ELSEVIER

Computational Geometry 15 (2000) 215–227

Computational Geometry
Theory and Applications

www. elsevier.nl/locate/comgeo

Dynamic data structures for fat objects and their applications ☆

Alon Efrat [a], Matthew J. Katz [b,*,1], Frank Nielsen [c], Micha Sharir [a,d,2]

[a] *School of Mathematical Sciences, Tel Aviv University, Tel-Aviv 69978, Israel*
[b] *Department of Mathematics & Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel*
[c] *SONY Computer Science Laboratories Inc., Tokyo, Japan*
[d] *Courant Institute of Mathematical Sciences, New York University, New York, USA*

Communicated by M. Overmars; received 2 November 1998; received in revised form 5 October 1999; accepted 1 December 1999

Abstract

We present several efficient dynamic data structures for point-enclosure queries, involving convex fat objects in $\mathbb{R}^2$ or $\mathbb{R}^3$. Our planar structures are actually fitted for a more general class of objects – (β, δ)-covered objects – which are not necessarily convex, see definition below. These structures are more efficient than alternative known structures, because they exploit the fatness of the objects. We then apply these structures to obtain efficient solutions to two problems: (i) finding a perfect containment matching between a set of points and a set of convex fat objects, and (ii) finding a piercing set for a collection of convex fat objects, whose size is optimal up to some constant factor.

Keywords: Fat objects; Dynamic data structure; Point enclosure; Containment matching; Piercing set

1. Introduction

A convex object c in $\mathbb{R}^d$ is α-fat, for some parameter $\alpha > 1$, if the ratio between the radii of the balls s^+ and s^- is at most α, where s^+ is the smallest ball containing c and s^- is a largest ball that is contained in c. Often the input set in practical instances of geometric problems consists of fat objects. Fat objects have several desirable properties, which were used by many authors to obtain more efficient solutions to

☆ A preliminary version of this paper appeared as [14].

* Corresponding author.

E-mail addresses: matya@cs.bgu.ac.il (M.J. Katz), alone@math.tau.ac.il (A. Efrat), nielsen@csl.sony.co.jp (F. Nielsen), sharir@math.tau.ac.il (M. Sharir).

[1] Supported by the Israel Science Foundation founded by the Israel Academy of Sciences and Humanities.

[2] Supported by NSF Grant CCR-97-32101, by a grant from the U.S.–Israeli Binational Science Foundation, by the ESPRIT IV LTR project No. 21957 (CGAL), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

0925-7721/00/$ – see front matter
PII: S0925-7721(99)00059-0

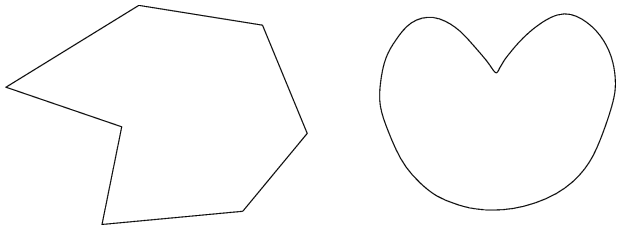

Fig. 1. Two (β, δ)-covered objects.

a variety of algorithmic problems, when the underlying objects are fat. See [4,6,12,17,18,21,23,25,26] for a sample of these results.

In a recent paper [17], Katz has designed a data structure of nearly linear size for sets of convex α-fat objects in the plane. By augmenting the data structure in various ways he obtains efficient and simple solutions to several query-type problems, including the *point enclosure* problem, where we wish to determine whether a query point q lies in the union of the input set, and, if so, to report a witness object containing q, or, alternatively, report all k objects containing q. The cost of such a query is $O(\text{polylog}\, n)$ (or $O(\text{polylog}\, n + k \cdot \text{polylog}\, n)$), as opposed to roughly $O(\sqrt{n})$ (or $O(\sqrt{n} + k)$), which is the cost of a query when the fatness assumption is dropped and only nearly linear storage is allowed [3,19,20].

In this paper we continue the work of [17]. We first extend the above definition of fatness (in the plane) to non-convex objects. A triangle Δ is a (β, δ)-*triangle* of a planar object c ($0 < \beta \leqslant \pi/3$, $0 < \delta < 1$), if $\Delta \subseteq c$, each of the angles of Δ is at least β, and the length of each of its edges is at least $\delta \cdot \text{diam}(c)$. An object c in the plane is (β, δ)-*covered* if for each point $p \in c$ there exists a (β, δ)-triangle Δ of c that contains p (see Fig. 1).

It is easy to see that for convex objects the two definitions are equivalent, in the sense that if c is α-fat, for some constant α, then it is also (β, δ)-covered, for appropriate constants β and δ, and vice versa. However, for non-convex objects the definition of being fat is more general than the definition of being (β, δ)-covered; the former definition does not capture the class of objects that we wish to consider as fat here, while the latter definition does. It is easy to verify that the same remark holds with respect to the alternative definition of being fat due to van der Stappen et al. [25], which is also more general than the definition of being (β, δ)-covered.

Let $\mathcal{C}$ be a set of n objects that consists of either convex α-fat objects in $\mathbb{R}^3$ (for some small constant $\alpha > 1$), or (β, δ)-covered objects in $\mathbb{R}^2$ (for some not too small constants β and δ). In the paper we present dynamic data structures for $\mathcal{C}$ that enable us to answer a point-enclosure query efficiently, and to insert or delete objects into/from $\mathcal{C}$ efficiently. The specific bounds depend on the dimension (2D or 3D) and on the type of objects that are stored in the data structure (e.g., convex α-fat polygons or polyhedra, or convex α-fat general objects). In general, these bounds are significantly better than the corresponding known bounds where fatness is not assumed. Consider, for example, the case where the objects are (not necessarily axis-parallel) cubes in $\mathbb{R}^3$. The standard storage/query tradeoff in this case lets s (the size of the data structure) vary in the range n to n^3, and the query cost, expressed as a function of both n and s, is close to $n/s^{1/3}$. Since cubes are fat objects, we may use our data structure in this case. The effect of using our structure is equivalent to a reduction by one in the dimension, in the sense that the storage/query tradeoff that we obtain is roughly the same as for triangles in the plane. That is, s varies in the range n to n^2, and the query cost is only about $n/\sqrt{s}$. Moreover, the

structure can be maintained dynamically, when inserting or deleting objects, at a cost of about s/n per update.

In addition, in the planar case, the static version of our data structure for planar (β, δ)-covered objects, improves upon the corresponding structure of [17] in two ways: it applies to a larger class of objects, i.e., to (β, δ)-covered objects rather than to convex α-fat objects, and its corresponding bounds (preprocessing, storage and query) are better by a logarithmic factor.

In Section 3 we present two applications of our data structures as stated in the abstract. In both applications it is crucial that the structures be dynamic.

In the first application, we are given a set of n points and a set of n convex α-fat objects in $\mathbb{R}^2$ or in $\mathbb{R}^3$ (or a set of n (β, δ)-covered objects in $\mathbb{R}^2$), and the goal is to compute a perfect matching between the points and objects, so that each point is matched to an object that contains it. Variants of this problem arise frequently in geometric pattern matching [7,13]. Under appropriate (but rather weak) assumptions on the input objects, we obtain algorithms that solve the matching problem in time close to $O(n^{11/6})$ in $\mathbb{R}^3$, and close to $O(n^{3/2})$ in $\mathbb{R}^2$. Our algorithms are based on the recent efficient matching technique of Efrat and Itai [13].

Our second application concerns piercing fat objects. A set of points $\mathcal{P}$ in $\mathbb{R}^d$ is a *piercing set* for a set $\mathcal{C}$ of objects, if for each object $c \in \mathcal{C}$ there exists a point in $\mathcal{P}$ that lies in c. Finding a minimal piercing set is NP-complete for $d \geqslant 2$ [16], so it is natural to seek approximate solutions, in which the size of the computed piercing set is not much larger than the optimal size. The problem of finding a minimal piercing set is a special instance of the well-known *set cover* problem, if we regard each cell in the arrangement of $\mathcal{C}$ as the subset of objects of $\mathcal{C}$ containing it. Therefore we can apply the greedy algorithm for finding a set cover [10] to obtain, in polynomial time, a piercing set whose size is larger than the optimal size by a factor of $(1 + \log l)$, where $l \leqslant n$ is the *depth* of the arrangement of $\mathcal{C}$ (the maximum number of objects containing a common point). Brönnimann and Goodrich [9] (see also Clarkson [11]) presented a polynomial-time algorithm for computing a set cover in which the approximation factor depends both on the optimal cover size a and on the VC-dimension of the underlying set system. If the VC-dimension is some constant, then their algorithm finds a cover of size $O(a \log a)$. We present algorithms for sets of convex α-fat objects in $\mathbb{R}^2$ or in $\mathbb{R}^3$ (or sets of (β, δ)-covered objects in $\mathbb{R}^2$) that find a piercing set whose size is larger than the optimal size by only a constant factor. The running time is close to $O(n^{4/3})$ in $\mathbb{R}^3$ and close to linear in $\mathbb{R}^2$.

A third application is described in [15]. Let $\mathcal{C}$ be a set of n convex α-fat objects in the plane. In the bounded-length segment shooting problem, we wish to preprocess $\mathcal{C}$, so that, for a given oriented query segment $\vec{r} = \overrightarrow{ab}$, whose length is at most some constant times the smallest diameter of an object in $\mathcal{C}$, the first object of $\mathcal{C}$ hit by $\vec{r}$ (if such an object exists) can be found efficiently. (This is an object c for which there exists $z \in \vec{r}$ such that $z \in c$, and the relative interior of az does not meet any object of $\mathcal{C}$.) An efficient solution to this problem is presented in [17] for the special case where $\mathcal{C}$ consists of either (constant-complexity) polygons or disks. In [15] we present a solution for the general (fat) case. The data structure we describe is based on the (static version of the) data structure for point enclosure (see remark just after Theorem 2.4); its size is nearly linear in n, and the query cost is polylogarithmic, as opposed to roughly $O(\sqrt{n})$ in the non-fat setting (see [1]).

2. Dynamic data structures for fat objects

In this section we present efficient dynamic data structures for point-enclosure queries involving a collection $\mathcal{C}$ of either convex fat objects in $\mathbb{R}^3$, or (not necessarily convex) (β, δ)-covered objects in $\mathbb{R}^2$. Specifically, we want such a structure to support queries in which we are given a point q and wish to determine whether q lies in the union of the objects of $\mathcal{C}$, and, if so, report an object containing q (or, alternatively, report all objects containing q). We also want to maintain this structure under insertions and deletions of objects into/from $\mathcal{C}$. We present several data structures for this problem, depending on the dimension and on the type of objects in $\mathcal{C}$.

2.1. Fat polytopes in three dimensions

Let $\mathcal{C}$ be a set of n convex polytopes in $\mathbb{R}^3$. We assume that each polytope is α-fat, for some fixed constant parameter $\alpha > 1$, and has a constant number of facets. We further assume that each facet of each polytope in $\mathcal{C}$ is triangulated.

We first use a straightforward extension to three dimensions of the planar data structure of Katz [17], to obtain a 3-level tree $\mathcal{T}$, so that given a query point q, we can return, in $O(\log^3 n)$ time, $O(\log^3 n)$ disjoint canonical subsets of $\mathcal{C}$, each stored at some node of $\mathcal{T}$, so that any polytope of $\mathcal{C}$ containing q belongs to one of these subsets, and so that each canonical subset has a nonempty intersection. (The constant of proportionality in these bounds depends on α.) The structure $\mathcal{T}$ can be constructed in $O(n \log^3 n)$ time and requires $O(n \log^2 n)$ storage. The idea behind the structure in [17], is to construct a (three-dimensional) segment tree for the axis-parallel bounding boxes of the objects of $\mathcal{C}$, and to use the fact that only a constant number of points are needed to stab all objects associated with each canonical subset of this tree. Our tree $\mathcal{T}$ can be maintained dynamically, using standard binary decomposition techniques of Bentley and Saxe [8]. If constructing a static version of the data structure takes time $T(n)$, then inserting an object takes time $O((T(n)/n) \log n)$. Since in our case $T(n) = O(n \log^3 n)$, the cost of an insertion is $O(\log^4 n)$. Deletion of an object c is done as follows: we mark c as being deleted from each pre-stored subset containing c in each of the three levels of $\mathcal{T}$. When the actual number of objects in such a subset becomes less than half its original cardinality, we reconstruct all the substructures associated with this subset.

We augment $\mathcal{T}$ as follows. Let $\mathcal{C}^* \subseteq \mathcal{C}$ be a canonical subset, and let p^* be a (pre-computed) point common to all its elements (the construction enables us to assume that p^* does not lie on the boundary of any of the elements of $\mathcal{C}^*$). Let Q^* be an axis-parallel unit cube centered at p^*. We centrally project the boundary of each $c \in \mathcal{C}^*$ from p^* onto ∂Q^*, to obtain a collection of $O(n)$ polygons, each having $O(1)$ edges, on ∂Q^*. We process each facet f of Q^* for efficient point enclosure queries (in the non-fat planar setting). That is, we construct a data structure (see [3]) using $O(s)$ storage, where s varies between n and n^2, so that for a given point $q \in f$, we can report the set of polygons on f that contain q as the disjoint union of $O(n^{1+\varepsilon}/\sqrt{s})$ canonical subsets. The cost of a query is $O(n^{1+\varepsilon}/\sqrt{s})$, and the structure can be maintained dynamically, when inserting or deleting polygons, at a cost of $O(s/n^{1-\varepsilon})$ per update.

We next construct another layer of our data structure, as follows. For each canonical set $\mathcal{P}$ of polygons stored in one of the point-enclosure substructures, we replace each polygon π in $\mathcal{P}$ by the plane containing the polytope facet that has been projected onto π, and store the (boundary of the) intersection of the halfspaces bounded by these planes and not containing p^*. For this we use the data structure of Agarwal and Matoušek [5, Theorem 2.8], which maintains dynamically the upper envelope of these

planes (relative to the normal direction of the facet f). This structure enables us to determine in $O(\log n)$ time whether a query point lies above all these planes. Alternatively, it enables us to report in $O(\log n + k)$ time the k planes of this structure lying above a query point. Moreover, a plane can be inserted or deleted in time $O(n^\varepsilon)$. The construction of the structure is doable in time $O(n^{1+\varepsilon})$, and this is also the storage needed. This completes the description of our data structure.

Answering a query. Let q be a query point. We wish to determine whether some polytope of $\mathcal{C}$ contains q and, if so, produce a witness polytope that contains q (or, alternatively, report all such polytopes). We start by querying the first layer of our structure, and obtain a collection of $O(\log^3 n)$ canonical subsets, each augmented as above. For each subset $\mathcal{C}^*$, with a common point p^*, we compute the intersection q' of the ray emerging from p^* towards q with the boundary of the cube Q^*, and query the corresponding planar point-enclosure substructure with q'. The answer to this query consists of $O(n^{1+\varepsilon}/\sqrt{s})$ disjoint canonical subsets of projected polytope facets, where all members of such a subset contain q'. We finally query each of the corresponding third-layer upper-envelope substructures with q. It is easy to verify that q lies below the upper envelope of at least one such substructure if and only if q lies in the union of the polytopes of $\mathcal{C}^*$. If this is the case, we can either report all polytopes containing q, by reporting all the planes that lie above q in each of the corresponding substructures, or stop after reporting just one such plane. The overall cost of a query is thus easily seen to be $O(n^{1+\varepsilon}/\sqrt{s})$, or $O(n^{1+\varepsilon}/\sqrt{s}+k)$ in the reporting version, where k is the output size.

Updating the structure. Each of the three layers of our structure is dynamic, and the updating of the whole structure is easy to do layer-by-layer. We omit the straightforward details. The overall cost of an update operation is $O(s/n^{1-\varepsilon})$. Thus we obtain the following theorem.

Theorem 2.1. *Let $\mathcal{C}$ be a set of n convex α-fat polytopes in $\mathbb{R}^3$, each with a constant number of facets. For any parameter $n \leqslant s \leqslant n^2$, we can preprocess $\mathcal{C}$ in time $O(s^{1+\varepsilon})$, into a data structure of size $O(s)$, such that finding a polytope of $\mathcal{C}$ containing a query point or reporting all k such polytopes can be done in time $O(n^{1+\varepsilon}/\sqrt{s})$ or $O(n^{1+\varepsilon}/\sqrt{s}+k)$, respectively. Moreover, we can insert or delete an objects into/from $\mathcal{C}$ in time $O(s/n^{1-\varepsilon})$.*

2.2. Balls in three dimensions

The solution in this case is an immediate consequence of standard techniques, but we include its description here for the sake of completeness. (See Theorem 2.5 for a summary of the results.)

Let $B_1, \ldots, B_n$ be the n given balls. For each i, let $p_i = (a_i, b_i, c_i)$ denote the center of B_i and let r_i denote its radius. A query point $q = (x, y, z)$ is inside B_i if and only if

$$(x - a_i)^2 + (y - b_i)^2 + (z - c_i)^2 \leqslant r_i^2$$

or

$$(x, y, z, x^2 + y^2 + z^2) \cdot (-2a_i,\ -2b_i,\ -2c_i, 1) \leqslant r_i^2 - a_i^2 - b_i^2 - c_i^2.$$

Hence, if we transform the query point q to the point $\bar{q} = (x, y, z, x^2 + y^2 + z^2)$ in $\mathbb{R}^4$, then q lies in the union of $\mathcal{C}$ if and only if $\bar{q}$ lies below the upper envelope of the hyperplanes $\xi_i \cdot \boldsymbol{x} = \alpha_i$, for $i = 1, \ldots, n$, where $\xi_i = (-2a_i,\ -2b_i,\ -2c_i, 1)$ and $\alpha_i = r_i^2 - a_i^2 - b_i^2 - c_i^2$. The data structure of Agarwal and

Matoušek [5] cited above allows us to answer such queries in time $O(n^{1+\varepsilon}/\sqrt{s})$, using a data structure that requires $O(s)$ storage and $O(s^{1+\varepsilon})$ preprocessing time, where s can vary between n and n^2. Moreover, we can insert or delete a hyperplane (i.e., a ball in $\mathbb{R}^3$) in time $O(s/n^{1-\varepsilon})$.

2.3. General fat objects in three dimensions

Next consider the case where $\mathcal{C}$ is a collection of general convex α-fat objects in $\mathbb{R}^3$. We assume here that each object in $\mathcal{C}$ has *constant description complexity*, in the usual sense that its boundary is a semialgebraic set defined in terms of a constant number of polynomial equalities and inequalities of constant maximum degree.

In this case we use the first layer of the data structure described in Section 2.1 which also applies for this kind of objects. For each canonical set $\mathcal{C}^*$, with a common point p^*, we represent the boundary of each $c \in \mathcal{C}^*$ as a function $r = f_c(\theta, \phi)$ in spherical coordinates about p^*. With an appropriate standard re-parameterization (which we will not detail here), the graphs of these functions are algebraic of constant description complexity, in the above sense. We need to maintain the upper envelope E^* of these functions. Indeed, a query point q lies in the union of $\mathcal{C}^*$ if and only if $r_q \leqslant E^*(\theta_q, \phi_q)$, where (r_q, θ_q, ϕ_q) are the spherical coordinates of q about p^*.

The maintenance of this envelope can be accomplished using the 'shallow-levels' data structure of Agarwal et al. [2]. This structure has size $O((n^*)^{2+\varepsilon})$ and can be constructed in time $O((n^*)^{2+\varepsilon})$, where $n^* = |\mathcal{C}^*|$. Using this structure, we can determine whether $r_q \leqslant E^*(\theta_q, \phi_q)$ in $O(\log n^*) = O(\log n)$ time, or report all k objects of $\mathcal{C}^*$ that contain q in time $O(\log n + k)$. An insertion or deletion of an object takes $O((n^*)^{1+\varepsilon})$ time. It follows that the overall size of the full data structure is also $O(n^{2+\varepsilon})$, that a query can be performed in time $O(\log^4 n)$ (or $O(\log^4 n + k)$), and that an update takes $O(n^{1+\varepsilon})$ time. These bounds are summarized in Theorem 2.2.

Theorem 2.2. *Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^3$, each with constant description complexity. We can preprocess $\mathcal{C}$ in time $O(n^{2+\varepsilon})$, into a data structure of size $O(n^{2+\varepsilon})$, such that finding an object of $\mathcal{C}$ containing a query point or reporting all k such objects can be done in time $O(\log^4 n)$ or $O(\log^4 n + k)$, respectively. Moreover, we can insert or delete an object into/from $\mathcal{C}$ in time $O(n^{1+\varepsilon})$. The constant of proportionality depends on ε and on the algebraic complexity of each object.*

2.4. General (β, δ)-covered objects in the plane

In this subsection we consider the case where $\mathcal{C}$ is a set of n general (β, δ)-covered objects in the plane. Recall that a planar object c is (β, δ)-covered if for each point $p \in c$, there exists a (β, δ)-triangle of c that contains p. That is, there exists a triangle $\Delta \subseteq c$, such that each of the angles of Δ is at least β, each of the edges of Δ is at least $\delta \cdot \mathrm{diam}(c)$, and $p \in \Delta$. As before, we also assume that each object in $\mathcal{C}$ has constant description complexity.

Recall that if all objects in $\mathcal{C}$ are convex, then the two-dimensional data structure of [17], mentioned above, can be used for the static version of the point enclosure problem. (Since in this case, all objects in $\mathcal{C}$ are α-fat for an appropriate constant α.) Using this data structure, one can find an object $c \in \mathcal{C}$ containing a query point, or determine that no such object exists, in $O(\log^3 n)$ time. Below we describe a dynamic data structure whose static version has two advantages over the data structure in [17]. It is

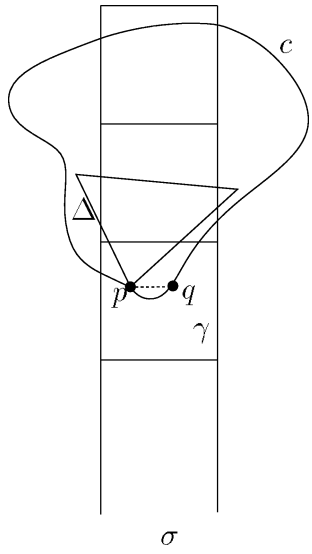

Fig. 2. The proof of Lemma 2.3.

suitable for a more general class of objects, namely, (β, δ)-covered objects, and the cost of a query is only $O(\log^2 n)$.

Let σ be a vertical infinite strip of unit width. Let $c \in \mathcal{C}$ be an object whose x-projection contains the x-projection of σ, and its diameter is large enough, so that any (β, δ)-triangle of c contains a horizontal segment of unit length. Let Γ_σ be a partition of σ into pairwise openly-disjoint square cells of unit edge length, let γ be a cell of Γ_σ, and let $c_\gamma = c \cap \gamma$.

Let l_{y_0} denote the horizontal line $y = y_0$. We say that a cell γ is *abnormal* (with respect to c), if there exists y_0 such that l_{y_0} intersects c_γ, and the two endpoints of one (or more) of the connected portions (intervals) of $c_\gamma \cap l_{y_0}$ lie strictly in the interior of γ. All cells which are not abnormal are *normal*.

Lemma 2.3. *At most* $O(1)$ *cells of* Γ_σ *are abnormal with respect to* c.

Proof. Since c has constant description complexity, the boundary of c intersects the boundary of σ at a set $\mathcal{X}$ of $O(1)$ points. We charge each abnormal cell γ to a point of $\mathcal{X}$ that lies either in γ or in one of the $2s_2$ neighboring cells of γ, s_2 in each direction, where s_2 is an appropriate constant. Thus each point of $\mathcal{X}$ is charged at most $2s_2 + 1$ times.

Let $\gamma \in \Gamma_\sigma$ be an abnormal cell, and let $z \subseteq l_{y_0}$ be an interval of $c_\gamma \cap l_{y_0}$, for some l_{y_0}, such that both its endpoints p and q lie in the interior of γ. Let Δ be a (β, δ)-triangle of c that contains p; see Fig. 2. Let $\Delta(y_0)$ be the intersection of Δ with the line l_{y_0}. We investigate the function $|\Delta(y)|$, the length of $\Delta(y) = \Delta \cap l_y$. Clearly, $|\Delta(y)|$ is not maximal at y_0, since $|\Delta(y_0)| < 1$. However, we can choose a constant s_2, such that either in γ, or within the s_2 upwards/downwards neighboring cells of γ, $|\Delta(y)|$ becomes greater than 1. Thus, ∂c must intersect the boundary of σ either in γ, or in one of the s_2 neighboring cells of γ, and we charge γ to such an intersection point. □

We next describe the data structure. Let $\mathcal{T}$ be a segment tree constructed on the projections on the x-axis of the objects of $\mathcal{C}$. Each node v of $\mathcal{T}$ corresponds to a vertical infinite slab σ_v, and to a subset

$\mathcal{C}_v \subseteq \mathcal{C}$ of objects. We divide each slab σ_v into a constant number of narrower non-overlapping *strips*, $\sigma_{v_1}, \ldots, \sigma_{v_l}$, of equal width, such that any (β, δ)-triangle of an object $c \in C_v$ can contain a horizontal line segment whose length is equal to the width of a strip. (This is possible since $\operatorname{diam}(c)$ is greater than the width of σ, and since β is a not too small constant.) By abusing notation, we replace each node $v \in \mathcal{T}$ by a set of nodes $v_1, \ldots, v_\ell$, such that v_i corresponds to the strip σ_{v_i} and $\mathcal{C}_{v_i} = \mathcal{C}_v$ (for $i = 1, \ldots, l$). Combining Lemma 2.3 with the properties of segment trees, one can easily show that

$$\sum_{v \in \mathcal{T}} \sum_{c \in \mathcal{C}_v} \text{number of cells in } \Gamma_{\sigma_v} \text{ for which } c \text{ is abnormal} = O(n \log n).$$

Let v be a vertex of $\mathcal{T}$, and let γ be a cell of Γ_{σ_v} that is abnormal for at least one of the objects in $\mathcal{C}_v$. Put $\mathcal{C}_v^\gamma \equiv \{c \mid c \in \mathcal{C}_v,\ \gamma \text{ is abnormal with respect to } c\}$. We can find a set of points P_γ of constant size, such that if Δ is any (β, δ)-triangle of an object $c \in \mathcal{C}_v^\gamma$ that intersects γ, then at least one of the points in P_γ lies in Δ. The existence of P_γ follows from the fact that Δ is fat and its area is a constant factor times the area of γ (and $\Delta \cap \gamma \neq \emptyset$). Therefore, each point $p \in c_\gamma$ can be connected to a point $\xi \in P_\gamma$ by a segment that is fully contained in c. (Simply, connect p to a point $\xi \in P_\gamma$ that lies in one of the (β, δ)-triangles of c containing p.) We say that p is *visible* from ξ, and the union of all points of c which are visible from ξ is called the *visible region of* ξ *in* c, and denoted c_ξ. We thus compute for each object $c \in \mathcal{C}_v^\gamma$ and for each point $\xi \in P_\gamma$ the region c_ξ. Put $\mathcal{C}_\xi = \{c_\xi \mid c \in \mathcal{C}_v^\gamma\}$, for $\xi \in P_\gamma$.

Clearly, the boundary of c_ξ can be expressed as a function $f_{c_\xi}(\theta)$, $0 \leqslant \theta < 2\pi$; $f_{c_\xi}(\theta)$ is the distance from ξ to the point on the boundary of c_ξ where the ray of orientation θ emanating from ξ crosses the boundary of c_ξ. Consider the upper envelope of the set of functions $\mathcal{F}_\xi = \{f_{c_\xi}(\theta) \mid c \in \mathcal{C}_v^\gamma\}$, for $\xi \in P_\gamma$. Analogously to Section 2.3, a query point q whose polar coordinates with respect to ξ are (θ, ρ), is contained in some visible region c_ξ if and only if the value attained by the upper envelope of $\mathcal{F}_\xi$ at θ is at least ρ. The complexity of the upper envelope of $\mathcal{F}_\xi$ is $O(\lambda_{s_1}(|\mathcal{C}_v^\gamma|))$, where s_1 is an appropriate constant, and $\lambda_s(n)$ is the maximum length of (n, s) Davenport–Schinzel sequences [24].

We construct $\Psi_{v,\gamma,\xi}^{(3)}$, which is the shallow-level data structure of Agarwal et al. [2] to maintain the functions of $\mathcal{F}_\xi$. Let $\Psi_{v,\gamma}^{(2)}$ be the list of roots of the structures $\Psi_{v,\gamma,\xi}^{(3)}$, $\xi \in P_\gamma$. We construct a balanced search tree $\Psi_v^{(1)}$ over the cells in Γ_{σ_v} that are abnormal for at least one of the objects in $\mathcal{C}_v$, and attach to each such cell γ its corresponding list $\Psi_{v,\gamma}^{(2)}$. We associate $\Psi_v^{(1)}$ with the node v.

For each $c \in \mathcal{C}_v$ denote

$$c^{(n)} = (c \cap \sigma_v) \Big\backslash \bigcup \{\gamma \in \Gamma_{\sigma_v} \mid \gamma \text{ is abnormal for } c\}.$$

We next construct a data structure for the regions $c^{(n)}$, $c \in \mathcal{C}_v$. Note that $c^{(n)}$ (and its boundary) may intersect any number of cells of Γ_{σ_v}. For each $c^{(n)}$ we define two (not necessarily disjoint) regions, $c^{(nl)}$ and $c^{(nr)}$, that are contained in $c^{(n)}$ and whose union is $c^{(n)}$ (see Fig. 3). $c^{(nl)}$ is the region of $c^{(n)}$ in which $c^{(n)}$ is a (partially defined) function $g_c^{(l)}(y)$ defined on the left boundary of σ_v, and $c^{(nr)}$ is the region of $c^{(n)}$ in which $c^{(n)}$ is a (partially defined) function $g_c^{(r)}(y)$ defined on the right boundary of σ_v. Put $\mathcal{C}^{(nl)} = \{c^{(nl)} \mid c \in \mathcal{C}_v\}$ and $\mathcal{C}^{(nr)} = \{c^{(nr)} \mid c \in \mathcal{C}_v\}$. Let $\mathcal{G}_v^{(l)}$ (respectively $\mathcal{G}_v^{(r)}$) denote the right (respectively left) envelope of the functions $g_c^{(l)}(y)$ (respectively $g_c^{(r)}(y)$), $c \in \mathcal{C}_v$. Clearly, a query point $q = (x, y)$ which is contained in a cell $\gamma \in \Gamma_{\sigma_v}$ is also contained in some object $c \in \mathcal{C}_v$ for which the cell γ is normal if and only if it is to the left of $\mathcal{G}_v^{(l)}$, or to the right of $\mathcal{G}_v^{(r)}$.

We maintain these envelopes again using the shallow level data structure cited above, denoted in this context $\Xi_v^{(l)}$ and $\Xi_v^{(r)}$, respectively. These structures are associated with the node v.

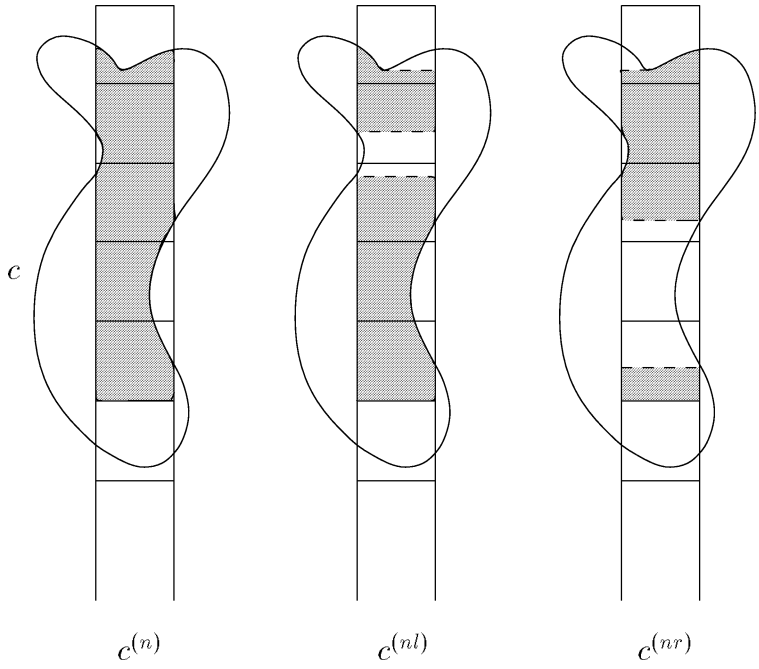

Fig. 3. The regions $c^{(n)}$, $c^{(nl)}$ and $c^{(nr)}$ (in grey).

Answering a query. Given a query point q, we first query $\mathcal{T}$ to find the $O(\log n)$ slabs σ_v containing q. For each such slab, we first need to check whether q lies in $c^{(n)}$, for some $c \in \mathcal{C}_v$. This can be done in $O(\log n)$ time, using the structures Ξ_v^l and Ξ_v^r.

Next we search in $\Psi_v^{(1)}$ (using q's y-coordinate) for the cell γ containing q and its appropriate list of structures $\Psi_{v,\gamma}^{(2)}$. We perform a query in each of the (constant number of) corresponding structures $\Psi_{v,\gamma,\xi}^{(3)}$ in order to determine whether one of the objects $c \in \mathcal{C}_v$ for which γ is abnormal contains q. As easily seen, the overall required time is $O(\log^2 n)$.

Updating the structure. Each of the ingredients of our structure is dynamic, and the updating of the whole structure is easy to do layer-by-layer. We omit the straightforward details. The overall cost of an update operation is $O(n^\varepsilon)$. We have thus shown the following theorem.

Theorem 2.4. *Let $\mathcal{C}$ be a set of n (β, δ)-covered objects in the plane, of constant description complexity, for fixed positive parameters β, δ. We can preprocess $\mathcal{C}$ into a data structure of size $O(n^{1+\varepsilon})$, using $O(n^{1+\varepsilon})$ preprocessing time, such that one can determine in time $O(\log^2 n)$ whether a query point q is contained in some object of $\mathcal{C}$, or report all objects of $\mathcal{C}$ containing q in time $O(\log^2 n + k)$, where k is the size of the output. Moreover, we can insert or delete an objects into/from $\mathcal{C}$ in time $O(n^\varepsilon)$.*

Remark. Note that if we are only interested in a static version, we use a standard sorted array for the vertices of the upper envelope of the functions $\mathcal{F}_\xi$ and for the envelopes $\mathcal{G}_v^{(l)}$ and $\mathcal{G}_v^{(r)}$. As easily checked (see [17]), performing a query is still doable in time $O(\log^2 n)$, the preprocessing time is $O(\lambda_{s_1}(n) \log^2 n)$, and the storage needed is $O(\lambda_{s_1}(n) \log n)$. For the reporting version, the

preprocessing time is $O(\lambda_{s_1}(n)\log^3 n)$, the storage required is $O(\lambda_{s_1}(n)\log^2 n)$, and the cost of a query is $O(\log^2 n + k\log^2 n)$. In both modes (decision and reporting), if the objects are polygons, then $\lambda_{s_1}(n)$ in the bounds above is replaced by n. All these bounds are better by a logarithmic factor in comparison to the corresponding bounds in [17].

2.5. (β, δ)-covered polygons

We can do somewhat better if the objects in $\mathcal{C}$ are (β, δ)-covered polygons in $\mathbb{R}^2$, each with $O(1)$ edges. In this case, we use the data structure of the previous section, applying an idea from Section 2.1. Observe that the visible regions c_ξ that are computed are star-shaped polygons (with 'center' ξ). For each canonical set $\mathcal{C}_\xi$, we map each edge of each of the polygons in $\mathcal{C}_\xi$ to an angular interval about ξ, and store these intervals in a segment tree. Each node v of the tree is associated with some subset Σ_v of polygon edges. We replace each such edge by the line containing it, and maintain the intersection of the halfplanes bounded by these lines and not containing ξ, using the dynamic data structure of Overmars and van Leeuwen [22]. Hence, for a query point q, we compute θ_q, the orientation of q about ξ, compute the $O(\log n)$ canonical subsets in the segment tree, each of whose ranges contains θ_q, in $O(\log n)$ time, and determine for each of them whether q lies in the corresponding intersection of halfplanes. The cost of querying a single canonical set $\mathcal{C}_\xi$ is thus $O(\log^2 n)$ (or $O(\log^2 n + k)$ in the reporting mode).

We handle the sets $\mathcal{C}^{(nl)}$ (alternatively, $\mathcal{C}^{(nr)}$) in a similar way. We project each edge of the polygonal regions in $\mathcal{C}^{(nl)}$ on the left boundary of σ_v, and store these projections in a segment tree and proceed as above. The cost of querying the canonical set $\mathcal{C}^{(nl)}$ (alternatively, $\mathcal{C}^{(nr)}$) is thus $O(\log^2 n)$ (or $O(\log^2 n + k)$ in the reporting mode).

Since we repeat this $O(\log n)$ times, the total cost of a query is $O(\log^3 n)$ (or $O(\log^3 n + k)$ in the reporting mode). Insertion or deletion of a polygon can be done in time $O(\log^4 n)$, using the binary decomposition technique of Bentley and Saxe [8] mentioned above. If only deletions are required, then a deletion can be done in time $O(\log^3 n)$. We thus obtain Theorem 2.5.

Theorem 2.5. *Let $\mathcal{C}$ be a collection of n (β, δ)-covered polygons in $\mathbb{R}^2$, each with $O(1)$ edges. We can preprocess $\mathcal{C}$ in time $O(n\log^3 n)$ into a data structure of size $O(n\log^3 n)$, such that finding a polygon of $\mathcal{C}$ containing a query point or reporting all k such objects can be done in time $O(\log^3 n)$ or $O(\log^3 n + k)$, respectively. Moreover, we can insert or delete a polygon into/from $\mathcal{C}$ in time $O(\log^4 n)$.*

Table 1

Objects:	General objects	Polytopes	Balls	General objects	Polygons
Dimension	3D	3D	3D	2D	2D
Preprocessing	$O(n^{2+\varepsilon})$	$O(s^{1+\varepsilon})$	$O(s^{1+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(n\log^3 n)$
Storage	$O(n^{2+\varepsilon})$	$O(s)$	$O(s)$	$O(n^{1+\varepsilon})$	$O(n\log^3 n)$
Query	$O(\log^4 n)$	$O(n^{1+\varepsilon}/\sqrt{s})$	$O(n^{1+\varepsilon}/\sqrt{s})$	$O(\log^2 n)$	$O(\log^3 n)$
Update	$O(n^{1+\varepsilon})$	$O(s/n^{1-\varepsilon})$	$O(s/n^{1-\varepsilon})$	$O(n^{\varepsilon})$	$O(\log^4 n)$

The results obtained in this section are summarized in Table 1. The parameter s represents any fixed integer between n and n^2.

3. Applications of the data structures

3.1. Matching points and fat objects

Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^2$ or $\mathbb{R}^3$, or a set of n (β,δ)-covered objects in $\mathbb{R}^2$, and let P be a set of n points. We want to solve the *containment matching problem*, which is to determine whether there exists a perfect matching in the bipartite graph whose edges are of the form (p,c), where $p\in P$, $c\in\mathcal{C}$ and $p\in c$. That is, we want to match each point of P to a distinct object that contains it.

Questions of this kind arise frequently in geometric *pattern matching*, where we seek a bijection between two sets of points of equal size, say, $A=\{a_1,\ldots,a_n\}$ and $B=\{b_1,\ldots,b_n\}$, and the distance between any pair of matched points has to be at most some parameter r. In this case the objects of $\mathcal{C}$ are r-neighborhoods of the points b_i (typically, balls of radius r centered at the points b_i). See [13] for a general discussion of this problem, as well as for the relevant literature.

This problem is also similar to the problem investigated by Arkin et al. [7], where a set of points and a set of pairwise-disjoint objects are given, and we seek a transformation that places each point into one of the objects. This problem, however, is different from our problem, in which the objects do not have to be disjoint, and no transformation of the points is allowed.

We can solve the matching problem by applying the bottleneck matching algorithm of Efrat and Itai [13]. This algorithm maintains a dynamic data structure that stores a subset of the objects of $\mathcal{C}$, and supports queries where we specify a point p and wish to find an object in the current subset that contains p, and then delete that object from the structure. The algorithm performs $O(n^{3/2})$ such operations, and its running time is dominated by the cost of these operations.

We use the appropriate data structure from among those developed in the preceding section, depending on the type of objects in $\mathcal{C}$. In the three-dimensional cases, we set the storage parameter s to be $n^{4/3}$, so that both queries and updates take $O(n^{1/3+\varepsilon})$ time each. We thus obtain the following theorem.

Theorem 3.1. *Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^d$ (for $d=2,3$) or a set of n (β,δ)-covered objects in $\mathbb{R}^2$, each of a constant description complexity, and let P be a set of n points in $\mathbb{R}^d$. Then we can either find a one-to-one matching between P and $\mathcal{C}$, such that each point $p\in P$ is contained in the object of $\mathcal{C}$ matched to p, or determine that no such matching exists. The running time of the algorithm is $O(n^{11/6+\varepsilon})$ for polytopes in $\mathbb{R}^3$ and for balls in $\mathbb{R}^3$. The running time is close to $O(n^{3/2+\varepsilon})$ for general objects and $O(n^{3/2}\operatorname{polylog} n)$ for polygons in $\mathbb{R}^2$.*

3.2. Piercing fat objects

Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^2$ or $\mathbb{R}^3$, or a set of n (β,δ)-covered objects in $\mathbb{R}^2$. In this subsection we present algorithms for computing a piercing set for $\mathcal{C}$. Recall that a set of points $\mathcal{P}$ is a *piercing set* for $\mathcal{C}$, if for each object $c\in\mathcal{C}$ there exists a point in $\mathcal{P}$ that lies in c. The algorithms produce piercing sets whose size is optimal up to a constant factor.

The high-level description of the algorithm is simple: For each object $c\in\mathcal{C}$, let Q_c denote the smallest axis-parallel cube enclosing c. We sort the objects of $\mathcal{C}$ in increasing order of the size of Q_c. The

Table 2

Objects:	Polytopes	Balls	General	Polygons
Dimension	3D	3D	2D	2D
Running time	$O(n^{4/3+\varepsilon})$	$O(n^{4/3+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(n \log^4 n)$

algorithm works in stages, where the ith stage starts with the subset $\mathcal{C}_i$ of $\mathcal{C}$ consisting of those objects that have not yet been pierced (initially, $\mathcal{C}_1 = \mathcal{C}$). Let c_i be the smallest object (in the above order) in $\mathcal{C}_i$. Let bQ_{c_i} be the cube Q_{c_i} scaled by some fixed factor $b > 1$ about its center (we can choose, e.g., $b = 2$). The fatness/covering property of the objects of $\mathcal{C}$ and the fact that c_i is the smallest object in $\mathcal{C}_i$ imply that for any object $c \in \mathcal{C}_i$ that intersects c_i, the measure of $c \cap bQ_{c_i}$ is at least some fixed fraction of the measure of bQ_{c_i}. Hence, we can place a constant number of points inside bQ_{c_i} (this number only depends on α and d), so that any $c \in \mathcal{C}_i$ that intersects c_i will contain one of these points. We add these points to the output piercing set, and delete from $\mathcal{C}_i$ all the objects that are pierced by any of them. The subset $\mathcal{C}_{i+1}$ of the remaining objects is then passed to the next stage. The algorithm terminates when this set becomes empty.

The termination of the algorithm, and the fact that its output is a piercing set are both obvious. Moreover, the objects $c_1, c_2, \ldots$ are pairwise disjoint, so if the algorithm terminates after j stages, then the size of the optimal piercing set is at least j, whereas the size of the output is $O(j)$, so the output size is indeed optimal up to a constant factor. To implement the algorithm, we use the appropriate data structure developed in the preceding section, to obtain the following result.

Theorem 3.2. *Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^d$, for some fixed constant $\alpha > 1$ and for $d = 2, 3$, or a set of n (β, δ)-covered objects in $\mathbb{R}^2$, where each object in $\mathcal{C}$ has constant description complexity. Then we can compute a piercing set for $\mathcal{C}$ of size $O(j)$, with the constant of proportionality depending on α and d, where j is the size of a minimal-cardinality piercing set for $\mathcal{C}$. The running time of the algorithm depends on d, and on the type of objects in $\mathcal{C}$, see Table 2.*

References

[1] P.K. Agarwal, Ray shooting and other applications of spanning trees with low stabbing number, SIAM J. Comput. 21 (1992) 540–570.
[2] P.K. Agarwal, A. Efrat, M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, in: Proc. 11th ACM Symp. Comput. Geom., 1995, pp. 39–50.
[3] P.K. Agarwal, J. Erickson, Geometric range searching and its relatives, in: B. Chazelle, E. Goodman, R. Pollack (Eds.), Advances in Discrete and Comput. Geom., Amer. Math. Soc., Providence, RI, 1998.
[4] P.K. Agarwal, M.J. Katz, M. Sharir, Computing depth orders and related problems, Computational Geometry 5 (1995) 187–206.
[5] P.K. Agarwal, J. Matoušek, Dynamic half-space range reporting and its applications, Algorithmica 14 (1995) 325–345.
[6] H. Alt, R. Fleischer, M. Kaufmann, K. Mehlhorn, S. Näher, S. Schirra, C. Uhrig, Approximate motion planning and the complexity of the boundary of the union of simple geometric figures, Algorithmica 8 (1992) 391–406.
[7] E.M. Arkin, K. Kedem, J.S.B. Mitchell, J. Sprinzak, M. Werman, Matching points into pairwise-disjoint noise regions: combinatorial bounds and algorithms, ORSA J. Comput. 4 (1992) 375–386.

[8] J. Bentley, J. Saxe, Decomposable searching problems I: Static-to-dynamic transformation, J. Algorithms 1 (1980) 301–358.
[9] H. Brönnimann, M.T. Goodrich, Almost optimal set covers in finite VC-Dimension, Discrete Comput. Geom. 14 (1995) 263–279.
[10] V. Chvatal, A greedy heuristic for the set-covering problem, Math. Oper. Res. 4 (1979) 233–235.
[11] K.L. Clarkson, Algorithms for polytope covering and approximation, in: Proc. 3rd Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, Vol. 709, 1993, pp. 246–252.
[12] M. de Berg, Linear size binary space partitions for fat objects, in: Proc. 3rd European Symp. Algorithms, Lecture Notes in Computer Science, Vol. 979, 1995, pp. 252–263; also: Technical Report UU-CS-1998-12, Department of Computer Science, Utrecht University, 1998.
[13] A. Efrat, A. Itai, Improvements on bottleneck matching and related problems using geometry, in: Proc. 12th ACM Symp. Comput. Geom., 1996, pp. 301–310; see also: A. Efrat, A. Itai, M.J. Katz, Geometry helps in bottleneck matching and related problems, Algorithmica, to appear.
[14] A. Efrat, M.J. Katz, F. Nielsen, M. Sharir, Dynamic data structures for fat objects and their applications, in: Proc. 5th Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science, Vol. 1272, 1997, pp. 297–306.
[15] A. Efrat, M.J. Katz, F. Nielsen, M. Sharir, Dynamic data structures for fat objects and their applications, Technical Report 99-06, Department of Mathematics and Computer Science, Ben-Gurion University, 1999.
[16] R.J. Fowler, M.S. Paterson, S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, Inform. Process. Lett. 12 (3) (1981) 133–137.
[17] M.J. Katz, 3-D vertical ray shooting and 2-D point enclosure, range searching, and arc shooting, amidst convex fat objects, Computational Geometry 8 (1997) 299–316.
[18] M.J. Katz, M.H. Overmars, M. Sharir, Efficient hidden surface removal for objects with small union size, Computational Geometry 2 (1992) 223–234.
[19] J. Matoušek, Efficient partition trees, Discrete Comput. Geom. 8 (1992) 315–334.
[20] J. Matoušek, Range searching with efficient hierarchical cuttings, Discrete Comput. Geom. 10 (1993) 157–182.
[21] M.H. Overmars, Point location in fat subdivisions, Inform. Process. Lett. 44 (1992) 261–265.
[22] M.H. Overmars, J. van Leeuwen, Maintenance of configurations in the plane, J. Comput. Syst. Sci. 23 (1981) 166–204.
[23] M.H. Overmars, A.F. van der Stappen, Range searching and point location among fat objects, J. Algorithms 21 (1996) 629–656.
[24] M. Sharir, P.K. Agarwal, Davenport Schinzel Sequences and Their Geometric Applications, Cambridge University Press, New York, 1995.
[25] 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 3 (1993) 353–373.
[26] A.F. van der Stappen, M.H. Overmars, Motion planning amidst fat obstacles, in: Proc. 10th ACM Symp. Comput. Geom., 1994, pp. 31–40.