Range Searching and Point Location among Fat Objects *

Mark H. Overmars A. Frank van der Stappen

Dept. of Computer Science, Utrecht University,
P.O. Box 80.089, 3508 TB Utrecht, The Netherlands.

Abstract

We present a data structure that can store a set of disjoint fat objects in d-space such that point location and bounded-size range searching with arbitrarily-shaped ranges can be performed efficiently. The structure can deal with either arbitrary (fat) convex objects or non-convex polytopes. The multi-purpose data structure supports point location and range searching queries in time $O(\log^{d-1} n)$ and requires $O(n \log^{d-1} n)$ storage, after $O(n \log^{d-1} n \log\log n)$ preprocessing. The data structure and query algorithm are rather simple.

1 Introduction

Fatness turns out to be an interesting phenomenon in computational geometry. Several papers present surprising combinatorial complexity reductions [3, 15, 22, 26, 32] and efficiency gains for algorithms [1, 4, 19, 28, 33] if the objects under consideration have a certain fatness. Fat objects are compact to some extent, rather than long and thin. Fatness is a realistic assumption, since in many practical instances of geometric problems the considered objects are fat. The aim of studying fatness is to find new fast and simple algorithms or to demonstrate enhanced efficiency of existing algorithms for such practical instances. The achievements of the study of fatness so far include near-linear bounds on the complexity of the union of certain fat figures (e.g. triangles, wedges) in the plane [3, 15, 22, 26], a linear bound on the complexity of the free space for motion planning amidst fat obstacles [32], and efficient algorithms for computing depth orders on certain fat objects [1], binary space partitions for scenes of non-intersecting fat objects in the plane [4], hidden surface removal for fat horizontal triangles [19], point location in fat subdivisions [28], and motion planning in certain realistic settings of fat obstacles in two- and three-dimensional workspaces [33].

In this paper we study two fundamental problems in computational geometry in a context of fat objects: point location and range searching. The point location problem aims at preprocessing a set of disjoint geometric objects for efficiently reporting the specific object containing a query point. The objective of the (general) version of the range searching problem is to preprocess a set of geometric objects for quickly reporting all objects intersecting some query range (e.g. rectangloids, simplices, hyperspheres). We show that arbitrary fat convex objects and/or non-convex fat polytopes in d-dimensional space can

*Research is supported by the Dutch Organization for Scientific Research (N.W.O.) and by the ESPRIT III BRA Project 7141 (ALCOM II).

be preprocessed in time $O(n \log^{d-1} n \log\log n)$ into a data structure of size $O(n \log^{d-1} n)$ which supports point location queries and range searching queries with arbitrarily-shaped but bounded-size regions in time $O(\log^{d-1} n)$[1]. The data structure is based on Overmars' structure for point location in fat subdivisions [28]. Let us briefly review some relevant results in both point location and range searching to place our result in a broader perspective.

Point location in 2-space has been studied extensively and solved in a satisfactory way for many types of scenes, as several solutions achieve logarithmic query time and (near-)linear storage, after (near-)linear preprocessing time [12, 14, 20]. In higher-dimensional spaces, on the contrary, efficient solutions are available only for restricted problem instances. In 3-space, Chazelle [5] obtains $O(\log^2 n)$ query time and an $O(n)$ storage requirement for the case where the stored geometric objects are the 3-cells of a spatial subdivision, consisting of a total of n facets and satisfying the restrictive constraint that the vertical dominance relation on its cells is acyclic. Preparata and Tamassia [31] consider point location in a set of disjoint convex polyhedra with total complexity n. (The polyhedra subdivide $\mathbb{R}^3$ into a number of convex cells - the polyhedra - and a single non-convex cell - the complement of the polyhedra.) Their data structure uses $O(n \log^2 n)$ storage and is capable of answering point location queries in time $O(\log^2 n)$ after $O(n \log^2 n)$ preprocessing time. Goodrich and Tamassia [17] improve the storage requirement for sets of disjoint convex polyhedra to $O(n \log n)$ without affecting the query time. The results for arbitrary dimension d are further restricted, applying only to arrangements of hyperplanes or hypersurfaces of bounded degree. Clarkson [11] presents a data structure for point location in an arrangement of n hyperplanes in d-dimensional space. The structure supports queries in time $O(\log n)$ and requires roughly[2] $O(n^d)$ storage and preprocessing (improved to exactly $O(n^d)$ by Chazelle and Friedman [7] at the cost of an increase of the preprocessing time to $O(n^{2d+2})$). Chazelle et al. [6] achieve the same $O(\log n)$ query time for point location in arrangements of hypersurfaces of constant degree with a data structure of size roughly $O(n^{2d-3})$, which is computable in roughly $O(n^{2d-2})$ time. Apparently, efficient solutions for sets of non-convex objects or non-polyhedra in 3-space and for scenes other than arrangements of hyperplanes or hypersurfaces of bounded degree in higher-dimensional spaces are lacking.

Nearly all papers on range searching discuss how to preprocess a very elementary class of geometric objects, namely points, for efficiently answering range search queries with specific range types. The most extensively studied type of query range is the orthogonal range, and a long-established result says (see e.g. [30]) that a set consisting of n points can be preprocessed in time $O(n \log^{d-1} n)$ into a data structure of size $O(n \log^{d-1} n)$, which is capable of answering an orthogonal range query in time $O(\log^{d-1} n)$[3]. (Some small improvements are possible.) Different range types give rise to more complicated solutions. In general, the solutions to e.g. simplicial range searching only provide low query time at the cost of a relatively high storage requirement and preprocessing time or vice versa. At the one end, one finds solutions providing polylogarithmic query time and roughly $O(n^d)$ storage and preprocessing (see [9, 25]), whereas, at the other end of the spectrum,

[1] The result is a generalization of an earlier reported result [29], which is restricted to $d = 2, 3$.

[2] All quoted rough bounds are adequate up to a factor $n^{\epsilon} \log^{c} n$, for some arbitrarily small ϵ and some constant c.

[3] In all quoted time bounds we neglect the dependence of the query time on the size of the answer to the query. All results actually have the form $O(f(n) + h)$, where h is the output size.

solutions require only $O(n)$ storage, but guarantee only a larger query time of $O(n^{1-1/d})$ (see [24, 25]). Van Kreveld [21] gives similar bounds for the problem of reporting all simplices that are entirely contained in a query simplex. In between the previous results are the trade-off solutions which allow for exchanging storage for query time. An example of such a solution is given by Matoušek [25]: the presented structure supports queries in time $O((n/m^{1/d})\log^{d+1}(m/n))$ at the cost of an $O(m)$ storage requirement, where $n \leq m \leq n^d$. Alternative trade-off solutions provide similar bounds. More specific results (with respect to dimension) are reported by Chazelle and Welzl [10] who give a solution with $O(\sqrt{n}\log n)$ query time and $O(n)$ storage for triangular range searching in 2-space and a solution with $O(n^{2/3}\log^2 n)$ query time and a storage requirement of $O(n\log n)$ storage for tetrahedral range searching in 3-space. Semi-algebraic query ranges are considered by Agarwal and Matoušek [2], resulting in a data structure of size $O(n)$ with preprocessing time $O(n\log n)$ which supports range queries with a region , in d-space in time $O(n^{1-1/b+\delta})$, where δ is some arbitrarily small value and b is bounded by $d \leq b \leq 2d-3$, although its precise value depends on , .

Overmars [28] discusses a data structure for efficient and simple point location in fat subdivisions or sets of fat objects with total complexity n. The structure supports point location queries in time $O(\log^{d-1} n)$ and uses $O(n\log^{d-1} n)$ storage. In his paper, Overmars does not touch the issue of efficiently computing the data structure, that is, in time comparable to the storage requirement. We show here that for arbitrary convex objects and for non-convex polytopes, the structure can be built incrementally in time $O(n\log^{d-1} n\log\log n)$. Besides supporting efficient point location queries, the data structure storing the arbitrary convex objects and/or polytopes can, surprisingly, also be used for range searching with arbitrarily-shaped but bounded-size ranges. In fact, we show that each bounded-size range query can be implemented by a constant number of point location queries, thus leading to a time bound of $O(\log^{d-1} n)$ for range search queries.

The main contribution of our results with respect to the point location problem lies in their dimensional generality, where previous results in higher dimensions are restricted to arrangements of hyperplanes or hypersurfaces of bounded degree, and to severely restricted subdivisions and scenes of non-intersecting convex polyhedra in 3-space. We find that point location queries in scenes of non-intersecting (or mildly intersecting) fat arbitrary convex objects and/or non-convex polytopes can be performed in polylogarithmic time at the cost of near-linear storage and preprocessing. To see the contribution for range searching we recall the results by Van Kreveld [21] quoted earlier. A query with a simplex in d-space for all contained simplices takes logarithmic query time at the cost of roughly $O(n^d)$ storage or linear storage at the cost of polynomial query time. The solutions to range searching among points have similar performance. Although our range searching results apply only to fat objects and small query ranges, they succeed in combining polylogarthmic query time with near-linear storage and query time for simplicial range searching among arbitrary convex shapes and non-convex polytopes. Moreover, the data structure caters for arbitrary query ranges.

The paper is organized as follows. In Section 2, we give the definition of fatness we use, along with a key result for scenes of fat objects. Section 3 discusses Overmars' data structure for efficient point location in fat subdivisions or among fat objects, while Section 4 shows how the structure can be used for simple and efficient range searching among interesting classes of fat objects. The range searching results are used in Section 5 to support the incremental construction of the multi-purpose data structure, starting from

the largest object and repeatedly adding the next largest object. Finally, we summarize the results and point out the various potential generalizations.

2 Fatness

This section reveals the definition of fatness that is used throughout the paper and which was introduced in [32]. Contrary to many other definitions of fatness in literature [1, 3, 15, 19, 22, 26], the notion given below applies to general objects in arbitrary dimension d. The definition involves a parameter k, supplying a qualitative measure of the fatness of an object: the smaller the value of k, the fatter the object must be.

Definition 2.1 *Let $E \subseteq \mathbb{R}^d$ be an object and let k be a positive constant. The object E is k-fat if for all hyperspheres $S \in U_E$:*

$$k \cdot volume(E \cap S) \geq volume(S),$$

where U_E consists of all hyperspheres centered inside E and not fully containing E.

According to the definition, a k-fat object E must cover at least $1/k$-th of any hypersphere that is centered inside E but not fully contains it. If, in the sequel, we occasionally speak of a fat object, we actually mean a k-fat object, for some *constant* k. The definition of fatness forbids fat objects to be long and thin, or to have long and thin parts.

The fatness definition requires a fat object E to have a 'high density' in the vicinity of every point $p \in E$. As a result, it intuitively seems impossible to have many other fat objects (of a certain minimum size) in the vicinity of p. Theorem 2.2 formalizes this low object density property for scenes of non-intersecting fat objects. The theorem, as well as a few other results in this paper, requires a means of expressing the size of an object. We find the size, or more specifically the radius, of the minimal enclosing hypersphere of an object the most convenient among the many ways to express the object size. We are now ready to formulate the low object density property. The reader is referred to [34] for a proof of the theorem.

Theorem 2.2 *Let $\mathcal{E}$ be a set of non-intersecting k-fat objects in $\mathbb{R}^d$ with minimal enclosing hypersphere radii at least ρ, and let $c \in \mathbb{R}^+$ be a constant. Then the number of objects $E \in \mathcal{E}$ intersecting any region R with minimal enclosing hypersphere radius $c \cdot \rho$ is bounded by the constant $k \cdot (c+1)^d$.*

Informally, the theorem states that the number of k-fat objects intersecting a region, that is not too large compared to the objects, is constant. Range searching among k-fat objects with such regions is the subject of Section 4. Section 3 provides the tools for the efficient solution of these range queries.

3 Point location among fat objects

This section discusses a data structure for point location among disjoint fat objects by Overmars [28]. The author presents the data structure as a structure for solving the problem of point location in subdivisions of d-dimensional space into fat cells. The answer to the query is the specific cell containing the query point. The point location problem in fat subdivisions can be seen as an instance of the following, more general, formulation of the point location problem:

Given a set $\mathcal{E}$ of non-intersecting constant-complexity k-fat objects in $\mathbb{R}^d$ and a query point $p \in \mathbb{R}^d$, report the object $E \in \mathcal{E}$ that contains p, or report that no such object exists.

If the objects in $\mathcal{E}$ entirely cover $\mathbb{R}^d$ then we obtain a fat subdivision like in Overmars' paper. In the more general setting, the complement of the objects need not be fat, nor does it have constant complexity. Figure 1 shows two point location queries in a set of five non-intersecting fat objects. The query with the point p should yield the answer E_1, whereas the query with q must result in the answer that no object contains q. The ideas

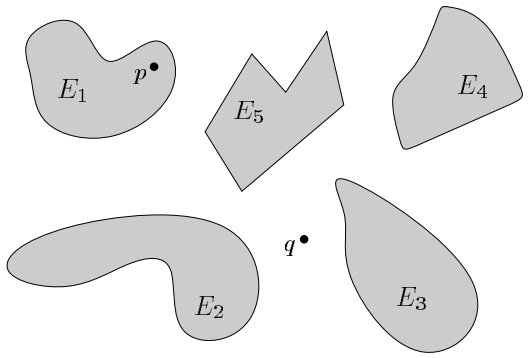

Figure 1: Two point location queries in the set $\{E_1, \ldots, E_5\}$; the query with p yields the answer E_1, while q is reported to lie in no object.

from [28] are discussed below in the context of the more general problem formulation.

Overmars' paper only presents a data structure for efficiently answering point location queries; the issue of building the structure remains untouched. Later in this paper, a solution for this problem is given for arbitrary convex objects and non-convex polytopes. The solution relies on the ability to do efficient range searching queries among these objects. Before discussing range searching and its application to building the point location structure, this section simply summarizes the results of [28].

Let us assume that the constant-complexity k-fat objects in $E_1, \ldots, E_n \in \mathcal{E}$ are ordered by radius of their minimal object enclosing hyperspheres. Let furthermore ρ_i be the radius of E_i's minimal enclosing hypersphere. Let us denote the axis-parallel enclosing hypercube of the minimal enclosing hypersphere of an object E_i by C_i. By construction, the hypercubes are ordered by increasing size. Note that the side length of the hypercube C_i is $2\rho_i$. Notice that the hypercubes C_i may be overlapping, although the objects E_i are disjoint. Furthermore, let V_i be defined as follows:

$$V_i = \{E_j \in \mathcal{E} | E_j \cap C_i \neq \emptyset \wedge j \geq i\}.$$

Hence, V_i is the set of objects that are larger than E_i, i.e., with larger minimal enclosing hypersphere, intersecting the box C_i. Theorem 2.2 immediately supplies a useful property of the sets V_i, $1 \leq i \leq n$.

Lemma 3.1 *For all $i : 1 \leq i \leq n :$ $|V_i| = O(1)$.*

A second crucial lemma from [28] is the following.

Lemma 3.2 *Let $p \in E_j$ and $i = \min\{h | p \in C_h\}$. Then $E_j \in C_i$.*

In words, the lemma states that if the query point p lies in an object E_j, then E_j is an element of the set of objects V_i associated to the smallest hypercube C_i containing p. This suggest the approach outlined below.

Assuming that the hypercubes C_i and the sets V_i for all $1 \leq i \leq n$ are available, we proceed as follows to find the answer to a point location query with a point $p \in \mathbb{R}^d$. Determine the smallest hypercube, if any, containing p. If no hypercube contains p than p lies in no object; if, on the contrary, C_i is the smallest hypercube containing p, then the set V_i supplies the answer to the query. To find it, we check the objects in V_i for containment of p. Note that the check for containment of a point in an object $E_j \in V_i$ takes constant time due to the constant complexity of the objects. The constant cardinality of V_i yields that the entire inspection of all objects in V_i takes $O(1)$ time. If no object in V_i contains p then no object in $\mathcal{E}$ contains p; otherwise the unique object $E_j \in V_i$ containing p obviously is the answer to the query. As the inspection of V_i takes constant time, the point location query time is dominated by the time to find the smallest hypercube C_i containing the query point p. An appropriate data structure that solves this problem is given below.

The point location problem is now essentially reduced to the following priority point stabbing problem among (intersecting) hypercubes:

> *Given a set of hypercubes $\mathcal{C}$ in $\mathbb{R}^d$ and a query point $p \in \mathbb{R}^d$, report the smallest hypercube $C \in \mathcal{C}$ containing p, or report that no hypercube in $\mathcal{C}$ contains p.*

To solve a priority point stabbing query among hypercubes, Overmars proposes a d-level data structure in which the upper $d-1$ levels are based on the segment tree and the lowest level is a list or a balanced binary tree.

$d = 1$ The hypercubes $C_1, \ldots, C_n$ are intervals on the real line. The interval endpoints partition the real line into $2n + 1$ so-called elementary intervals. All points within a single elementary interval are covered by exactly the same one-dimensional hypercubes. We organize the intervals as an ordered list and label each elementary interval with the index of the smallest of all one-dimensional hypercubes covering it. The list structure requires $O(n)$ storage. The answer to a point stabbing query is provided by the label of the elementary interval containing the query point. The interval can be identified in time $O(\log n)$.

$d > 1$ The d-dimensional hypercubes are stored in a segment tree T on their projections onto the d-th coordinate axis. The endpoints of the hypercube projections partition the d-th coordinate axis into a number of elementary intervals. An interval I_ν is associated with each node ν in T: I_ν is the union of all (consecutive) elementary intervals associated with the leaves of the subtree rooted at ν. With ν we store in an associated structure the intersections $[-\infty, \infty]^{d-1} \times I_\nu \cap C_i$ for hypercubes C_i that entirely span the slab $[-\infty, \infty]^{d-1} \times I_\nu$ but do not entirely span the slab $[-\infty, \infty]^{d-1} \times I_{\nu'}$ corresponding to the parent ν' of ν in T. The projection of such an intersection $[-\infty, \infty]^{d-1} \times I_\nu \cap C_i$ onto the subspace spanned by the first $d-1$ coordinate axes is a $(d-1)$-dimensional hypercube. We store these hypercubes in a (recursively defined) similar $(d-1)$-level data structure on the first $d-1$ coordinates, that is, if $d - 1 > 1$. If $d - 1 = 1$ we use the one-dimensional construction outlined

above to store the intersection of the squares and the planar slab. So, the bottom-level structure is a list or ordinary balanced binary tree instead of a segment tree.

Searching the multi-level data structure with a query point p proceeds in the following recursive manner: start at the root and repeatedly continue towards the child corresponding to the slab containing p. The search ends at the leaf corresponding to the elementary interval containing the last coordinate. The search is continued recursively in the substructures associated to the $O(\log n)$ nodes on the search path, each corresponding to a slab containing p. The entire search from top to bottom in the multi-level data structure takes therefore $O(\log^d n)$ time, resulting in $O(\log^{d-1} n)$ candidate answers. The minimum among these candidates is the final answer to the query. The query time can be improved by applying fractional cascading [8] to the two lower levels of the data structure. This is possible because the bottom-level structures are ordered lists (a sequence of intervals). Fractional cascading improves the query time in a 2-level data structure consisting of a segment tree with the one-dimensional ordered lists as substructures from $O(\log^2 n)$ to $O(\log n)$. Hence, a priority point stabbing query among hypercubes takes $O(\log^{d-1} n)$ time. The structure uses $O(n \log^{d-1} n)$ storage. The result is summarized in the following theorem.

Theorem 3.3 *A set $\mathcal{E}$ of non-intersecting constant-complexity k-fat objects in $\mathbb{R}^d$ can be stored in a data structure of size $O(n \log^{d-1} n)$, such that, for a query point $p \in \mathbb{R}^d$, it takes $O(\log^{d-1} n)$ time to report the object $E \in \mathcal{E}$ that contains p, or to conclude that no object contains p.*

The remaining open problem concerns the preprocessing phase, that is, the computation of the data structure: given a set of non-intersecting k-fat constant-complexity objects $E_1, \ldots, E_n$ ordered by increasing radii of their minimal enclosing hyperspheres, compute the multi-level data structure storing the enclosing hypercubes $C_1, \ldots, C_n$ of the minimal enclosing hyperspheres of these objects plus the sets $V_1, \ldots, V_n$ of larger objects intersecting the respective hypercubes $C_1, \ldots, C_n$. Building the multi-level data structure can be accomplished in time $O(n \log^{d-1} n)$ using standard techniques. Another option, that is exploited in Section 5, is to build the structure in an incremental way. It is well-known that a hypercube can be inserted in a d-level segment tree in time $O(\log^d n)$. Moreover, if we use dynamic fractional cascading [27] instead of 'regular' fractional cascading, the insertion time can be further reduced to $O(\log^{d-1} n \log\log n)$ (see Section 5).

The computation of the sets V_i, on the other hand, seems to pose more problems. Finding the objects E_j with $j \geq i$ intersecting the hypercube C_i requires a range search query with C_i. The query is not an ordinary range search query, since we are only interested in objects with a certain minimal size (or index). Performing a range search query among all objects and subsequently filtering out the smaller objects is not a good idea, as the answer to the range query might by orders of magnitude larger than V_i. A better idea would be to perform a range search query with V_i only among objects that are larger than E_i. Surprisingly, we show in the next section that it is possible, in most interesting cases, to use the point location structure itself for solving the range search query. This suggests an approach where we add the hypercubes from large to small, meanwhile computing the sets V_i in the following (incremental) way: use, before insertion of the hypercube C_{n-m}, the sets $V_{n-m+1}, \ldots, V_{n-m}$ and the multi-level data structure storing the hypercubes $C_{n-m+1}, \ldots, C_n$ to compute V_{n-m} by a range query with C_{n-m} among the

objects $E_{n-m+1}, \ldots, E_n$. Next, insert C_{n-m} into the structure and continue with C_{n-m-1}. Section 5 contains the details of the approach.

4 Range searching by point location

In this section we use the point location data structure to tackle the following general version of the range searching problem:

> *Given a set $\mathcal{E}$ of non-intersecting constant-complexity k-fat objects in $\mathbb{R}^d$ with minimal enclosing hyperspheres with radii at least ρ and a constant-complexity query region R of arbitrary shape with diameter at most $h \cdot \rho$ for some positive constant h, report all objects $E \in \mathcal{E}$ that intersect R.*

Figure 2 shows a bounded-size range query in a set of five non-intersecting fat objects. The query with the range R must yield the answer $\{E_2, E_3, E_5\}$. Using Theorem 2.2 it

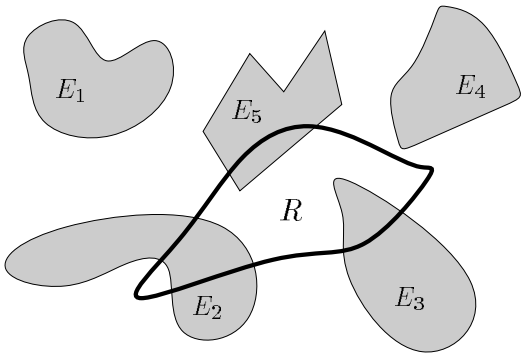

Figure 2: A range query in the set $\{E_1, \ldots, E_5\}$; the query with R yields the answer $\{E_2, E_3, E_5\}$.

is easy to verify that the answer to the query with a region R, satisfying the diameter bound, is a set of objects of constant cardinality. Let us define the set $Q(R)$ of objects intersecting the region R:

$$Q(R) = \{E \in \mathcal{E} | E \cap R \neq \emptyset\}.$$

We show how the point location structure can be used to solve the range searching problem in time $O(\log^{d-1} n)$ in the case that $\mathcal{E}$ is a set of arbitrary convex objects and/or non-convex polytopes. The solution relies on local properties of fat objects. The definition of fatness requires a k-fat object E to have a large 'density' in the vicinity of any point p in the object: $1/k$-th of a hypersphere centered at p is covered by E. It turns out that this property makes it possible to hit any object or object part with a certain minimum size, regardless of its exact location, with at least one point from a sufficiently dense, but not too large, pattern of sample points, whereas this would clearly be impossible if the object is non-fat: the chance to hit a line segment by an extremely dense pattern of sample points is practically zero. To structure the problem and the shape of its solution, we restrict the sample points to be arranged as a regular orthogonal grid.

Definition 4.1 *A regular orthogonal grid $\mathcal{G}(r)$ with resolution r is defined by:*

$$\mathcal{G}(r) = \{(z_1 r, \ldots, z_d r) | z_1, \ldots, z_d \in \mathbf{Z}\}.$$

This section focuses on the problem of finding a grid resolution such that a small subset of the corresponding regular orthogonal grid is guaranteed to hit any object E having non-empty intersection with the query region R. We shall first determine a suitable grid resolution for convex objects, and subsequently use the results obtained there to find an appropriate resolution for general polytopes.

The main implication of these results is that the range query with the bounded-size range $R \subseteq \mathbb{R}^d$ can be solved by a sequence of point location queries, each taking $O(\log^{d-1} n)$ time, using the data structure for point location among fat objects. Under the assumption that the diameter of the query region does not exceed $h \cdot \rho$, for some constant $h \geq 0$, the sequence of point location queries will have constant length.

The aim is to find a grid resolution r establishing that each object $E \in Q(R)$ is hit by at least one point in some subset $\Pi \subset \mathcal{G}(r)$, where, preferably, the size of Π depends on k and h (and the complexity of the individual objects) only. Before we focus on the different types of objects, we first give some basic results that ease the task to find a grid resolution.

To simplify the approach, we will define for each object a large hypersphere that is contained in the object $E \in Q(R)$. For the hypersphere, it is easy to determine the grid resolution r such that the hypersphere is always hit by at least one of the grid points.

Property 4.2 *Any hypersphere with radius at least $\frac{1}{2} r\sqrt{d}$ contains at least one point of the orthogonal grid $\mathcal{G}(r)$.*

The hypersphere itself is determined in two steps. First, a result by Leichtweiß [23] makes it possible to identify a large ellipsoid inside a convex part of the object E. Due to the fatness of the object E, the ellipsoid, in turn, indeed contains a large hypersphere.

Ellipsoids play an important role throughout this section. Let us therefore briefly review some relevant properties. Any ellipsoid $L \subseteq \mathbb{R}^d$ can be regarded as a translated and rotated copy of an ellipsoid in so-called standard position. An ellipsoid L_s in standard position has the form

$$L_s : \sum_{i \in \{1,\ldots,d\}} \frac{x_i^2}{a_i^2} \leq 1,$$

where $a_1, \ldots, a_d$ are constants. The segment connecting the points $0^{i-1} \times -a_i \times 0^{d-i} \in L_s$ and $0^{i-1} \times a_i \times 0^{d-i} \in L_s$, which has length $2a_i$, is referred to as an axis of L_s; L_s has d such axes. If $w \leq a_i$ for all $1 \leq i \leq d$, then the hypersphere with radius w centered at the origin is entirely contained in L_s (see Figure 3). The volume of an ellipsoid can be given as a function of the lengths of its (half-)axes; the volume of L_s (and of its translated and rotated copies L) is given by (see e.g. [35])

$$\mathit{volume}(L_s) = \omega_d \cdot \prod_{i \in \{1,\ldots,d\}} a_i,$$

where ω_d is the dimension-dependent constant multiplier from the volume formulae for hyperspheres[4].

[4]For even dimension $\omega_d = \omega_{2m} = \frac{\pi^m}{m!}$. For odd dimension $\omega_d = \omega_{2m+1} = \frac{2(2\pi)^m}{(2m+1)!!}$. See, e.g., [16, Section 394].

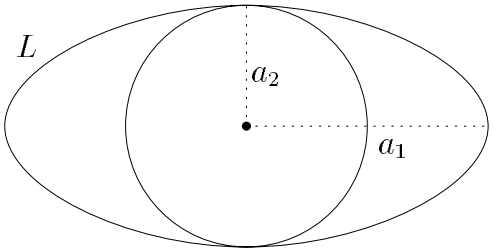

Figure 3: The ellipse L with half-axes $a_1 \geq a_2$ contains a circle with radius a_2.

A lower bound V on the volume of an ellipsoid L alone does not suffice to prove that L encloses a large hypersphere, as it is possible to construct a 'very long and thin' ellipsoid. An additional upper bound on the diameter of the ellipsoid, and, hence, on the a_i's ($1 \leq i \leq d$), however, makes such a construction impossible. This follows easily from the volume formula for ellipsoids.

Lemma 4.3 *Let $L \subseteq \mathbb{R}^d$ be an ellipsoid with volume$(L) \geq V$ and let δ be an upper bound on its diameter. Then L contains a hypersphere with radius at least $\frac{V}{\omega_d} \cdot (2/\delta)^{d-1}$.*

Proof: Assume without loss of generality that L is in standard position, and, hence, of the form $\sum_{i \in \{1,\ldots,d\}} \frac{x_i^2}{a_i^2} = 1$. Then, its volume is $volume(L) = \omega_d \cdot \prod_{i \in \{1,\ldots,d\}} a_i$. The upper bound of δ on the diameter of L implies that none of the axes of L is longer than δ, and, thus, for all $1 \leq i \leq d$:

$$a_i \leq \delta/2.$$

Now assume, for a contradiction, that $a_j < \frac{V}{\omega_d} \cdot (2/\delta)^{d-1}$. Subsequent application of this inequality and the upper bound $a_i \leq \delta/2$ ($1 \leq i \leq d$) yields

$$\begin{aligned} volume(L) &= \omega_d \cdot a_j \cdot \prod_{i \in \{1,\ldots,d\}, i \neq j} a_i \\ &< \omega_d \cdot \frac{V}{\omega_d} \cdot (2/\delta)^{d-1} \cdot \prod_{i \in \{1,\ldots,d\}, i \neq j} a_i \\ &\leq \omega_d \cdot \frac{V}{\omega_d} \cdot (2/\delta)^{d-1} \cdot (\delta/2)^{d-1} \\ &= V, \end{aligned}$$

contradicting the assumption $volume(L) \geq V$.

The ellipsoid L entirely contains the hypersphere with radius $\frac{V}{\omega_d} \cdot (2/\delta)^{d-1}$ centered at the origin. □

In the two subsections below, we use the property and lemma to find a valid grid resolution, and identify a small subset of that grid that suffices to hit all objects intersecting the query region $R \subseteq \mathbb{R}^d$.

4.1 Searching among convex objects

Let $E \subseteq \mathbb{R}^d$ be a convex k-fat object intersecting the query region $R \subseteq \mathbb{R}^d$, and let $m \in E \cap R$. The hypersphere $S_{m,\rho}$ with center m and radius ρ belongs to U_E as it is

centered inside E and can impossibly have E entirely in its interior. The membership $S_{m,\rho} \in U_E$ and the k-fatness of E yield:

$$k \cdot volume(E \cap S_{m,\rho}) \geq volume(S_{m,\rho}) = \omega_d \cdot \rho^d. \tag{1}$$

The shape $E \cap S_{m,\rho}$ is convex as it is the intersection of the convex objects E and $S_{m,\rho}$. The following result due to Leichtweiß [23] holds for any convex shape.

Lemma 4.4 *Let $E \subset \mathbb{R}^d$ be a convex object. There exist ellipsoids $L^I, L^O \subseteq \mathbb{R}^d$ such that $L^I \subseteq E \subseteq L^O$ and*

$$d^d \cdot volume(L^I) \geq volume(L^O).$$

Corollary 4.5 is a trivial consequence of Lemma 4.4.

Corollary 4.5 *Any convex object $E \subseteq \mathbb{R}^d$ contains an ellipsoid L with*

$$d^d \cdot volume(L) \geq volume(E).$$

Application of Corollary 4.5 to the shape $E \cap S_{m,\rho}$, satisfying (1), implies the containment of an ellipsoid $L \subseteq E \cap S_{m,\rho}$ such that

$$volume(L) \geq \frac{\omega_d \rho^d}{k d^d}. \tag{2}$$

The diameter of L is bounded by 2ρ, because L is contained in the hypersphere $S_{m,\rho}$ with diameter 2ρ. The application of Lemma 4.3 to the ellipsoid L with diameter at most 2ρ and volume bounded by (2) yields that L contains a hypersphere S with radius at least $k^{-1}d^{-d}\rho$. Property 4.2 subsequently implies that such a hypersphere is hit by at least one point from the regular orthogonal grid with resolution $2k^{-1}d^{-(d+\frac{1}{2})}\rho$, or $\mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho)$. Hence, at least one point from $\mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho)$ hits $S \subseteq L \subseteq E \cap S_{m,\rho}$.

So far, we have only bothered about finding a sufficiently high resolution for a grid to hit all objects $E \in Q(R)$. Clearly, it is unnecessary and even undesirable to perform point location queries with a too large subset of the grid, both because it increases the query time and because it would lead to many accidental hits of objects $E \notin Q(R)$. Fortunately, the size of the sample set (and the number of accidental hits) can be adequately limited by a quick glance at the computation of the grid resolution: the resolution is chosen such that any object $E \in Q(R)$ is hit inside some hypersphere $S_{m,\rho}$ with $m \in R$. This hypersphere lies entirely inside the region $R \ominus S_{O,\rho}$, where O is the origin of the Euclidean coordinate frame and $\ominus$ denotes the Minkowski difference operator. The Minkowski difference of two sets A and B is defined by $A \ominus B = \{a - b | a \in A \wedge b \in B\}$. Hence, at least one of the grid points hitting E lies in $R \ominus S_{O,\rho}$. As a result, it suffices to restrict the set of sample points Π to be the set of grid points in the grown query region:

$$\Pi = \mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho) \cap (R \ominus S_{O,\rho}).$$

The result is summarized in the following lemma.

Lemma 4.6 *Let $\mathcal{E}$ be a set of convex k-fat objects $E \subseteq \mathbb{R}^d$ with minimal enclosing hyperspheres with radii at least ρ and let $R \subseteq \mathbb{R}^d$ be a region with diameter $h \cdot \rho$, for some constant h. The set $Q(R)$ of objects $E \in \mathcal{E}$ intersecting R can be found by point location queries with the points from $\mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho) \cap (R \ominus S_{O,\rho})$.*

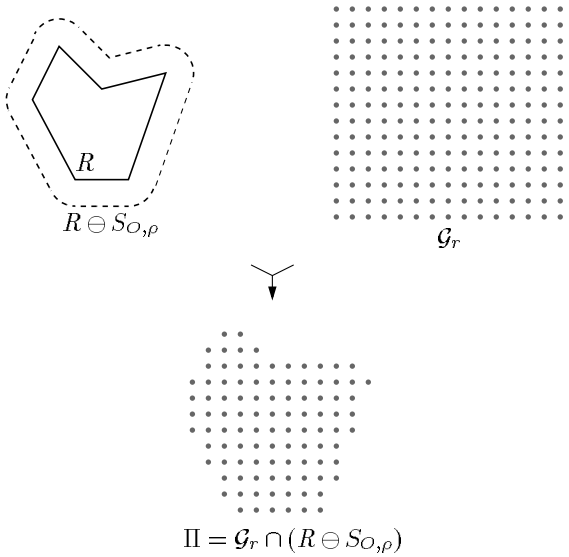

Figure 4: A two-dimensional example of the construction of the set of sample points Π for a query with a region R with diameter $h \cdot \rho$ among a set $\mathcal{E}$ of objects with minimal enclosing circle radius ρ. The resolution r of the orthogonal grid is determined by the type of the objects in $\mathcal{E}$.

The data structure presented in the previous section allows us to perform point location queries among the objects of $\mathcal{E}$ with all points in Π in time $O(|\Pi| \log^{d-1} n)$. The resulting set $\{E \in \mathcal{E} | \Pi \cap E \neq \emptyset\}$ of query answers, which clearly has at most $|\Pi|$ elements, is a superset of the answer $Q(R)$ to the range query with R. For each of the objects $E \in \{E \in \mathcal{E} | \Pi \cap E \neq \emptyset\}$, additional constant time suffices to verify the membership $E \in Q(R)$ by a simple test for the non-emptiness of $E \cap R$, provided that R and all $E \in \mathcal{E}$ have constant complexity. Hence, the computation of $Q(R)$ takes $O(|\Pi| \log^{d-1} n)$ time.

It remains to bound the size of Π. Clearly, the Minkowski difference $R \ominus S_{O,\rho}$ fits entirely in the hypercube $[x_0^R - \rho, x_0^R + (h+1)\rho] \times \ldots \times [x_d^R - \rho, x_d^R + (h+1)\rho]$, where x_i^R $(1 \leq i \leq d)$ is the minimal x_i-coordinate occurring in R. As a result, the number of elements in Π is bounded by the number of grid points in the enclosing hypercube, leading to

$$\begin{aligned}
|\Pi| &= |\mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho) \cap (R \ominus S_{O,\rho})| \\
&\leq |\mathcal{G}(2k^{-1}d^{-(d+\frac{1}{2})}\rho) \\
&\quad \cap [x_0^R - \rho, x_0^R + (h+1)\rho] \times \ldots \times [y_d^R - \rho, y_d^R + (h+1)\rho]| \\
&= (\frac{1}{2}kd^{d+\frac{1}{2}}(\lfloor h \rfloor + 2))^d \\
&= O((kd^d h)^d)
\end{aligned}$$

In a setting where all objects are k-fat for some constant k and the diameter of the query region R does not exceed a constant multiple h of ρ it follows that $|\Pi| = O(1)$, which implies an $O(\log^{d-1} n)$ time bound for range searching with a bounded-size region R.

Theorem 4.7 *Let $k > 0$ and $h \geq 0$ be constants, and let $\mathcal{E}$ be a set of convex k-fat constant-complexity objects $E \subseteq \mathbb{R}^d$ with minimal enclosing hyperspheres with radii at least ρ. A range searching query with a region $R \subseteq \mathbb{R}^d$ of diameter at most $h \cdot \rho$ among $\mathcal{E}$ takes $O(\log^{d-1} n)$ time.*

4.2 Searching among polytopes

Having solved the range searching problem among convex objects we now turn our attention to (non-convex) polytopes. We assume that all polytopes $E \subseteq \mathbb{R}^d$ in $\mathcal{E}$ are bounded by c hyperplanar faces, that is, each face is part of a $(d-1)$-dimensional hyperplane. Let E be a k-fat polytope intersecting R, and let $m \in E \cap R$. Inequality (1) applies to the intersection of the polytope E and the hypersphere $S_{m,\rho} \in U_E$, on exactly the same grounds as in the convex case. Unfortunately, the intersection $E \cap S_{m,\rho}$ is not a polytope as its boundary contains portions of the hyperspherical boundary of $S_{m,\rho}$. This can be remedied by replacing $S_{m,\rho}$ by its (axis-parallel) enclosing hypercube $C_{m,\rho}$ (with 'center' m and side length 2ρ) with $\mathit{volume}(C_{m,\rho})/\mathit{volume}(S_{m,\rho}) = 2^d/\omega_d$. The ratio of the volumes, the inequality (1), and the obvious inequality $\mathit{volume}(E \cap C_{m,\rho}) \geq \mathit{volume}(E \cap S_{m,\rho})$, together yield

$$\frac{2^d k}{\omega_d} \cdot \mathit{volume}(E \cap C_{m,\rho}) \geq \mathit{volume}(C_{m,\rho}) = 2^d \rho^d. \tag{3}$$

The intersection $E \cap C_{m,\rho}$ is a collection of polytopes. The arrangement of the $c + 2d$ supporting hyperplanes of the c faces of E and the $2d$ faces of the hypercube $C_{m,\rho}$ subdivide $\mathbb{R}^d$ and, more importantly, $E \cap C_{m,\rho}$ into convex regions: the d-cells of the arrangement. Edelsbrunner's book on combinatorial geometry [13] supplies bounds on the numbers of faces of various dimensions in arrangements of hyperplanes. Lemma 4.8 reproduces the bounds.

Lemma 4.8 *The maximum number $f_k^{(d)}(n)$ of k-faces in an arrangement of n hyperplanes in $\mathbb{R}^d$ is given by*

$$f_k^{(d)}(n) = \sum_{i \in \{0,\ldots,k\}} \binom{d-i}{k-i} \binom{n}{d-i}.$$

We are interested in the maximum number of d-faces in an arrangement of $c + 2d$ hyperplanes in $\mathbb{R}^d$, or $f_d^{(d)}(c+2d)$ for short. By Lemma 4.8, we have that

$$\begin{aligned} f_d^{(d)}(c+2d) &= \sum_{i \in \{0,\ldots,d\}} \binom{d-i}{d-i} \binom{c+2d}{d-i} \\ &= \sum_{j \in \{0,\ldots,d\}} \binom{c+2d}{j} \\ &\leq \frac{1}{2} \cdot \sum_{j \in \{0,\ldots,c+2d\}} \binom{c+2d}{j} \\ &\leq 2^{c+2d-1}. \end{aligned}$$

The basis of the first of the above inequalities lies in the simple observation that $d \leq \frac{1}{2}(c+2d)$. Note that the bound of 2^{c+2d-1} on the number of d-faces is probably not very tight, as the $c + 2d$ hyperplanes include many parallel pairs of hyperplanes. The $c + 2d$

hyperplanes subdivide the collection of polytopes $E \cap C_{m,\rho}$ into $g \leq 2^{c+2d-1}$ convex regions. The largest region $E' \subseteq E \cap C_{m,\rho}$ of these g convex regions clearly satisfies

$$\begin{aligned} \mathit{volume}(E') &\geq \frac{1}{g} \cdot \mathit{volume}(E \cap C_{m,\rho}) \geq \frac{1}{2^{c+2d-1}} \cdot \mathit{volume}(E \cap C_{m,\rho}) \\ &\geq \frac{\omega_d \rho^d}{2^{c+2d-1}k}. \end{aligned} \tag{4}$$

The convexity of the subshape $E' \subseteq E \cap C_{m,\rho}$ allows for the subsequent application of Corollary 4.5, Lemma 4.3, and Lemma 4.2. First of all, Corollary 4.5 tells us that the convex shape E' contains an ellipsoid L with

$$\mathit{volume}(L) \geq \frac{\omega_d \rho^d}{2^{c+2d-1}kd^d}. \tag{5}$$

As the ellipsoid L lies entirely inside the hypercube $C_{m,\rho}$, the diameter of L is bounded by $2\rho\sqrt{d}$. Lemma 4.3 now implies that L contains a hypersphere S with radius at least $2^{-(c+2d-1)}k^{-1}d^{-\frac{1}{2}(3d-1)}\rho$. Property 4.2, finally, shows that any such hypersphere S is hit by at least one point from the regular orthogonal grid with resolution $2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho$. Hence, at least one point from $\mathcal{G}(2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho)$ hits $S \subseteq L \subseteq E' \subseteq E \cap C_{m,\rho}$. Notice that in the case of polytopes, unlike for convex objects, the required grid resolution depends on the complexity c of the objects of $\mathcal{E}$.

By the considerations of the previous paragraphs, one of the grid points that hit any object $E \in Q(R)$ lies inside a hypercube $C_{m,\rho}$ with $m \in E \cap R$. This hypercube must therefore lie completely inside the Minkowski difference $R \ominus C_{O,\rho}$. As a consequence, the set Π may be restricted to

$$\Pi = \mathcal{G}(2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho) \cap (R \ominus C_{O,\rho}).$$

Lemma 4.9 summarizes the results obtained so far in this subsection.

Lemma 4.9 *Let $\mathcal{E}$ be a set of k-fat polytopes $E \subseteq \mathbb{R}^d$ bounded by c hyperplanar faces and with minimal enclosing hyperspheres with radii at least ρ. Furthermore, let $R \subseteq \mathbb{R}^d$ be a region with diameter $h \cdot \rho$, for some constant h. The set $Q(R)$ of objects $E \in \mathcal{E}$ intersecting R can be found by point location queries with the points from $\mathcal{G}(2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho) \cap (R \ominus S_{O,\rho})$.*

Similar to the convex case, the sequence of point location queries with all points in Π takes $O(|\Pi| \log^{d-1} n)$ time and results in the set $\{E \in \mathcal{E} | \Pi \cap E \neq \emptyset\}$. The extraction of $Q(R)$ from this set takes $O(|\Pi|)$ time under the additional assumption that R and all $E \in \mathcal{E}$ have constant complexity, so c must be constant. To bound the number of elements in Π, we notice that $R \ominus C_{O,\rho}$ also fits completely in the hypercube $[x_0^R - \rho, x_0^R + (h+1)\rho] \times \ldots \times [x_d^R - \rho, x_d^R + (h+1)\rho]$, where x_i^R $(1 \leq i \leq d)$ is once again the minimal x_i-coordinate in R. The number of grid points in the hypercube bounds the number of elements in Π:

$$\begin{aligned} |\Pi| &= |\mathcal{G}(2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho) \cap (R \ominus C_{O,\rho})| \\ &\leq |\mathcal{G}(2^{-(c+2d-2)}k^{-1}d^{-\frac{3}{2}d}\rho) \\ &\quad \cap [x_0^R - \rho, x_0^R + (h+1)\rho] \times \ldots \times [y_d^R - \rho, y_d^R + (h+1)\rho]| \\ &= (2^{c+2d-2}kd^{\frac{3}{2}d}(\lfloor h \rfloor + 2))^d \\ &= O((2^c k d^d h)^d) \end{aligned}$$

If, besides c, the parameters k and h are also constant, then we get $|\Pi| = O(1)$, which induces polylogarithmic query time for range searching with bounded-size ranges among fat objects.

Theorem 4.10 *Let $k > 0$ and $h \geq 0$ be constants, and let $\mathcal{E}$ be a set of k-fat constant-complexity polytopes $E \subseteq \mathbb{R}^d$ with minimal enclosing hyperspheres with radii at least ρ. A range searching query with a region $R \subseteq \mathbb{R}^d$ of diameter at most $h \cdot \rho$ among $\mathcal{E}$ takes $O(\log^{d-1} n)$ time.*

5 Building the data structure

The results of the previous section can be used for the incremental construction of the point location (and range searching) data structure. Let us assume we are given the d-level data structure for priority point stabbing queries among hypercubes from Section 3, storing the m largest hypercubes $C_{n-m+1}, \ldots, C_n$, and the corresponding constant-cardinality sets $V_{n-m+1}, \ldots, V_n$. We refer to this partial priority point stabbing structure as T_m. Hence, the objective is to eventually compute T_n from some initial structure T_0. The outline of the incremental construction is as follows.

compute T_0;
$m := 0$;
while $m < n$ **do**
1. compute V_{n-m} by a range query with C_{n-m} (using T_m and the sets V_j, $n - m < j \leq n$);
2. compute T_{m+1} by inserting C_{n-m} into T_m.

We study both steps in the loop in more detail, starting with the second step, as the implications of its solution influence the first step as well.

Computation of T_{m+1}

The problem with the insertion of a hypercube into the d-level priority point stabbing structure lies in the use of fractional cascading, which was incorporated to improve the point location and range search query time from $O(\log^d n)$ to $O(\log^{d-1} n)$. Unfortunately, insertions into the multi-level data structure do not benefit from fractional cascading, so an insertion into the structure T_m would require $O(\log^d n)$ time instead of $O(\log^{d-1} n)$. Moreover, a sequence of insertions into the multi-level data structure with the static fractional cascading part is likely to increase the time for a query back to $O(\log^d n)$ as the fractional cascading part no longer 'suits' the updated multi-level data structure. Building the data structure would, even with fractional cascading, require $O(n \log^d n)$ time. Fortunately, Mehlhorn and Näher describe in [27] a dynamic version of fractional cascading. Incorporation of dynamic fractional cascading in the data structure during the construction phase improves the preprocessing time to $O(n \log^{d-1} n \log\log n)$. The alternative requires only minor modifications. We give the main result from [27] in a formulation that is tailored to our applications.

Theorem 5.1 *Let T be a tree with t nodes and let an ordered list $L(\nu)$ of elements from a given domain D be associated to each node ν. Furthermore, define l to be the total length of all lists $L(\nu)$, so $l = \sum_\nu |L(\nu)|$.*

(a) *Let T' be a connected subtree of T with s nodes. Location of a query value x in $L(\nu')$ for every $\nu' \in T'$, that is, finding the position in $L(\nu')$ of the smallest value larger or equal than x, takes $O(\log(l+t) + s \log\log(l+t))$ time worst-case.*

(b) *The deletion of a value x from a list $L(\nu)$ takes, given x's position in $L(\nu)$, amortized time $O(\log\log(l+t))$.*

(c) *The insertion of a value x from a list $L(\nu)$ takes, given the position in $L(\nu)$ of the smallest value larger than x, amortized time $O(\log\log(l+t))$.*

To simplify the process of incrementally building the objective structure T_n, we observe that all hypercubes that are to be added throughout the construction process can be computed in advance. This observation facilitates a less complex semi-dynamic (instead of dynamic) preprocessing. The prior knowledge of all n hypercubes means that the endpoints of the projections of the hypercubes on the i-th coordinate axis are from a fixed finite universe U_i of size $O(n)$. We act, however, as if we only know the projections of the hypercubes on the last $d-1$ coordinate axes; hence each corner $(x_1, x_2, \ldots, x_d)$ is assumed to be a point from $\mathbb{R} \times U_2 \times \ldots \times U_d$. The static nature of the hypercubes with respect to the last $d-1$ axes is used to compute the major part of the d-level data structure in advance by recursively building a segment tree on the projections of the hypercubes onto the last $d-1$ coordinate axes. The resulting $(d-1)$-level segment tree, which can be regarded as the initial tree T_0 in our incremental construction, differs from our objective d-level data structure, T_n, only in that the one-dimensional ordered lists in the nodes of the substructures at level $d-1$ are missing. These lists are built incrementally by 'inserting' the hypercubes from large to small into the skeleton provided by the $(d-1)$-level segment tree. Note that the substructures at level $d-1$ represent decompositions into vertical slabs of the plane spanned by the second and first coordinate axes.

Let us now consider the intermediate structure T_m, obtained after inserting the largest m hypercubes $C_{n-m+1}, \ldots, C_n$ into the skeleton T_0. Following the standard insertion procedure for multi-level segment trees, the insertion of the hypercube C_{n-m} into T_m boils down to the insertion of a square, that is, the projection of C_{n-m} onto the plane spanned by the first two coordinate axes, into $O(\log^{d-2} n)$ substructures T at level $d-1$. Note that the prior computation of the skeleton T_0 guarantees that no new nodes have to be created in any of the higher-level structures during the insertion of a hypercube.

We move on to study the reduced problem of inserting the planar projection $[i_1^L, i_1^H] \times [i_2^L, i_2^H]$ of C_{n-m} into a level-$(d-1)$ substructure T of T_m. The upper level of T is a segment tree on the projections onto the second coordinate axis of all, at most n, planar hypercube projections stored in T. The associated (ordered list) structure $L(\nu)$ of a node ν of T stores the sequence (from $x_1 = -\infty$ to $x_1 = +\infty$) of disjoint slab intervals $[\alpha, \beta] \times I_\nu$ within which all points share the same smallest containing hypercube. The intervals are labeled with the respective hypercube index.

The hypercube C_{n-m} must be stored in the ordered lists $L(\nu)$ at nodes ν of T corresponding to intervals I_ν that are entirely spanned by $[i_2^L, i_2^H]$ and have parents $parent(\nu)$ corresponding to intervals $I_{parent(\nu)}$ that are not spanned by $[i_2^L, i_2^H]$. Although the resulting nodes do not form a connected subtree of T they are in fact never more than one node off the search path from root to leaf in T for either the endpoint i_2^L or the endpoint i_2^R. Hence, we can apply Theorem 5.1(a) to the connected subtree T' of T consisting of all nodes on and just off both search paths and therefore efficiently search all $L(\nu')$ for ν' in

T' simultaneously for the location of the left endpoint i_1^L of the projection of C_{n-m} onto the first coordinate axis. The search time depends (see Theorem 5.1) on the number of nodes (s) in the connected subtree T', the cumulative length (l) of all associated ordered lists in T, and the number of nodes (t) in T. First of all, the tree T is a priorly built tree on a subset of the projections of all hypercubes $C_1, \ldots, C_n$, so $t = O(n)$. The subtree T' consists of two root-leaf paths in T plus all nodes that are only one node off these search paths, thus $s = O(\log n)$. Moreover, an ordered list $L(\nu)$ (before insertion of C_{n-m}) stores only projections of the m hypercubes $C_{n-m+1}, \ldots, C_n$, so $|L(\nu)| = O(m)$. Because (a part of) each projection appears at no more than two nodes at a single height-level in T, we have $l = \sum_\nu |L(\nu)| = O(m \log m) = O(n \log n)$. Application of Theorem 5.1(a) to the subtree T' of T and the query value i_1^L yields that the location of the query value is identified in all lists $L(\nu')$ for ν' in T' in time $O(\log n \log\log n)$ worst-case. Note that the set of nodes in T' is a superset of the set of nodes in whose associated substructures C_{n-m} must be inserted.

The problem that remains is to, given the interval $[i_1^L, i_1^H]$ and pointers to the location of i_1^L in all lists $L(\nu)$ with ν in T', insert the interval $[i_1^L, i_1^H]$ only into the lists $L(\nu)$ of nodes in which C_{n-m} must be inserted, i.e., $[i_2^L, i_2^H]$ spans I_ν but not $I_{parent(\nu)}$. Let us consider a node ν in T'. Verifying whether $[i_1^L, i_1^H]$ must be inserted into $L(\nu)$ is easily done in constant time by comparing $[i_2^L, i_2^H]$ with I_ν and $I_{parent(\nu)}$. Assume that $[i_1^L, i_1^H]$ must indeed be inserted into $L(\nu)$, labeled with the index '$n-m$'. After insertion of the interval, a query with a point $p \in [i_1^L, i_1^H] \times I_\nu$ for the smallest, or lowest indexed, covering hypercube (projection) must obviously yield the answer '$n-m$'. Hence, the interval $[i_1^L, i_1^H]$ must overwrite all parts of intervals that have non-empty intersection with $[i_1^L, i_1^H]$ and are present in $L(\nu)$ upon insertion of the latter interval. Using the pointer into $L(\nu)$ we can identify the subsequence of intervals that have non-empty intersection with the interval $[i_1^L, i_1^H]$ in time proportional to its length. Let $[\alpha_1, \beta_1], \ldots, [\alpha_g, \beta_g]$ be this sequence and note that only $[\alpha_1, \beta_1]$ may contain i_1^L and only $[\alpha_g, \beta_g]$ may contain i_1^H. The update of $L(\nu)$ proceeds in four simple steps, in which we scan all intervals intersected by $[i_1^L, i_1^H]$:

1. **if** $i_1^L \in (\alpha_1, \beta_1]$ **then** replace $[\alpha_1, \beta_1]$ by $[\alpha_1, i_1^L]$
   else delete $[\alpha_1, \beta_1]$;
2. **for all** $2 \leq h \leq g-1$ **do delete** $[\alpha_h, \beta_h]$;
3. **if** $i_1^H \in [\alpha_g, \beta_g)$ **then** replace $[\alpha_g, \beta_g]$ by $[i_1^H, \beta_g]$
   else delete $[\alpha_g, \beta_g]$;
4. **insert** $[i_1^L, i_1^H]$.

As $l = O(n \log n)$ and $t = O(n)$, the amortized time spent on each of the above deletions or insertions is $O(\log\log n)$, by Theorem 5.1(b,c). The four steps lead to one insertion and g deletions in $L(\nu)$. The fact that some varying number of g deletions take place during a single update of a list $L(\nu)$ is not a problem, due to the observation that each deletion must follow an earlier insertion of the same interval. Hence, at any time during the preprocessing, the number of insertions so far exceeds the number of deletions. So, the amortized number of deletions per list update is one as well, which implies that the amortized time spent in updating a single list with a new hypercube is $O(\log\log n)$. Within each level-$(d-1)$ substructure T, the required associated list updates are restricted to a subset of the $O(\log n)$ nodes of the subtree T', so the time spent on all necessary list updates in T is $O(\log n \log\log n)$. Combined with the $O(\log n \log\log n)$ time bound for the simultaneous search in all lists $L(\nu')$ with ν' in T' for the locations of the value i_1^L,

this implies that the total time spent on the update of a single level-$(d-1)$ substructure is $O(\log n \log\log n)$. Since the total number of level-$(d-1)$ substructures that have to be updated is bounded by $O(\log^{d-2} n)$, the insertion of C_{n-m} into T_m to obtain T_{m+1} takes time $O(\log^{d-1} n \log\log n)$.

Lemma 5.2 *The insertion of the hypercube C_{n-m} into T_m to obtain T_{m+1} requires $O(\log^{d-1} n \log\log n)$ amortized time.*

The computation of the structure T_{m+1} by inserting the hypercube C_{n-m} into the intermediate structure T_m is independent of the actual shape of the objects under consideration. The efficiency of this part is therefore guaranteed, irrespective of the object shape. The efficiency of the computation of V_{n-m}, however, relies on the fact that the objects under consideration are convex or polytopes. The dependence follows from the use of the results from Section 4.

Computation of V_{n-m}

The computation of $V_{n-m} = \{E_j \in \mathcal{E} | E_j \cap C_{n-m} \neq \emptyset \wedge j \geq n-m\}$ is based on a sequence of point location queries. In the static point location structure, the query time was found to be $O(\log^{d-1} n)$ due to the incorporation of fractional cascading. Throughout the incremental construction of the point location structure, however, we use dynamic fractional cascading instead of fractional cascading to achieve efficient insertions, which leads to a query time of $O(\log^{d-1} n \log\log n)$. A point stabbing query with a point p in the intermediate structure T_m proceeds in nearly the same recursive manner as in the static case outlined in Section 3. The exception is that the lists $L(\nu)$ of all nodes ν on the search path of p in a level-$(d-1)$ substructure T, which form a connected subtree T' of T of size $O(\log n)$, can be searched simultaneously in worst-case time $O(\log n \log\log n)$, by Theorem 5.1(a). The entire point stabbing query time amounts to $O(\log^{d-1} n \log\log n)$. Note that a single search with p yields $O(\log^{d-1} n)$ candidate answers: one for each list $L(\nu)$ that is searched. The ultimate answer to the query is clearly the minimum among all hypercube indices found.

The computation of the set V_{n-m} benefits from the fact that the hypercubes are inserted into the data structure from large to small in the sense that at the time of the set's computation, the intermediate data structure only stores hypercubes and objects from the appropriate index range $[n-m+1, \ldots, n]$. Therefore, we may restrict ourselves to finding the hypercubes in the data structure that intersect the query hypercube C_{n-m} without having to bother about the sizes of these hypercubes. Moreover, note that future additions of hypercubes and their corresponding objects do not affect the earlier computed sets V_j. To apply the range searching results from Section 4, we must verify the validity of the constant ratio between the diameter of the search region (the hypercube C_{n-m}) and the lower bound on the radii of the minimal enclosing hyperspheres of the stored objects. The radii of the minimal enclosing hyperspheres of the objects in $\{E_{n-m+1}, \ldots, E_n\}$ are bounded from below by ρ_{n-m+1}, and, by the ordering on the radii, also by ρ_{n-m}. The query region C_{n-m} is the axis-parallel enclosing hypercube of the minimal enclosing hypersphere of E_{n-m} with radius ρ_{n-m}. As a result, the diameter of C_{n-m} is $2\sqrt{d}\cdot\rho_{n-m}$. The application of Theorems 4.7 and 4.10, yields, taking into account the modified point location query time due to dynamic fractional cascading, a worst-case time bound of $O(\log^{d-1} n \log\log n)$ for the computation of V_{n-m}.

Lemma 5.3 *Let, for all $n-m < j \leq n$, V_j be a set of convex objects or polytopes. Then the computation of the set V_{n-m} from T_m and $\{V_j | n-m < j \leq n\}$ takes $O(\log^{d-1} n \log\log n)$ time.*

Lemmas 5.2 and 5.3 show that each of the $O(n)$ steps in the incremental construction of the d-level data structure for point location and range searching among convex objects or polytopes takes $O(\log^{d-1} n \log\log n)$ time, resulting in a time bound of $O(n \log^{d-1} n \log\log n)$ for the computation of T_n from the skeleton T_0. Adding to this bound the $O(n \log^{d-1} n)$ time bound for building the $(d-1)$-level segment tree T_0, we obtain the desired result.

Theorem 5.4 *Let $\mathcal{E}$ be a set of non-intersecting constant-complexity k-fat convex objects or arbitrary polytopes. Then the d-level point stabbing structure can be built in time $O(n \log^{d-1} n \log\log n)$.*

After the construction of the data structure, the query time can be improved back to $O(\log^{d-1} n)$. To this end, it suffices to rebuild the structure using static fractional cascading. As all the sets V_i are now known, this can easily be achieved in time $O(n \log^{d-1} n)$.

6 Summary of results and extensions

We have presented a data structure for both point location and range searching with arbitrarily-shaped bounded-size query ranges in certain scenes of fat objects. Theorem 6.1 summarizes the results by combining Theorems 3.3, 4.7, 4.10, and 5.4.

Theorem 6.1 *Let $k > 0$ and $h \geq 0$ be constants and let $\mathcal{E}$ be a set of non-intersecting constant-complexity k-fat arbitrary convex objects and/or non-convex polytopes $E \subseteq \mathbb{R}^d$ with minimal enclosing hypersphere radii at least ρ. Then the set $\mathcal{E}$ can be stored in time $O(n \log^{d-1} n \log\log n)$ in a data structure of size $O(n \log^{d-1} n)$ which supports point location queries and range searching queries with ranges $R \subseteq \mathbb{R}^d$ of diameter at most $h \cdot \rho$ among the objects of $\mathcal{E}$ in time $O(\log^{d-1} n)$.*

The theorem does not apply to scenes of arbitrarily-shaped non-convex (constant-complexity) objects. Our firm belief, though, is that a similar result holds in that case as well. Preliminary results in that direction with two-dimensional objects bounded by algebraic polygonal curves of bounded degree are promising.

Throughout the paper, the assumption that the objects in $\mathcal{E}$ are non-intersecting only plays a role in showing that the number of larger objects E' intersecting the enclosing hypercube C of some object E is bounded by a constant. No other lemma or theorem relies on the disjointness of the objects. As a consequence, all results remain valid if we drop the requirement of disjointness and instead impose the weaker restriction upon $\mathcal{E}$ that each enclosing hypercube C of $E \in \mathcal{E}$ is intersected by at most a constant number of objects E' larger than E. In the generalized setting of intersecting objects, a query point may be contained in more than one object. The answer to a point location, or point stabbing, query should therefore be the collection of objects containing the query point. The new (relaxed) restriction on the data set $\mathcal{E}$ still leads to only a constant number of simultaneous containments of a single query point. An interesting example of such a set is obtained by enlarging the elements of a collection $\mathcal{E}$ of non-intersecting k-fat constant-complexity objects by an amount that is proportional to the size of the smallest object

in $\mathcal{E}$. If the resulting expanded objects are convex or polytopes and, in addition, k-fat themselves (for some constant k), then Theorem 6.1 applies to the set of intersecting expanded objects [34]. The example has applications to molecular modelling [18].

References

[1] P.K. Agarwal, M.J. Katz, and M. Sharir, Computing depth orders and related problems, *Proc. 4th Scandinavian Workshop on Algorithm Theory (SWAT'94)* (E.M. Schmidt and S. Skyum Eds.) Lecture Notes in Computer Science **824** (1994), pp. 1-12.

[2] P.K. Agarwal and J. Matoušek, On range searching with semialgebraic sets, *Discrete & Computational Geometry* **11** (1994), pp. 393-418.

[3] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, and C. Uhrig, Approximate motion planning and the complexity of the boundary of the union of simple geometric figures, *Algorithmica* **8** (1992), pp. 391-406.

[4] M. de Berg, M. de Groot, and M. Overmars, New results on binary space partitions in the plane, *Proc. 4th Scandinavian Workshop on Algorithm Theory (SWAT'94)* (E.M. Schmidt and S. Skyum Eds.) Lecture Notes in Computer Science **824** (1994), pp. 61-72.

[5] B. Chazelle, How to search in history, *Information and Control* **64** (1985), pp. 77-99.

[6] B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir, A singly-exponential stratification scheme for real semi-algebraic varieties and its applications, *Theoretical Computer Science* **84** (1991), pp. 77-105.

[7] B. Chazelle and J. Friedman, Point location among hyperplanes and unidirectional ray shooting, *Computational Geometry: Theory and Applications* **4** (1994), pp. 53-62.

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

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

[10] B. Chazelle and E. Welzl, Quasi-optimal range searching in spaces of finite VC-dimension, *Discrete & Computational Geometry* **4** (1989), pp. 467-489.

[11] K.L. Clarkson, New applications of random sampling in computational geometry, *Discrete & Computational Geometry* **2** (1987), pp. 195-222.

[12] R. Cole, Searching and storing similar lists, *Journal of Algorithms* **7** (1986), pp. 202-220.

[13] H. Edelsbrunner, *Algorithms in combinatorial geometry*, Springer-Verlag, Berlin (1987).

[14] H. Edelsbrunner, L.J. Guibas, and J. Stolfi, Optimal point location in a monotone subdivision, *SIAM Journal on Computing* **15** (1986), pp. 317-340.

[15] A. Efrat, G. Rote, and M. Sharir, On the union of fat wedges and separating a collection of segments by a line, *Computational Geometry: Theory and Applications* **3** (1993), pp. 277-288.

[16] G.M. Fikhtengol'ts, *The Fundamentals of Mathematical Analysis, Vol. II*, English edition, Pergamon Press (1965).

[17] M.T. Goodrich and R. Tamassia, Dynamic trees and dynamic point location, *Proc. 23rd ACM Symp. on Theory of Computing* (1991), pp. 523-533.

[18] D. Halperin and M.H. Overmars, Spheres, molecules, and hidden surface removal, *Proc. 10th Ann. ACM Symp. on Computational Geometry* (1994), pp. 113-122.

[19] M.J. Katz, M.H. Overmars, and M. Sharir, Efficient hidden surface removal for objects with small union size, *Computational Geometry: Theory and Applications* **2** (1992), pp. 223-234.

[20] D.G. Kirkpatrick, Optimal search in planar subdivisions, *SIAM Journal on Computing* **12** (1983), pp. 28-35.

[21] M. van Kreveld, *New results on data structures in computational geometry*, Ph.D. Thesis, Dept. of Computer Science, Utrecht University (1992).

[22] M. van Kreveld, On fat partitioning, fat covering and the union size of polygons, Technical Report RUU-CS-93-36, Dept. of Computer Science, Utrecht University (1993).

[23] K. Leichtweiss, Über die affine Exzentrizität konvexer Körper, *Archiv der Mathematik* **10** (1959), pp. 187-199 (in German).

[24] J. Matoušek, Efficient partition trees, *Discrete & Computational Geometry* **8** (1992), pp. 315-334.

[25] J. Matoušek, Range searching with efficient hierarchical cuttings, *Discrete & Computational Geometry* **10** (1993), pp. 157-182.

[26] J. Matoušek, J. Pach, M. Sharir, S. Sifrony, and E. Welzl, Fat triangles determine linearly many holes, *SIAM Journal on Computing* **23** (1994), pp. 154-169.

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

[28] M.H. Overmars, Point location in fat subdivisions, *Information Processing Letters* **44** (1992), pp. 261-265.

[29] M.H. Overmars and A.F. van der Stappen, Range searching and point location among fat objects, *Proc. 2nd Annual European Symp. on Algorithms (ESA'94)* (J. van Leeuwen Ed.), Lecture Notes in Computer Science (1994).

[30] F.P. Preparata and M.I. Shamos, *Computational geometry: an introduction*, Springer Verlag, New York (1985).

[31] F.P. Preparata and R. Tamassia, Efficient spatial point location, *Proc. 1st Workshop on Algorithms and Data Structures*, Lecture Notes in Computer Science **382** (1989), pp. 3-11.

[32] A.F. van der Stappen, D. Halperin, M.H. Overmars, The complexity of the free space for a robot moving amidst fat obstacles, *Computational Geometry: Theory and Applications* **3** (1993), pp. 353-373.

[33] A.F. van der Stappen and M.H. Overmars, Motion planning amidst fat obstacles, *Proc. 10th ACM Symp. on Computational Geometry* (1994), pp. 31-40.

[34] A.F. van der Stappen, *Motion planning amidst fat obstacles*, Ph.D. Thesis, Dept. of Computer Science, Utrecht University (1994).

[35] W. Walter, *Analysis II (Grundlagen Mathematik Bd. 4)*, Springer-Verlag, Berlin (1990) (in German).