tangent to $\ell_1$, and then leaves it, in which case the corresponding final cap is a full disk.)
Similarly, if the center of \( C' \) lies on \( \ell_1 \) (which can happen when the motion in the preceding canonization step reaches \( v_0 \) or \( u_0 \)), then we canonize its disk by translating the center to the left and to the right along \( l_1 \) until the bounding circle touches another point of \( N \cap \ell_1^+ \), exactly as in the preceding case (shown in Figure 4.6(right)).

Let \( C'' \) be one of the four new caps. In all cases \( C'' \) is canonical: For the first kind of caps, the stopping condition that defines \( C'' \) is (ii) or (iii) then either the circle bounding \( C'' \) passes through three points of \( N \) or it passes through two diametrically opposite points of \( N \). There are a total of \( O^*(r^3) \) such circles, and since \( C'' \) is obtained (in a unique manner) by the interaction of one of these circles and one of the \( O^*(r^2) \) canonical chord-lines, there is a total of \( O^*(r^5) \) such caps. If the stopping condition is (i), the cap is also canonical, because the center of the containing disk is the intersection point of a bisector of two points of \( N \) with one of the \( O^*(r^2) \) canonical chord-lines, so there is a total of \( O^*(r^4) \) such caps. In the case of condition (iv), there are only \( O^*(r^2) \) pairs \( q_1, q_2 \) and \( O^*(r^2) \) canonical chords, for a total of \( O^*(r^4) \) caps. Similar reasoning shows that the caps resulting in the second case are also canonical, and their number is \( O^*(r^4) \).

We take the test set \( Q \) to consist of all the caps of the final forms, and argue that, with high probability, it satisfies the properties of a good test set: (a) Compactness: \(|Q| = O^*(r^5)\), so its size is small. (b) Shallowness: With high probability, each range in \( Q \) is \((n/r)\)-shallow (and we assume that this property does indeed hold). (c) Containment: Each empty cap is also \( N \)-empty, so, by the above canonization process, it is contained in the union of \( O(1) \) caps of \( Q \). (d) Efficiency: Each cap \( C \in Q \) is locally \( \gamma \)-fat, for an appropriate fixed constant \( \gamma > 0 \), in the terminology of [34], meaning that for each point \( p \in C \) and a disk \( D \) centered at \( p \) and not fully containing \( C \), we have \( \text{area}(D \cap C) \geq \gamma \cdot \text{area}(D) \). In addition, the boundaries of any two ranges in \( Q \) (or of any two circular caps, for that matter) intersect in at most four points, as is easily checked. As follows from the recent analysis of Aronov et al. [13] (see also [34]), the complexity of the union of any \( m \) ranges in \( Q \) is \( m2^{O(\log^r m)} = O^*(m) \). Hence, the complement of the union of any \( m \) ranges in \( Q \) can be decomposed into \( O^*(m) \) elementary cells, making \( \zeta(m) = O^*(m) \).

In conclusion, we obtain:

**Theorem 4.5.2** Let \( P \) be a set of \( n \) points in the plane. We can preprocess \( P \), in near-
linear time, into a data structure of linear size, so that, for any query circular cap $C$, larger than or equal to a semidisk, we can test whether $C \cap P$ is empty in $O^*(1)$ time.

Combining Theorem 4.5.2 with parametric searching, we obtain:

**Corollary 4.5.3** Let $P$ be a set of $n$ points in the plane. We can preprocess $P$, in near-linear time, into a data structure of linear size, so that, for any query half-plane $\ell^+$ and point $q \in \ell^+$, we can find, in $O^*(1)$ time, the point in $P \cap \ell^+$ nearest to $q$.

**Remark.** The machinery developed in this section also applies to smaller circular caps, as long as they are not too small. Formally, if the central angle of each cap is at least some fixed constant $\alpha > 0$, the same technique holds, so we can test emptiness of such ranges in $O^*(1)$ time, using a data structure which requires $O(n)$ storage and $O^*(n)$ preprocessing. Thus Theorem 4.5.2 carries over to this scenario, but Corollary 4.5.3 does not, because we have no control over the “fatness” of the cap, as the disk shrinks or expands, when the center of the disk lies in $\ell^-$, and once the canonical caps become too thin, the complexity of their union may become quadratic.

The technique described here can be used to answer queries that seek the farthest point above a line, where we want to preprocess a set $P$ of $n$ points, so that, given a line $\ell$ and a point $p$ on (or above) $\ell$, we can find the point $q \in P$ that lies above $\ell$ and is farthest from $p$. Similarly to the problem of finding the farthest point from a convex shape in 3-space, studied in Section 4.3.1, the test set created here is a set of semidisks, each containing at least $n - n/r$ points of $P$ for some constant $r > 0$. The function $\zeta(r)$ we consider here bounds the number of elementary cells in the decomposition of the intersection of $r$ semidisks. It is easy to verify that the complexity of the decomposition of the intersection of $r$ semidisks is $O(\lambda_4(r))$. This immediately leads to a linear size data structure that can answer queries for the farthest point above a line in $O^*(1)$ time.

**Reporting points in semidisks and circular caps.** As in the case of fat triangles, we can extend the technique to answer efficiently range reporting queries in semidisks or in sufficiently large circular caps. We use the same canonization process, with respect to a random sample $N$ of size $O(r \log r)$ which is a shallow $(1/r)$-net, and argue, exactly as in the case of fat triangles, that the resulting collection of canonical caps is a good test set for shallow semidisk or larger cap ranges. Applying the machinery of Section 4.2, we obtain a data structure of linear size, which can be constructed in near-linear time, and which can perform reporting queries in semidisks or larger caps, in time $O^*(1) + O(k)$, where $k$ is the output size of the query.
4.6 Conclusions

In this chapter we have presented a general approach to efficient range emptiness searching with semialgebraic ranges, and have applied it to several specific emptiness searching and ray shooting problems. The present study resolves and overcomes the technical problems encountered in our earlier study [71], and presents more applications of the technique.

Clearly, there are many other applications of the new machinery, and an obvious direction for further research is to “dig them up”. In each such problem, the main step would be to design a good test set, with associated function $\zeta(\cdot)$ as small as possible, using either the general recipe or an appropriate ad-hoc analysis. Many specific instances of this step are likely to generate interesting (and often hard) combinatorial questions. For example, as already mentioned earlier, we still do not know whether the complement of the union of $n$ (congruent) cylinders in $\mathbb{R}^3$ can be decomposed into $O^*(n^2)$ elementary cells. This subproblem arises in the design of a data structure for a set $P$ on $n$ points in $\mathbb{R}^3$, which supports efficient queries where we specify a line $\ell$ and want to find the point of $P$ nearest to $\ell$. 
Chapter 5

Approximate Semialgebraic Range Counting

Only by counting could humans demonstrate their independence of computers.
(The Hitchhiker’s Guide to the Galaxy)

Ford carried on counting quietly. This is about the most aggressive thing you
can do to a computer, the equivalent of going up to a human being and saying
Blood... blood... blood... blood...
(The Hitchhiker’s Guide to the Galaxy)

5.1 Overview

Given a set $P$ of $n$ points in $\mathbb{R}^d$, a set $\Gamma$ of semialgebraic ranges of constant description
complexity, and a parameter $\varepsilon > 0$, the approximate range counting problem is to preprocess
$P$ into a data structure such that, for any query range $\gamma \in \Gamma$, we can efficiently compute
an approximate count $t_\gamma$ which satisfies

$$(1 - \varepsilon)|P \cap \gamma| \leq t_\gamma \leq (1 + \varepsilon)|P \cap \gamma|.$$ 

As in Chapters 2 and 4, we only consider the case where the size of the data structure is
to be (almost) linear, and the goal is to find solutions with small query time.

The problem has been studied in several recent papers [15, 16, 17, 51], for the special
case where $P$ is a set of points in $\mathbb{R}^d$ and $\Gamma$ is the collection of half-spaces (bounded by
hyperplanes). A variety of solutions, with near-linear storage, were derived; in all of them,
the dependence of the query cost on $n$ is close to $n^{1-1/\lfloor d/2 \rfloor}$, which, as reviewed in the
introduction, is roughly the same as the cost of half-space range emptiness queries, or the
overhead cost of half-space range reporting queries [57].

The fact that the approximate range counting problem is closely related to range empti-
ness comes as no surprise, because, when $P \cap \gamma = \emptyset$, the approximate count $t_\gamma$ must be 0,
so range emptiness is a special case of approximate range counting. The goal is therefore to derive solutions that are comparable, in their dependence on $n$, with those that solve emptiness (or reporting) queries. As just noted, this has been accomplished for the case of half-spaces. In this chapter we extend this technique to the general semialgebraic case, exploiting the results that we have obtained in Chapter 4.

We present two solutions for this problem. The first solution, given in Section 5.2, plugs our algorithm for emptiness queries as a black box into the machinery of Aronov and Har-Peled [15], which effectively performs a binary search over the unknown size of the query output. The second solution is based on another recent technique of Aronov and Sharir [17]. Their technique, originally designed for half-space ranges, constructs a modified partition tree for half-space emptiness or reporting, where some extra data is stored at each node of the tree. In Section 5.3 we modify their technique to handle general semialgebraic ranges. In Section 5.4 we consider a variant where each point $p \in P$ has an associated weight $w(p)$, so that, given a range $\gamma$ and a parameter $k > 1$, we wish to report efficiently the $k$ points in $P \cap \gamma$ that have the largest weights.

5.2 First Approach

The simplest solution is to adapt the technique of Aronov and Har-Peled [15], which uses a procedure for answering range emptiness queries as a “black box”. Specifically, suppose we have a data structure, $D(P')$, for any set $P'$ of $n'$ points, which can be constructed in $T(n')$ time, uses $S(n')$ storage, and can determine whether a query range $\gamma \in \Gamma$ is $P'$-empty in $Q(n')$ time. Using such a black box, Aronov and Har-Peled show how to construct an approximate range counting data structure for a set $P$ of $n$ input points and a specified error parameter $\varepsilon$, using $O \left( \varepsilon^{3} + \sum_{i=1}^{\lceil \frac{1}{\varepsilon} \rceil} 1/i^{2-2} \right) S(n) \log n$ storage and $O \left( \varepsilon^{3} + \sum_{i=1}^{\lceil \frac{1}{\varepsilon} \rceil} 1/i^{2-2} \right) T(n) \log n$ preprocessing, where $\lambda \geq 1$ is some constant for which $S(n/r) = O(S(n)/r^\lambda)$ and $T(n/r) = O(T(n)/r^\lambda)$, for any $r > 1$. Given a range $\gamma \in \Gamma$, this structure returns, in $O(\varepsilon^{-2} Q(n) \log n)$ time, an approximate count $t_\gamma$, satisfying $(1 - \varepsilon)|\gamma \cap P| \leq t_\gamma \leq |\gamma \cap P|$.

The intuition behind this approach is that a range $\gamma$, containing $m$ points of $P$, is expected to contain $mr/n$ points in a random sample of $P$ of size $r$, and most likely it contains no points in a sample of size significantly smaller than $n/m$. The algorithm of [15] then guesses the value of $m$ (up to a factor of $1 + \varepsilon$), sets $r$ to be an appropriate multiple of $n/m$, and draws many (specifically, $O(\varepsilon^{-2} \log n)$) random samples of $P$ of size $r$. If $\gamma$ is empty (resp., nonempty) for many of the samples then, with high probability, the guess for $m$ is too large (resp., too small). When we cannot decide either way, we are at the correct value of $m$ (up to a relative error of $\varepsilon$). The actual details of the search are somewhat more contrived; see [15].

Plugging our emptiness data structures into the machinery of [15], we therefore obtain
the following results. In all these applications we can take \( \lambda = 1 \), because of the near-linear size of our data structures, so, using the above bounds, the overall data structure requires \( O(\varepsilon^{-2}S(n) \log n) \) storage and \( O(\varepsilon^{-2}T(n) \log n) \) preprocessing, where \( S(n) \) and \( T(n) \) are the storage and preprocessing bounds for our data structures. In the bounds stated in the corollaries, the \( O^\ast(\cdot) \) notation refers to the dependence on \( n \) only.

**Corollary 5.2.1** Let \( P \) be a set of \( n \) points in the plane, and let \( \alpha, \varepsilon \) be given positive parameters. Then we can preprocess \( P \) into a data structure of size \( O^\ast(\varepsilon^{-2}n) \), such that, for any \( \alpha \)-fat query triangle \( \Delta \), we can compute, in \( O^\ast(\varepsilon^{-2}) \) time, an approximate count \( t_\Delta \) satisfying \( (1 - \varepsilon)|\Delta \cap P| \leq t_\Delta \leq |\Delta \cap P| \).

**Corollary 5.2.2** Let \( P \) be a set of \( n \) points in the plane, and let \( \varepsilon \) be a given positive parameter. Then we can preprocess \( P \) into a data structure of size \( O^\ast(\varepsilon^{-2}n \log n) \), in time \( O^\ast(\varepsilon^{-2}) \), such that, for any line \( \ell \), a point \( p \) on \( \ell \) or above \( \ell \), and distance \( d \), we can compute, in \( O^\ast(\varepsilon^{-2}) \) time, an approximate count \( t_{\ell,p,d} \) of the exact number \( N_{\ell,p,d} \) of the points of \( P \) which lie above \( \ell \) and at distance at most \( d \) from \( p \), so that \( (1-\varepsilon)N_{\ell,p,d} \leq t_{\ell,p,d} \leq N_{\ell,p,d} \).

**Corollary 5.2.3** Let \( P \) be a set of \( n \) points in \( \mathbb{R}^3 \), \( \Gamma \) a collection of convex semialgebraic ranges of constant description complexity, and \( \varepsilon \) a given positive parameter. Then we can preprocess \( P \) into a data structure of size \( O^\ast(\varepsilon^{-2}n \log n) \), in time \( O^\ast(\varepsilon^{-2}) \), such that, for any query range \( \gamma \in \Gamma \), we can compute, in \( O^\ast(\varepsilon^{-2}n^{1/2}) \) time, an approximate count \( t_\gamma \) of the number of points of \( P \) outside \( \gamma \), satisfying \( (1 - \varepsilon)|\gamma^c \cap P| \leq t_\gamma \leq |\gamma^c \cap P| \).

**Corollary 5.2.4** Let \( \mathcal{B} \) be a set of \( n \) balls in \( \mathbb{R}^3 \), and let \( \varepsilon \) be a given positive parameter. Then we can preprocess \( \mathcal{B} \) into a data structure of size \( O^\ast(\varepsilon^{-2}n \log n) \), in time \( O^\ast(\varepsilon^{-2}) \), such that, for any query ray or segment \( \rho \), we can compute, in \( O^\ast(\varepsilon^{-2}n^{2/3}) \) time, an approximate count \( t_\rho \) of the exact number \( N_\rho \) of the balls of \( \mathcal{B} \) intersected by \( \rho \), satisfying \( (1 - \varepsilon)N_\rho \leq t_\rho \leq N_\rho \).

### 5.3 Second Approach

Another approach to approximate range counting has been presented in [16, 17], in which, rather than using a range emptiness searching procedure as a black box, one modifies the partition tree of the range emptiness data structure, and augments each of its inner nodes with a relative \( (p, \varepsilon) \)-approximation set (as reviewed in Section 1.1.9). These sets are then used to obtain the approximate count of any “heavy” range that visits that node (see below for details). As we show in this section, this approach too can be adapted to yield efficient approximate range counting algorithms for semialgebraic ranges, with a slightly improved dependence of their performance on \( \varepsilon \) (as in the case of halfspaces, studied in [16, 17]).

Let \( P \) be a given set of \( n \) points in \( \mathbb{R}^d \), and let \( \varepsilon > 0 \) be a given parameter. Following the scheme of [17], we build a partition tree \( T \), where each node \( v \) represents a subset \( P_v \subset P \).
Algorithm 1

\textbf{function} \textsc{Node::ApproxCount}(\textit{Node} \textit{v}, \textit{γ} ∈ \textit{Γ}, \varepsilon > 0)

\hspace{1em} // if \textit{v} is a leaf use a special procedure to approximate \(|\gamma \cap P_v|\)
\hspace{1em} if \textit{v} is a leaf then
\hspace{2em} \textbf{return} \textsc{Leaf::ApproxCount}(\textit{v}, \textit{γ}, \varepsilon);

\hspace{1em} // find the children of \textit{v} whose cells intersect \(\partial \gamma\),
\hspace{1em} // and those whose cells are contained in \(\gamma\)
\hspace{1em} \((\text{INT}_{\text{H}_v, \gamma}, \text{CNT}_{\text{H}_v, \gamma}) := \text{Partition::compare}(\Pi_v, \gamma)\);

\hspace{1em} // if \(\gamma\) does not intersect too many cells, recurse into them
\hspace{1em} if \(|\text{CNT}_{\text{H}_v, \gamma}| == 0\) and \(|\text{INT}_{\text{H}_v, \gamma}| < \frac{r_v}{\zeta^{-1}(r_v)}\) then
\hspace{2em} \text{answer} := 0;
\hspace{2em} \textbf{for each} \(\xi \in \text{INT}_{\text{H}_v, \gamma}\)
\hspace{3em} \text{answer} += \textsc{Node::ApproxCount}(\xi, \gamma, \varepsilon);
\hspace{2em} \textbf{return} \text{answer};

\hspace{1em} // otherwise, \(\gamma\) is deep and we can use \textsc{A}_v to approximate \(|\gamma \cap P|\)
\hspace{1em} \textbf{else}
\hspace{2em} \textbf{return} \textsc{DeepRange::ApproxCount}(\textit{v}, \textit{γ}, \varepsilon);

(\text{where} \(P_{\text{root}} = P\)). \text{For each node} \textit{v} of \(\mathcal{T}\), we set \(n_v = |P_v|\). We set a threshold \(n_0 = n_0(\varepsilon)\) whose value will be fixed later. If \(n_v < n_0\) then \textit{v} becomes a leaf of \(\mathcal{T}\). Otherwise, we choose a parameter \(r_v\) (which generally depends on \(n_v\)), and construct a shallow elementary cell partition \(\Pi_v = \{(P_1, \sigma_1), \ldots, (P_s, \sigma_s)\}\) of \(P_v\), similar to the one in Theorem 4.2.2, so that \(n_v/r_v < |P_i| \leq 2n_v/r_v\) for each \(i\) (so \(s = O(r_v)\)), each \(\sigma_i\) is an elementary cell containing \(P_i\), and each \((n_v/r_v)\)-shallow range in \(\Gamma\) crosses at most \(r_v/\zeta^{-1}(r_v)\) cells of \(\Pi_v\), where \(\zeta(\cdot)\) is the associated function (as defined in Chapter 4). Note that no \((n_v/r_v)\)-shallow range \(\gamma\) can fully contain a cell \(\sigma_i\), so \(\gamma\) crosses every cell \(\sigma_i\) that it meets. We create \(s\) children \(v_1, \ldots, v_s\) of \textit{v} and assign \(P_i\) to \(v_i\) for each \(i\).

In addition, at each node \textit{v} of \(\mathcal{T}\) we store a relative \((1/r_v, \varepsilon/2)\)-approximation \(\textsc{A}_v\) of \(P_v\) (for the range space induced by \(\Gamma\)).

\textbf{Performing an approximate counting query.} Querying with a range \(\gamma \in \Gamma\) is summarized in Algorithm 1 above, and proceeds as follows: We traverse the augmented partition tree, starting at the root. When visiting a node \textit{v}, if the boundary of \(\gamma\) intersects more than \(\frac{r_v}{\zeta^{-1}(r_v)}\) cells of the partition \(\Pi_v\), or if \(\gamma\) fully contains one of these cells, then \(\gamma\) cannot be \((n_v/r_v)\)-shallow with respect to \(P_v\), that is, \(|\gamma \cap P_v| > n_v/r_v\). In that case we answer the approximate range counting query for \(P_v\) using \(\textsc{A}_v\). Since \(\textsc{A}_v\) is a \((1/r_v, \varepsilon/2)\)-approximation
and $\gamma$ is deep, in the above sense, we have, by definition,

$$(1 - \varepsilon/2) |P_v| |A_v| |\gamma \cap A_v| \leq |\gamma \cap P_v| \leq (1 + \varepsilon/2) |P_v| |\gamma \cap A_v|.$$ 

We therefore count (either in brute force or in some more sophisticated way) the points of $\gamma \cap A_v$ and multiply the count by $|P_v| |A_v|$ to get a good estimation of $|\gamma \cap P_v|$.

If $\gamma$ is shallow with respect to $P_v$, we recursively obtain an $\varepsilon$-approximate count at all children of $v$ whose cells intersect $\gamma$, and return the sum of the answers. By construction, the number of children we need to recurse into in this case is at most $\varepsilon^{-1} r_v \zeta - 1 (r_v)$.

Finally, if $v$ is a leaf we use a designated procedure for counting $\gamma \cap P_v$, typically by brute force.

Algorithm 1 displayed above presents a pseudo-code of the approximate counting algorithm. The algorithm calls three subprocedures:

1. **Leaf::ApproxCount($v$, $\gamma$, $\varepsilon$)** — a subprocedure that approximates the count for a range $\gamma$ at a leaf node $v$. This procedure can be naively implemented by checking, in brute force, each point of $P_v$ for containment in $\gamma$. However, this can be improved by recursively building a data structure that answers approximate counting queries on $P_v$, and then by using this data structure at query time to efficiently approximate $|\gamma \cap P_v|$.

2. **Partition::compare($\Pi$, $\gamma$)** — a subprocedure that compares a range $\gamma$ with a partition $\Pi = \{(P_1, \sigma_1), \ldots, (P_m, \sigma_m)\}$. This subprocedure finds all the elementary cells $\sigma_i$ that the boundary of $\gamma$ intersects, and all the elementary cells that $\gamma$ fully contains. The output of this subprocedure is the subset $INT_{\Pi, \gamma}$ of elementary cells intersected by $\partial \gamma$, and the subset $CNT_{\Pi, \gamma}$ of elementary cells contained in $\gamma$. We also assume that the sizes of these subsets ($|INT_{\Pi, \gamma}|$ and $|CNT_{\Pi, \gamma}|$) can be easily computed once the subsets are obtained. This procedure can be naively implemented by checking, in brute force, each elementary cell for intersection or containment. However this can be improved using a more elaborate procedure, for example, by lifting the cells into a higher-dimensional space where the ranges become linear and then by preprocessing the convex hulls of the images of the cells into a data structure for half-space range reporting or counting, which can compute $CNT_{\Pi, \gamma}$ and $INT_{\Pi, \gamma}$ in time sublinear in $r_v$.

3. **DeepRange::ApproxCount($v$, $\gamma$, $\varepsilon$)** — a subprocedure that approximates the count of a deep range $\gamma$ (with respect to $P_v$) using the relative approximation set $A_v$. Again, this can be done using either a naive brute force approach or a more sophisticated one.
5.3.1 Choosing the parameters

Choosing \( r_v \). We choose \( r_v = \frac{n_v}{(\zeta^{-1}(n_v))^{\alpha'}} \), for some \( 1 < \alpha' < \frac{\log(n_v/4)}{\log\zeta^{-1}(n_v)} \). The concrete value of \( \alpha' \) will be discussed later. Typically, \( \zeta(m) = \Theta(m^\delta) \) for some constant \( \delta \geq 1 \), so, for example, this restricts our choice to \( r_v < n_v^{1/2} \) for approximating the number of points outside a query cylinder in 3-space, or \( r_v < n_v^{3/4} \) for approximating the number of balls a query segment intersects in 3-space. The cases of estimating the number of points contained in a fat triangle, or the number of points contained in a semi-disk are especially interesting because in those cases \( \zeta(n_v) = \Theta(n_v \text{polylog}(n_v)) \), and our choice of \( r_v \) is therefore restricted by \( r_v < \text{polylog}(n_v) \).

Choosing the size of \( A_v \). We set \( A_v \) to be a random sample of size \( \frac{cr_v}{\varepsilon^2} \log r_v \) (for an appropriate constant \( c > 0 \)), which is a relative \((1/r_v, \varepsilon/2)\)-approximation with probability at least \( 1 - 1/r_v^b \), where \( b = b(c) \) is linear in \( c \). For technical reasons that arise in the analysis of the storage requirements we need to restrict the size of \( A_v \). Intuitively we need \( A_v \) to be small enough relatively to \( P_v \). More specifically, we will require that \( |A_v| = \frac{cr_v}{\varepsilon^2} \log r_v \leq \frac{n_v}{k \log^4 \log n_v} \), for some constant \( k \) that will be specified later. By the choice of \( r_v \) we need \( (\zeta^{-1}(n_v))^{\alpha'} \geq \frac{ck \log^3 \log n_v}{\varepsilon^2} \log \frac{n_v}{\zeta^{-1}(n_v)\alpha'} \), which holds if we choose \( n_v \geq \zeta \left( \left( \frac{c'}{\varepsilon^2} \log^3 (\log n_v) \log \frac{n_v}{\zeta^{-1}(n_v)^{\alpha'}} \right)^{1/\alpha'} \right) \) for an appropriate multiple \( c' \) of \( c \). We thus set

\[ n_0 = n_0(\varepsilon) = \zeta \left( \left( \frac{c'}{\varepsilon^2} \log^3 \left( \log \frac{1}{\varepsilon} \right) \log \frac{1}{\varepsilon} \right)^{\alpha'} \right), \]

and enforce this lower bound on \( n_v \) by declaring a node to be a leaf if it has at most \( n_0 \) points, and continuing recursively to build the tree if it has more than \( n_0 \) points.

5.3.2 Analysis of query-time, storage and preprocessing

Let \( Q_{\text{leaf}}(n, \varepsilon) \), \( Q_{\text{comp}}(m) \) and \( Q_{\text{deep}}(n, \varepsilon) \) be the query time of the three respective procedures

leaf::ApproxCount, Partition::compare and DeepRange::ApproxCount, where \( r_v/2 \leq m \leq r_v \) is the number of elementary points in the partition \( \Pi_v \), \( n = |P_v| \) is (with slight abuse of notation) the number of points associated with the current node, and \( \varepsilon \) is the approximation parameter. Similarly let \( T_{\text{leaf}}(n, \varepsilon) \), \( S_{\text{leaf}}(n, \varepsilon) \) \( (T_{\text{comp}}(m), S_{\text{comp}}(m), \)

\( T_{\text{deep}}(n, \varepsilon), S_{\text{deep}}(n, \varepsilon)) \) be the preprocessing time and storage required by leaf::ApproxCount (Partition::compare, DeepRange::ApproxCount). Also, let \( T_{\text{part}}(n, \varepsilon) \) be the time needed to construct the partition at a node \( v \). Similarly to the analysis in [17], we obtain the following recurrences for the query time \( Q(n, \varepsilon) \), preprocessing time \( T(n, \varepsilon) \) and storage \( S(n, \varepsilon) \) of our data structure; here \( r = r_v \) is the partitioning parameter used at \( v \).
5.3 Second Approach

\[ Q(n, \varepsilon) \leq \begin{cases} Q_{\text{comp}}(m) + \max \left\{ Q_{\text{deep}}(n, \varepsilon), \frac{c}{r^{\frac{1}{\varepsilon}}(r)} Q(n/r, \varepsilon) \right\}, & \text{if } n > n_0(\varepsilon), \\ Q_{\text{leaf}}(n, \varepsilon), & \text{otherwise,} \end{cases} \]

\[ S(n, \varepsilon) \leq \begin{cases} S_{\text{comp}}(m) + S_{\text{deep}}(n, \varepsilon) + \sum_{i=1}^{r} S(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon), \\ S_{\text{leaf}}(n, \varepsilon), & \text{otherwise,} \end{cases} \]

and

\[ T(n, \varepsilon) \leq \begin{cases} T_{\text{part}}(n, r) + T_{\text{comp}}(r) + T_{\text{deep}}(n, \varepsilon) + \sum_{i=1}^{r} T(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon), \\ T_{\text{leaf}}(n, \varepsilon), & \text{otherwise,} \end{cases} \]

where \( c \) is an appropriate constant, \( n_i \leq 2n/r \) for each \( i \), and \( \sum_{i=1}^{r} n_i = n \).

5.3.3 Naive implementation

We present a naive choice of \( n_0(\varepsilon), r_v \), and a simple implementation of the three subprocedures \textsc{Leaf::ApproxCount}, \textsc{Partition::Compare} and \textsc{DeepRange::ApproxCount}. Aronov and Sharir [17] also consider more sophisticated implementations, which yield a somewhat improved dependency on \( \varepsilon \), but we will restrict the analysis to the naive implementations.

We store at \( v \) a random sample of \( \frac{c r_v}{\varepsilon^2} \log r_v \) points, for some absolute constant \( c > 0 \). Following [47, 55] (see also Section 1.1.9) this sample is a relative \((1/r_v, \varepsilon/2)\)-approximation with probability at least \( 1 - \frac{1}{r_v^b} \), where \( b = b(c) \) is a linear increasing function of \( c \). In Section 5.3.5 we show, similar to [17], how to increase the success probability, making the failure probability polynomially small in \( n \).

In this naive implementation, \textsc{Leaf::ApproxCount} and \textsc{Partition::Compare} are executed in brute force. That is, \textsc{Leaf::ApproxCount}(\( v, \gamma, \varepsilon \)) checks every point \( p \in P_v \) and counts how many are contained in \( \gamma \), and \textsc{Partition::Compare}(\( \Pi, \gamma \)) checks every elementary cell \( \sigma_i \in \Pi \), and reports which of these cells are intersected by \( \partial \gamma \) and which are fully contained in \( \gamma \). The costs of these naive implementations are, respectively, \( O(r_v) \), and \( O(n_v) \).

Implementing \textsc{DeepRange::ApproxCount}(\( v, \gamma, \varepsilon \)) can also be done in brute force, by checking for each point \( p \in A_v \) whether it is contained in \( \gamma \), thereby computing the value \( |A_v \cap \gamma| \) exactly, and returning the value \( t_v := |A_v \cap \gamma| \frac{n_v}{|A_v|} \) as the desired approximation of \( |P_v \cap \gamma| \), satisfying \((1 - \varepsilon/2)|P_v \cap \gamma| \leq t_v \leq (1 + \varepsilon/2)|P_v \cap \gamma| \). The cost of such a naive implementation is \( O(|A_v|) \). We can improve this bound if we allow ourselves to compute an estimation of \( |A_v \cap \gamma| \). Such an estimation can be computed by calling \textsc{Node::ApproxCount}(\( v', \gamma, \varepsilon/3 \)) where \( v' \) points to a secondary approximation data structure that we build for \( A_v \).
From the properties of relative \((1/r_v, \varepsilon/2)\)-approximation we have
\[
(1 - \frac{\varepsilon}{2})|P_v \cap \gamma| |A_v| \leq |A_v \cap \gamma| \leq (1 + \frac{\varepsilon}{2})|P_v \cap \gamma| |A_v|.
\]

The output \(t_\gamma\) of NODE::APPROXCOUNT\((v', \gamma, \varepsilon/3)\) satisfies
\[
(1 - \frac{\varepsilon}{3})|A_v \cap \gamma| \leq t_\gamma \leq (1 + \frac{\varepsilon}{3})|A_v \cap \gamma|,
\]
and therefore, for \(\varepsilon < 1\), the following holds:
\[
(1 - \varepsilon)|P_v \cap \gamma| |A_v| \leq (1 - \frac{\varepsilon}{2})(1 - \frac{\varepsilon}{3})|P_v \cap \gamma| |A_v| \leq t_\gamma \leq (1 + \frac{\varepsilon}{2})(1 + \frac{\varepsilon}{3})|P_v \cap \gamma| |A_v| \leq (1 + \varepsilon)|P_v \cap \gamma| |A_v|.
\]

Recall that we have set
\[
n_0(\varepsilon) = \zeta \left( \frac{1}{\varepsilon^2 \log^3 \left( \log \frac{1}{\varepsilon} \right) \log \frac{1}{\varepsilon} } \right)^{1/\alpha'}.
\]

Our goal is to make the query time satisfy \(Q(n, \varepsilon) \leq F(\varepsilon) \frac{n}{\zeta^{-1}(n)} \log^\beta n\), for some absolute constant parameter \(\beta\) and function \(F(\varepsilon)\). In particular we want this bound to hold for leaf nodes as well as deep nodes. For a leaf node we have \(Q_{\text{leaf}}(n, \varepsilon) = O(n)\) so we require \(n_0(\varepsilon) \leq c''F(\varepsilon) \frac{n_0(\varepsilon)}{\zeta^{-1}(n_0(\varepsilon))} \log^\beta n_0(\varepsilon)\), where \(c'' \geq 1\) is some constant. Clearly this holds when \(n_0(\varepsilon) = F(\varepsilon) \frac{n_0(\varepsilon)}{\zeta^{-1}(n_0(\varepsilon))}\). We therefore put
\[
F(\varepsilon) = \zeta^{-1}(n_0(\varepsilon)) = \zeta^{-1} \left( \zeta \left( \frac{c'}{\varepsilon^2 \log^3 \left( \log \frac{1}{\varepsilon} \right) \log \frac{1}{\varepsilon} } \right)^{1/\alpha'} \right)
= \left( \frac{c'}{\varepsilon^2 \log^3 \left( \log \frac{1}{\varepsilon} \right) \log \frac{1}{\varepsilon} } \right)^{1/\alpha'}.
\]

Recall that \(\alpha' > 1\) is chosen such that \((\zeta^{-1}(n))^{\alpha'} < n\). If \(\zeta(n) = O(n^\delta)\), this means that \(1 < \alpha' < \delta\), and \(F(\varepsilon)\) is approximately of the form \(1/\varepsilon^\gamma\), where \(\gamma\) satisfies \(2/\delta < \gamma < 2\). Note that \(\gamma\) approaches its upper (resp., lower) bound as \(\alpha'\) approaches \(1\) (resp., \(\delta\)). If \(\zeta(n) = O(n \cdot \text{polylog}(n))\) then \(\alpha'\) must be chosen close to \(1\) and so \(F(\varepsilon)\) is approximately of the form \(1/\varepsilon^2\).

**Query time.** Once \(\alpha'\) is fixed, the recurrence for \(Q(n, \varepsilon)\) becomes:
\[
Q(n, \varepsilon) \leq \begin{cases} 
O \left( \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right) + \max \left\{ Q \left( \frac{n}{k \log^3 \left( \log n \right) \cdot \frac{1}{3}} \right), \mu \left( \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right) Q \left( 2(\zeta^{-1}(n))^{\alpha'}, \varepsilon \right) \right\}, & \text{if } n > n_0(\varepsilon), \\
O(n), & \text{otherwise},
\end{cases}
\]
where $\mu(r)$ is the maximum number of elementary cells a shallow range crosses. That is,

$$
\mu(r) = O \left( \frac{r}{\zeta^{-1}(r)} \right) = O \left( \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right).
$$

We show in Section 5.3.4 that the recurrence solves to $Q(n, \varepsilon) \leq F(\varepsilon) \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \log^\beta n$, for an appropriate choice of $\beta = \beta(\alpha')$.

**Storage.** The storage bound $S(n, \varepsilon)$ satisfies the recurrence:

$$
S(n, \varepsilon) \leq \begin{cases} 
O \left( \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right) + 
S \left( \frac{n}{k \log^3(n)} \frac{\varepsilon}{3} \right) + 
\sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} S(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon) \\
\text{otherwise}. & 
\end{cases}
$$

where $n_i \leq 2(\zeta^{-1}(n))^{\alpha'}$ for each $i$, and $\sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} n_i = n$.

We show in section 5.3.4 that the recurrence solves to $S(n) = O(n)$, with a constant that is independent of $\varepsilon$. As will follow from the analysis, the factor $\log^3 \log n$ can be replaced by any factor of the form $\log^{2+\delta}(\log n)$, for any positive constant $\delta$.

**Preprocessing.** In this naive implementation, $T_{\text{comp}}(k) = O(k)$ and $T_{\text{leaf}}(n, \varepsilon) = O(n)$. For $T_{\text{part}}(n, r)$ we use the bounds in Chapter 4, namely, $T_{\text{part}}(n, r) = O(n^{1+\delta})$, for any $\delta > 0$. The resulting recurrence for $T(n, \varepsilon)$ is thus:

$$
T(n, \varepsilon) \leq \begin{cases} 
O(n^{1+\delta}) + 
T \left( \frac{n}{k \log^3(n)} \frac{\varepsilon}{3} \right) + 
\sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} T(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon) \\
\text{otherwise}. & 
\end{cases}
$$

where $n_i \leq 2(\zeta^{-1}(n))^{\alpha'}$ for each $i$, and $\sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} n_i = n$, and we choose the $O(n^{1+\delta})$ version if $\alpha' > \alpha'_0$ and the $O(n \log n)$ version otherwise. It is straightforward to verify that the solution of this recurrence is

$$
T(n, \varepsilon) = O(n^{1+\delta}),
$$

for any $\delta > 0$. Here too the constants of proportionality are independent of $\varepsilon$. We summarize these results in the following theorem.

**Theorem 5.3.1** Given a family $\Gamma$ of semialgebraic ranges of constant description complexity, we can preprocess a set $P$ of $n$ points in $\mathbb{R}^d$, with a pre-specified error parameter $0 < \varepsilon < 1$, into a data structure of size $O(n)$, independent of $\varepsilon$, so that, with high probability, for any query range $\gamma \in \Gamma$, we can obtain a relative $\varepsilon$-approximate count of $\gamma \cap P$, in
Approximate Semialgebraic Range Counting

time $O\left(\varepsilon^{-\alpha} \frac{n}{\zeta^{-1}(n)} \log^\beta n\right)$, where $\alpha$ can be chosen anywhere in $\left(2 \log_n \frac{n}{\zeta^{-1}(n)}, 2\right)$, and $\beta$ is a constant that depends on $\alpha$.

The data structure can be constructed deterministically, except for the random samplings that produce the various relative approximations. The (worst-case) preprocessing cost is $O(n^{1+\delta})$, for any $\delta > 0$.

5.3.4 Solving the recurrences

In this section we show that the recurrences mentioned in the earlier section yield indeed the bounds asserted above.

Analysis of query time. We want to show that the recurrence

$$Q(n, \varepsilon) \leq \begin{cases} O\left(\frac{n}{(\zeta^{-1}(n))^{\alpha'}}\right) + \max \left\{ Q\left(\frac{n}{k \log^4(\log n)}, \frac{n}{\zeta^{-1}(n)}\right), \mu\left(\frac{n}{(\zeta^{-1}(n))^{\alpha'}}\right) Q(2(\zeta^{-1}(n))^{\alpha'}, \varepsilon) \right\}, & \text{if } n > n_0(\varepsilon) \text{,} \\ O(n), & \text{otherwise} \end{cases}$$

solves to $Q(n, \varepsilon) \leq F(\varepsilon) \frac{n}{(\zeta^{-1}(n))} \log^\beta n$, with $F(\varepsilon) = \zeta^{-1}(n_0(\varepsilon)) = \left(\frac{\varepsilon}{\zeta^{-1}(n_0(\varepsilon))}\log^3\left(\frac{1}{\varepsilon}\right)\log\frac{1}{\varepsilon}\right)^{1/\alpha'}$.

We prove this by induction on $n$. For $n \leq n_0(\varepsilon)$ we need to show that $F(\varepsilon) \frac{n}{(\zeta^{-1}(n))} \log^\beta n \geq O(n)$. Which can be easily seen to be true since $F(\varepsilon) \frac{n}{(\zeta^{-1}(n))} \log^\beta n = \zeta^{-1}(n_0(\varepsilon)) \frac{n}{(\zeta^{-1}(n))} \log^\beta n_0(\varepsilon)$, and $\frac{\zeta^{-1}(n_0(\varepsilon))}{(\zeta^{-1}(n))} \geq 1$. For carrying out the induction step for larger values of $n$ we need to show that

$$c_1 \frac{n}{(\zeta^{-1}(n))^{\alpha'}} + \max \left\{ \frac{F(\varepsilon) \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \log^\beta \frac{n}{(\zeta^{-1}(n))^{\alpha'}}}{\zeta^{-1}(n_0(\varepsilon))}, \frac{c_2}{(\zeta^{-1}(n))^{\alpha'}} F(\varepsilon) \frac{2(\zeta^{-1}(n))^{\alpha'}}{(\zeta^{-1}(2(\zeta^{-1}(n))^{\alpha'})) \log^\beta (2(\zeta^{-1}(n))^{\alpha'})} \right\} \leq F(\varepsilon) \frac{n}{(\zeta^{-1}(n))} \log^\beta n$$

for appropriate constants $c_1, c_2$. If we calibrate the constant in the definition of $F(\varepsilon)$, so that $F(\varepsilon)$ is always at least $2c_1$, then

$$c_1 \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \leq c_1 \frac{n}{\zeta^{-1}(n)} \leq \frac{1}{2} F(\varepsilon) \frac{n}{\zeta^{-1}(n)} \leq \frac{1}{2} F(\varepsilon) \frac{n}{\zeta^{-1}(n)} \log^\beta n.$$

Clearly,

$$\frac{\log^\beta \frac{n}{(\zeta^{-1}(n))^{\alpha'}}}{\zeta^{-1}(n)} \log^\beta \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \leq \frac{\log^\beta \frac{n}{(\zeta^{-1}(n))^{\alpha'}}}{\zeta^{-1}(n)} \log^\beta \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \leq \frac{n}{(\zeta^{-1}(n))} \log^\beta n \leq \frac{n}{(\zeta^{-1}(n))} \log^\beta n,$$
where the last inequality coming from our assumption that $\zeta(\cdot)$ is well behaved, in the sense that $\zeta(ab) \leq \zeta(a)\zeta(b)$ for $a,b \geq 1$. Also, under the same assumption,

$$\frac{n}{(\zeta^{-1}(n))^{\alpha'}} \leq \frac{2(\zeta^{-1}(n))^{\alpha'}}{\zeta^{-1}(2(\zeta^{-1}(n))^{\alpha'})} \leq \frac{2n}{\zeta^{-1}(2n)} \leq \frac{2n}{\zeta^{-1}(n)},$$

and

$$\left(\frac{\log(2(\zeta^{-1}(n)^{\alpha'})))}{\log n}\right)^\beta = \left(1 + \frac{\alpha' \log \zeta^{-1}(n)}{\log n}\right)^\beta = \left(\frac{1}{\log n} + \alpha' \log \zeta^{-1}(n)\right)^\beta.$$

Putting it all together, it suffices to show that

$$\frac{1}{2} + \max \left\{ \frac{F(\zeta)}{\zeta^{-1}(n)^{\alpha'}} \left( k \log^3(\log n) \right), \left( \frac{1}{\log n} + \alpha' \log n \zeta^{-1}(n)\right)^\beta \right\} \leq 1.$$ 

An appropriate choice of $\alpha'$ and $n_0(\varepsilon)$ guarantees that the expression in the rightmost parentheses is strictly smaller than 1. This, and the way $F(\varepsilon)$ is defined, imply the desired inequality if $\beta$ and $k$ are chosen large enough.

Note the trade-off between $\beta$ and $F(\varepsilon)$. We can make $F(\varepsilon)$ asymptotically smaller if we increase $\alpha'$ and consequently increase $\beta$. (The bound on the cost of a query can thus be optimized, for given values of $n$ and $\varepsilon$.)

**Analysis of storage.** We want to show that the solution of the recurrence

$$S(n, \varepsilon) \leq \begin{cases} O\left(\frac{n}{(\zeta^{-1}(n))^{\alpha'}}\right) + S\left(\frac{n}{k \log^3(\log n)}\right) + \sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} S(n_i, \varepsilon), & \text{if } n > n_0(\varepsilon) \\ O(n), & \text{otherwise} \end{cases},$$

is linear in $n$, where $n_i \leq 2(\zeta^{-1}(n))^{\alpha'}$ for each $i$ and $\sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} n_i = n$.

We first show that $S(n) \leq Dn \log \log n$, where $D$ is a constant that does not depend on $\varepsilon$. We prove this by induction on $n$. Clearly it holds for $n \leq n_0(\varepsilon)$ for an appropriate choice of $D$ (independent of $\varepsilon$). For larger values of $n$, we need to show that

$$c_2 \frac{n}{(\zeta^{-1}(n))^{\alpha'}} + \frac{Dn}{k \log^3(\log n)} \log \log \left(\frac{n}{k \log^3(\log n)}\right) + \sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} Dn_i \log \log(2(\zeta^{-1}(n))^{\alpha'}) \leq Dn \log \log n,$$

for an appropriate constant $c_2$. In the first term we use $\frac{n}{(\zeta^{-1}(n))^{\alpha'}} < n$; in the second term we use $\log \log \left(\frac{n}{k \log^3(\log n)}\right) < \log \log n$; and in the third term we use $\sum n_i = n$. It is therefore sufficient to prove that
\[
\frac{c_2}{D \log \log n} + \frac{1}{k \log^3(\log n)} + \frac{\log \log \left( \frac{2n^{(\zeta^{-1}(n))^{\alpha'}}}{n} \right)}{\log \log n} \leq 1.
\]

By our choice of \(\alpha'\), \(\frac{n^{\alpha'}}{(\zeta^{-1}(n))^{\alpha'}} > 4\), and so

\[
\frac{\log \log \left( \frac{2n^{(\zeta^{-1}(n))^{\alpha'}}}{n} \right)}{\log \log n} \leq \frac{\log \log \left( n^{(\frac{1}{2} (\frac{n}{(\zeta^{-1}(n))^{\alpha'}}))^{(\frac{1}{2})}} \right)}{\log \log n} = \frac{\log (\log n - \frac{1}{2} \log \frac{n}{(\zeta^{-1}(n))^{\alpha'}})}{\log \log n} = \frac{\log n + \log \left( 1 - \frac{1}{2} \log \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right)}{\log \log n}.
\]

It therefore remains to show that

\[
\frac{c_2}{D \log \log n} + \frac{1}{k \log^3(\log n)} + \frac{\log \left( 1 - \frac{1}{2} \log \frac{n}{(\zeta^{-1}(n))^{\alpha'}} \right)}{\log \log n} \leq 0.
\]

Recall that \(\zeta(r) = O(r^\delta)\), for some \(\delta\) that depends on the “complexity” of the ranges. In this case, the inequality becomes

\[
\frac{c_2}{D} + \frac{1}{k \log^2(\log n)} + \frac{\log \left( \frac{1}{2} \left( 1 + \frac{\alpha'}{\delta} \right) \right)}{\log \log n} \leq 0,
\]

which holds if we choose, say, \(\alpha' < (\frac{2}{22/\varepsilon} - 1)\delta\), and \(D > 2c_2k\). (Again, we have a trade-off: to reduce \(F(\varepsilon)\) asymptotically, we have to choose bigger \(\alpha'\), so \(k\) has to be chosen larger, which causes the constant of proportionality in the storage bound to increase.)

For certain classes of ranges we have \(\zeta(r) = O(r \cdot \text{polylog}(r)) < r^d\), for any \(q > 0\). In such cases we fix \(q\) to be some constant and the bound is established trivially. In both cases the argument that \(S(n, \varepsilon) < D \log \log n\) holds. Using this for \(n = |A|\) we get that \(S(|A|, \frac{\varepsilon}{2}) \leq \frac{Dn \log \log n}{k \log^3(\log n)} = O(\frac{n}{\log^2(\log n)})\).

Since \(O(\frac{n}{\log^2(\log n)}) > O(\frac{n}{\zeta^{-1}(n)})\) (recall that \(\zeta^{-1}(n)\) is at most \(n^q\) for some \(q < 1\)), we can replace the first two terms in the recurrence and obtain:

\[
S(n) \leq \begin{cases} 
\frac{D'n}{\log^2(\log n)} + \sum_{i=1}^{n/(\zeta^{-1}(n))^{\alpha'}} S(n_i), & \text{if } n > n_0(\varepsilon), \\
\frac{D'n}{\log^2(\log n)}, & \text{otherwise},
\end{cases}
\]

for some absolute constant \(D'\), with constraints on the \(n_i\) as mentioned earlier.

Aronov et al. showed in [17] that when \(\zeta^{-1}(n) = n^\delta\), for some \(0 < \delta < 1\), this recurrence solves to \(S(n) = O(n)\), with the constant of proportionality independent of \(\varepsilon\). We repeat their analysis for general \(\zeta^{-1}(n)\). We unwind the recurrence as follows: When expanding a node \(v\) of the tree and forming its children, each child \(w\) satisfies, by construction,
5.3 Second Approach

\((\zeta^{-1}(n_v))^{\alpha'} \leq n_w \leq 2(\zeta^{-1}(n_v))^{\alpha'} \leq \sqrt{n_v(\zeta^{-1}(n_v))^{\alpha'/2}}\). The last inequality is true because, by our choice of \(\alpha'\), we have \(\frac{n}{\zeta^{-1}(n_v)^{\alpha'}} > 4\). We say that a node \(v\) lies at a level \(j\) if

\[
\left(1 + \frac{\alpha' + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j+1} < \log n_v \leq \left(1 + \frac{\alpha' + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j},
\]

or, in other words,

\[
n \left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j+1} < n_v \leq n \left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j}.
\]

Thus the root lies at level 0, and the maximum level is \(O(\log \log n)\). Also, no two nodes on a common path have the same level, so the sum of the sizes \(n_v\), over all nodes \(v\) of a fixed level, is at most \(n\). Hence, the sum of the overhead terms of all nodes at level \(j\) is at most:

\[
\frac{D'n}{\log^2 \log n \left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j+1}} = \frac{D'n}{\log^2 \left(\left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^{j+1} \log n\right)} = \frac{D'n}{(\log \log n + (j + 1) \log \left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^2}.
\]

The sum of the amounts of storage \(S(n_w)\) at the leaves \(w\) of the tree is clearly at most \(D'n\). Hence, the overall storage requirements is at most

\[
D'n + \sum_{j \geq 0} \frac{D'n}{(\log \log n + (j + 1) \log \left(\frac{1}{2} + \frac{\alpha'}{2} \log_n (\zeta^{-1}(n))\right)^2}.
\]

The smallest value of any denominator is attained at the parents of the leaves \(w\), and is at least \((\log \log n_0(\varepsilon))^2\), because \(\log_n (\zeta^{-1}(n))^{\alpha'} < 1\). This implies (e.g., by replacing the sum by an integral) that the sum can be bounded by \(O(n / \log \log n_0(\varepsilon))\), which by the choice of \(n_0(\varepsilon)\), is \(O(n / \log \log \frac{1}{\varepsilon}) = O(n)\), if \(\varepsilon < 1\) is chosen sufficiently small.

5.3.5 Ensuring high probability

So far we have assumed that for each node \(v\), a relative \((1/r_v, \varepsilon/2)\)-approximation of a required size can be constructed efficiently. A deterministic process that constructs such a relative approximation is fairly complicated; see [21]. Another way to construct a relative \((1/r_v, \varepsilon/2)\)-approximation is by random sampling (see Theorems 1.1.1 and 1.1.2). Although this is much simpler, one still needs to verify that the points that were drawn indeed form a relative approximation with the appropriate parameters. In [17], Aronov and Sharir argue that just drawing points (without any verification) still ensures high success probability. For the sake of completeness, we repeat their argument for the case of semialgebraic ranges.
The implementation we suggested draws at each node $v$ a random sample of size \( \frac{c_v r_v \log r_v}{\varepsilon^2} \) and makes it the approximation set (without any verification). It is difficult to guarantee high success probability, because, as follows from Theorem 1.1.2, the failure probability of such a sample at a node $v$ is only $O(1/r_v^b)$, for some constant $b$ that depends on $c$. For nodes $v$ that are deep in the tree, $r_v$ is small, and the guaranteed success probability becomes smaller, approaching constant probability as we get closer to the leaves. Since the number of distinct ranges is polynomial in $n$, bounding the overall failure probability via a naive probability union bound does not keep the overall failure probability small. Alternatively, increasing the sample size by a factor of $\log n$ would guarantee low failure probability, but this might affect the algorithm performance.

Nevertheless, we argue that drawing random samples at each node (with some additional simple mechanisms) does indeed guarantee high success probability. The intuition is that we can think of all the elements of all approximation sets at all the nodes a query reaches as one approximation set, with elements coming from a node $v$ appearing with weight (or multiplicity) $n_v/r_v$. Another way to look at it is as a sequence of independent Bernoulli trials, with an appropriate weighted sum of their corresponding indicator variables being the approximate count that the algorithm produces. This implies that the errors that the individual relative approximation sets incur tend to cancel each other out, leading to an overall error that is much smaller than the sum of the individual errors.

In more detail, the analysis proceeds as follows. We consider a fixed range $\gamma$, and bound the probability that the query with $\gamma$ fails to produce the desired $\varepsilon$-approximate count. Consider the set $V = V(i)$ of all nodes $v$ that satisfy (i) $n/2^i < n_v < n/2^{i-1}$, for some fixed “level” $i \geq 1$, (ii) $v$ is reached by the query with $\gamma$, and (iii) $\gamma$ intersects too many children of $v$ (and therefore it is deep with respect to $P_v$), so $A_v$ is used to approximate $|\gamma \cap P_v|$. For each of these nodes $v$, we can think of the relative-error approximation $A_v$ as a random sample, where each point of $P_v$ is chosen independently with probability

\[
 p_v = \frac{c_v r_v \log r_v}{n_v},
\]

where $c_v$ is the constant in the bound of [47, 55] (see Theorem 1.1.2), which we adjust at $v$, multiplying it by at most a constant factor to ensure that the probabilities $p_v$ are all equal for nodes $v$ at the same level; we denote this common value by $p(i)$, for nodes at level $i$. At each node $v$ we approximate $|\gamma \cap A_v|$ by some count $A_\gamma$, and add $A_\gamma \cdot n_v/|A_v|$ to the global count. Recall that $A_v$ is a relative $(1/r_v, \varepsilon/2)$-approximation, and that $A_\gamma$ is computed by recursively calling APPROXCOUNT with $\varepsilon/3$ as the error parameter. We can therefore treat that $A_\gamma$ as if it is the actual count of $A_v \cap \gamma$, without affecting the asymptotic bounds. To fit into the new model, we slightly modify this step, and instead we add to the global count

\[
 \frac{|\gamma \cap A_v| \cdot n_v}{\mathbb{E}(|A_v|)} = \frac{|\gamma \cap A_v|}{p_v} = \frac{|\gamma \cap A_v|}{p(i)}.
\]
Hence, the overall count that the approximations yield at a fixed level $i$ is $\sum_{v \in V} |\gamma \cap A_v| = \frac{1}{p^{(i)}} \cdot |\gamma \cap A|$, where $A = \bigcup_{v \in V} A_v$. Under the above assumptions, we can treat $|\gamma \cap A|$ as the sum of independent indicator variables $I_x$, for $x \in \gamma \cap \bigcup_{v \in V} P_v$, where $I_x = 1$ if $x$ is chosen in the respective $A_v$. The expected value of $\sum_{x \in \gamma \cap \bigcup_{v \in V} P_v} I_x$ is

$$\mu = \mathbb{E}(|\gamma \cap A|) = \frac{1}{p^{(i)}} \cdot \sum_{v \in V} |\gamma \cap P_v| = \sum_{v \in V} \frac{c_v r_v \log r_v}{n_v} |\gamma \cap P_v|,$$

where $c_v = \min \{|c_v|, v \in V\}$ and $r_{\min} = \min \{r_v, v \in V\}$. By the variant of Chernoff’s bound, given in corollary A.14 of [12], we have

$$\Pr \{||\gamma \cap A| - \mu| > \varepsilon \mu\} < 2e^{-c(\varepsilon)\mu},$$

where $c(\varepsilon) = \min \left\{ (1 + \varepsilon) \ln(1 + \varepsilon) - \varepsilon, \frac{1}{2} \varepsilon^2 \right\} \approx \frac{1}{2} \varepsilon^2 > \frac{1}{3} \varepsilon^2$, where $\varepsilon$ is sufficiently small. Hence using the bound for $\mu$, the failure probability for $\gamma$ satisfies

$$P_{\text{fail}} = \Pr \{||\gamma \cap A| - \mu| > \varepsilon \mu\} < 2e^{-c_{\min} |V| \log r_{\min}}.$$

In other words, the failure probability (within the present fixed level $i$) depends on $|V|$. Specifically, putting $a = c_{\min}/3$, the above probability is smaller than

$$P_{\text{fail}} < 2e^{-a |V| \log r_{\min}} = \frac{2}{r_{\min}^a |V|},$$

Recall that $r_v$ is chosen to be $r_v = \frac{n_v}{\zeta^{-1}(n_v)^{a'}}$ in our naive implementation. Since $n_v \approx n/2^i$ at our fixed level $i$, the failure probability is at most

$$P_{\text{fail}} < \frac{2}{r_{\min}^a |V|} = 2 \left( \frac{2^i \zeta^{-1}(n/2^i)^{a'}}{n} \right)^a |V| < 2 \left( \frac{2^i \zeta^{-1}(n)^{a'}}{n} \right)^a |V| < 2 \left( \frac{2^i}{n} \right)^a |V|,$$

where $a'$ is some multiple of $a$. The last inequality holds since in the worst case $\zeta(r) = O(r^{\delta})$, for some $\delta > 1$ that depends on the “complexity” of the ranges, and since $a'$ is a fixed constant strictly smaller than $\frac{1}{\log_n \zeta^{-1}(n)}$. 
Denote for simplicity $n^{(i)} := n/2^i$. Fix some fraction $\rho < \log n \cdot \frac{n}{\zeta^{-1}(n)}$. For situations where $n^{(i)} > n^{\rho/2}$ we have the failure probability $P_{\text{fail}} < 2 \left(\frac{2}{n}\right)^{d|V|} < 2n^{-\rho d|V|}$, which is polynomially small in $n$. For situations where $n^{(i)} \leq n^{\rho/2}$ and $|V| > n^{\rho}/n^{(i)}$ we have $P_{\text{fail}} < 2 \left(\frac{2}{n}\right)^{d|V|} < 2 \left(\frac{2}{n}\right)^{d'n^{\rho/2}}$, which is exponentially small in $n$ (or even smaller).

We are left only with situations where $n^{(i)} \leq n^{\rho/2}$ and $|V| \leq n^{\rho}/n^{(i)}$. However, in such cases processing all the nodes in $V$ in brute-force takes time $O \left(\sum_{v \in V} n_v\right) = O(n^{(i)}|V|) = O(n^\rho) = O \left(\frac{n}{\zeta^{-1}(n)}\right)$, which is within our target bound. Note that this part of the procedure is purely deterministic.

To complete the analysis we estimate the probability that the approximate count in the above procedure will fail for at least one level for at least one query range $\gamma$. Since the number of levels is only logarithmic, and the number of (combinatorially different) ranges is only $O(n^\delta)$, it follows that this overall failure probability (namely, that for at least one range the count is not accurate enough) is at most polynomially small in $n$, with an exponent that can be chosen arbitrarily large.

To implement the modified procedure, we first collect, during the preprocessing of the query with $\gamma$, the sets $V^{(i)}$ of all the nodes $v$ at any fixed level $i$, where $h$ is deep in $P_v$. If, for any level $i$, $n^{(i)}$ and $|V^{(i)}|$ are both small, in the sense defined above, we count $\sum_{v \in V} |\gamma \cap P_v|$ by brute-force. Otherwise, we obtain this count by calling DeepRange::ApproxCount with the approximation sets stored at these nodes.

This technique requires us to store the sets $P_v$ for each node, so that we can search any of these sets by brute-force during a query, if necessary. Storing these sets explicitly would require $\Theta(n \log \log n)$ storage. However, as noted in [17] we can reduce the storage to linear, if we maintain just one master list (or array) of all the points in $P$, so that for any node $v$, $P_v$ is contiguous sublist, which can be specified by two pointers to its first and last elements.

The preceding discussion implies that, with high probability, the resulting data structure yields an $\varepsilon$-approximate count of $\gamma \cap P$, for any range $\gamma$. The storage, preprocessing and query time all remain asymptotically the same, but the query procedure is slightly modified, as explained above.

We conclude this discussion with the following theorem.

**Theorem 5.3.2** Given a set of ranges $\Gamma$, we can preprocess a set $P$ of $n$ points in $\mathbb{R}^d$, with a pre-specified error parameter $0 < \varepsilon < 1$, in time $O(n^{1+\delta})$, for any $\delta > 0$, into a data structure of size $O(n)$, independent of $\varepsilon$, so that, with high probability, for any query range $\gamma \in \Gamma$, we can obtain an $\varepsilon$-approximate count of $\gamma \cap P$, in time $O(\varepsilon^{-\alpha} \frac{n}{\zeta^{-1}(n)} \log^\beta n)$, where $\alpha$ can be chosen anywhere in $(2 \log_2 \frac{n}{\zeta^{-1}(n)}, 2)$, $\beta$ is a constant that depends on $\alpha$, and $\zeta(r)$ is the union complexity function associated with $\Gamma$, as defined earlier.
5.4 Reporting the $k$ Heaviest Points in Semialgebraic Ranges

In this section we consider a variation of the range reporting problem where we are given a set $\Gamma$ of ranges and a set $P$ of points so that each point $p \in P$ has a weight $w_p \in \mathbb{R}$. We show how to preprocess $P$ into a data structure of near-linear size such that, given a query range $\gamma \in \Gamma$, and a parameter $k$, we can report the $k$ points in $P \cap \gamma$ that have the highest weights, in time $O^*(\frac{n}{\zeta(r)}) + O(k)$, where $\zeta(r)$ is the union complexity function associated with $\Gamma$ as defined earlier.

The solution is fairly simple. For simplicity of presentation we assume that the weights are all distinct (otherwise we break ties arbitrarily). We build a tree-like data structure, where the root is associated with the set $P$. We store an approximate counting data structure $A_P$, for some absolute constant value of $\varepsilon$, say $\varepsilon = 1/2$, and a reporting data structure $R_P$ at the root. We partition $P$ into two subsets $P_L$ and $P_H$, such that $|P_H| = |P_L|$ (assuming for simplicity that $|P|$ is even) and $w(h) > w(l)$ for every $h \in P_H$ and $l \in P_L$. We associate $P_L$ with the left child $L$ of the root, and $P_H$ with the right child $H$, and then continue the processing recursively at both children.

For a range $\gamma \in \Gamma$, a node $v$ of the tree, and the corresponding subset $P_v$ of $P$ associated with $v$, let $\gamma_{P_v} = P_v \cap \gamma$, and denote by $A(\gamma_{P_v})$ the approximation of $|\gamma_{P_v}|$, as computed by the corresponding approximate range counting procedure $A_{P_v}$. Given a parameter $k$, the $k$ heaviest points of $\gamma \cap P$ can be computed by searching through the tree, starting at its root. We compare $A(\gamma_{P_H})$ to $k(1 + \varepsilon)$, and we distinguish between two cases:

(i) If $A(\gamma_{P_H}) < k(1 + \varepsilon)$ then $|\gamma_{P_H}| < A(\gamma_{P_H})/(1 - \varepsilon) \leq k(1 + \varepsilon)/(1 - \varepsilon) = 3k$. We compute all the points of $\gamma_{P_H}$ using $R_{P_H}$, and report the $k$ points among them with the highest weights. If there are at least $k$ points in $\gamma_{P_H}$, we are done. Otherwise, we have reported $|\gamma_{P_H}|$ points, so we report recursively the $k - |\gamma_{P_H}|$ points of $P_L$ of highest weights.

(ii) If $A(\gamma_{P_H}) > k(1 + \varepsilon)$ then there are more than $k$ points in $\gamma_{P_H}$. In this case we can ignore the points in $P_L$. We therefore continue the search recursively only at $P_H$.

The approximate range counting data structure stored at each node of the tree allows us to quickly decide the relation between $k$ and the size of the range, which lets us either discard half of the points or assure us that generating the full list of points in $P_v \cap \gamma$ does not affect the query time. Clearly, at each level of the tree we either answer the query, or recurse into one sub-tree containing only half of the points, which leads to the following result.

**Theorem 5.4.1** Let $\Gamma$ be a set of semialgebraic ranges of constant description complexity, let $P$ be a set of $n$ weighted points in $\mathbb{R}^d$, and let $\varepsilon$ be a given positive parameter. Then we can preprocess $P$ in time $O(n^{1+\delta})$, for any $\delta > 0$, into a data structure of size $O^*(n)$, in time $O^*(n)$, such that, for a range $\gamma \in \Gamma$ and a parameter $k$, we can report, in $O^*(\frac{n}{\zeta(r)}) + O(k)$ time, the $k$ heaviest points in $P \cap \gamma$, where $\zeta(r)$ is the union complexity function associated
with \( \Gamma \), as defined earlier.

The following are immediate corollaries of Theorem 5.4.1.

**Corollary 5.4.2** Let \( P \) be a set of \( n \) weighted points in the plane, and let \( \alpha, \varepsilon \) be given positive parameters. Then we can preprocess \( P \) into a data structure of size \( O^*(n) \), in time \( O^*(n) \), such that, for any \( \alpha \)-fat query triangle \( \Delta \) and parameter \( k \), we can report, in \( O^*(1) + O(k) \) time, the \( k \) heaviest points in \( P \cap \Delta \).

**Corollary 5.4.3** Let \( P \) be a set of \( n \) weighted points in the plane, and let \( \varepsilon \) be a given positive parameter. Then we can preprocess \( P \) into a data structure of size \( O^*(n) \), in time \( O^*(n) \), such that, for any line \( \ell \), point \( p \) on \( \ell \) or above \( \ell \), distance \( d \) and parameter \( k \), we can report, in \( O^*(1) + O(k) \) time, the \( k \) heaviest points of \( P \) which lie above \( \ell \) and at distance at most \( d \) from \( p \).

**Corollary 5.4.4** Let \( P \) be a set of \( n \) weighted points in \( \mathbb{R}^3 \), \( \Gamma \) a collection of convex semi-algebraic ranges of constant description complexity, and \( \varepsilon \) a given positive parameter. Then we can preprocess \( P \) into a data structure of size \( O^*(n) \), in time \( O^*(n) \), such that, for any query range \( \gamma \in \Gamma \) and a parameter \( k \), we can report, in \( O^*(n^{1/2}) + O(k) \) time, the \( k \) heaviest points of \( P \) outside \( \gamma \).

**Corollary 5.4.5** Let \( B \) be a set of \( n \) weighted balls in \( \mathbb{R}^3 \), and let \( \varepsilon \) be a given positive parameter. Then we can preprocess \( B \) into a data structure of size \( O^*(n) \), in time \( O^*(n) \), such that, for any query ray \( \rho \) and parameter \( k \), we can compute, in \( O^*(n^{2/3}) + O(k) \) time, the \( k \) heaviest balls of \( B \) intersected by \( \rho \).
Bibliography


