Dynamic Data Structures for Fat Objects and Their Applications[*]

September 21, 1999

Abstract

We present several efficient dynamic data structures for point-enclosure queries involving convex fat objects in $\mathbb{R}^2$ or $\mathbb{R}^3$. 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 three problems: (i) Finding a perfect matching between a set of points and a set of convex fat objects. (ii) Finding a piercing set for a collection of convex fat objects, whose size is within a constant factor off the optimum size. (iii) Constructing a data structure for answering *bounded-length segment-shooting* queries: Given a set of fat objects in the plane as above, and a query oriented segment $\overrightarrow{r}$, whose length is relatively short, find the first object hit by $\overrightarrow{r}$.

1 Introduction

A convex object c in $\mathbb{R}^d$ is α-fat, for some parameter $\alpha > 1$, if there exists an axis-parallel cube s^+ containing c and an axis-parallel cube s^- that is contained in c, such that the ratio between the edge lengths of s^+ and s^- is at most α. Practical instances of many geometric problems tend to have fat objects as input. Fat objects have several desirable properties, which were used by many authors to obtain more efficient solutions to a variety of algorithmic problems, when the underlying objects are fat. See [3, 5, 7, 15, 16, 21, 23, 26, 27, 28] for a sample of these results.

In a recent paper [15], Katz has designed a data structure of nearly linear size for certain kinds of queries involving a set 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

[*]A full version of this paper appears in *http://www.math.tau.ac.il/~alone/dfat.ps.gz*

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 [18, 19]. Although not noted in [15], this structure can be maintained dynamically, using standard techniques, as will be described below.

In this paper we continue the work of [15]. We assume that our set $\mathcal{C}$ of objects consists of convex α-fat objects, for some fixed small constant $\alpha > 1$, and we present compact dynamic data structures for $\mathcal{C}$ that enable us to answer a point enclosure query 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 Section 3 we present three applications of our data structures as stated in the abstract.

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 (possibly intersecting) convex *fat* objects in $\mathbb{R}^2$ or $\mathbb{R}^3$. 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 [15], 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 $\mathcal{C}$, 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.) This structure can be maintained dynamically, using standard techniques, as follows: Using the same ideas as in [6], we partition $\mathcal{C}$ into at most $\log n$ subsets, each containing 2^k objects, for distinct integers $k \leq \log n$. As shown in [6], 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. In this manner we only pay (in an amortized sense) an additional logarithmic factor for both insertions and deletions, and it can be shown that this bound can also be obtained in the worst case.

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 the data structure of Matoušek [18], 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.[1]

[1]Throughout the paper, ε stands for an arbitrarily small positive constant parameter.

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 [4, Thm. 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 $O(n^\varepsilon)$. 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 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 $O(n^{1+\varepsilon}/\sqrt{s})$, 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 $O(s/n^{1-\varepsilon})$. The results of this subsection are summarized in Theorem 2.2 below.

2.2 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 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. 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 $r_q \leq E^*(\theta_q, \phi_q)$, 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 $O((n^*)^{2+\varepsilon})$ time, where $n^* = |\mathcal{C}^*|$. Using this structure, we can determine whether $r_q \leq E^*(\theta_q, \phi_q)$ in $O(\log n)$ 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 below.

In the case that $\mathcal{C} = \{B_1 \ldots B_n\}$ is a set of n (not necessarily congruent) balls in $\mathbb{R}^3$, we use known techniques to obtain exactly the same bounds. These bounds also appear in Theorem 2.2.

2.3 The planar case

In this subsection we consider the case where $\mathcal{C}$ is a collection of general convex α-fat objects in the plane. As before, we assume that each object in $\mathcal{C}$ has constant description complexity. In the full version of this paper, we obtain the following result, which is a slight improvement over the data structure of Katz [15]:

Theorem 2.1 *We can store a (static) set $\mathcal{C}$ of n convex α-fat objects in the plane, into a data structure of size $O(n \log n)$, using $O(n \log^2 n)$ preprocessing time, such that one can determine in time $O(\log^2 n)$, whether a query point is contained in some object of $\mathcal{C}$.*

A dynamic data structure Again, our goal is to preprocess $\mathcal{C}$ into a dynamic data structure that can support insertions and deletions of objects, and queries where we are given a point q and wish to determine whether q lies in the union of $\mathcal{C}$ and, if so, to report an object of $\mathcal{C}$ containing q, or, alternatively, report all such objects. We use the data structure $\mathcal{T}$ of Theorem 2.1, so that given a query point q, we can obtain in $O(\log n)$

time a collection of $O(\log n)$ canonical subsets of $\mathcal{C}$, such that each object containing q appears in one of these subsets, and such that each subset $\mathcal{C}^*$ has a point p^* common to all its members. For each object $c \in \mathcal{C}^*$, we can represent ∂c as a continuous function $r = f_c(\theta)$ in polar coordinates about p^*, and each pair of these functions intersect at most a constant number s of times. Then q is contained in an element of $\mathcal{C}^*$ if and only if $r_q \leq E^*(\theta_q)$, where E^* is the upper envelope of these functions, and where (r_q, θ_q) are the polar coordinates of q about p^*. We therefore need to maintain the upper envelopes E^* for the canonical sets, so that searches and updates of them can be performed efficiently. For this we can use (a simplified version of) the shallow-level data structure of Agarwal et al. [2] mentioned in the preceding subsection. Recall that the complexity of E^* is $\lambda_s(n^*)$, where $n^* = |\mathcal{C}^*|$ and where $\lambda_s(n)$ is the maximum length of (n, s) Davenport-Schinzel sequences [25]. It follows from [2] that we can construct a data structure of size $O(n^{1+\varepsilon})$, in time $O(n^{1+\varepsilon})$, using which we can answer a query in $O(\log n)$ time (or in $O(\log n + k)$ time, for reporting all k objects of $\mathcal{C}^*$ containing q), and perform an insertion or a deletion in time $O(n^{\varepsilon})$. Combining all these substructures, using the decomposition technique of van Kreveld [17, Corollary 5.2 (ii)], into one overall structure, we obtain the bounds appearing in Theorem 2.2 below.

The case of fat polygons: We can do somewhat better if the objects in $\mathcal{C}$ are convex α-fat polygons in $\mathbb{R}^2$. Details are omitted due to lack of space, and appear in the full version. The results we obtain are listed in Theorem 2.2 below.

Theorem 2.2 *Let $\mathcal{C}$ be a set of n convex α-fat objects in $\mathbb{R}^d$, each having a constant description complexity, for some fixed constant $\alpha > 1$ and for $d = 2, 3$. For any parameter $n \leq s \leq n^2$, we can preprocess $\mathcal{C}$ into a data structure, such that finding an object of $\mathcal{C}$ containing a query point (*point-enclosure query*), and inserting or deleting an object into/from $\mathcal{C}$ can be done in the time listed in the following table: In all cases below we can also report all objects containing a query point in time $O(Q(n) + k)$, where $Q(n)$ is the time for a point-enclosure query, and k is the number of reported objects.*

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:	$O(n^{2+\varepsilon})$	$O(s)$	$O(s)$	$O(n^{1+\varepsilon})$	$O(n\log^3 n)$
point-enc. query:	$O(\log^4 n)$	$O(\frac{n^{1+\varepsilon}}{\sqrt{s}})$	$O(\frac{n^{1+\varepsilon}}{\sqrt{s}})$	$O(\log n)$	$O(\log^3 n)$
Update:	$O(n^{1+\varepsilon})$	$O(s/n^{1-\varepsilon})$	$O(s/n^{1-\varepsilon})$	$O(n^{\varepsilon})$	$O(\log^4 n)$

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 $\mathbb{R}^3$, and let P be a set of n points. We want to solve the *matching problem*, which is to match each point of P to a distinct object that contains it. Please see the full version of this paper for motivation and a list of relevant results. We can solve the matching problem by applying the matching algorithm of Efrat and Itai [11]. 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:

Theorem 3.1 *Let $\mathcal{C}$ be a set of n convex α-fat objects, each of a constant description complexity, in $\mathbb{R}^d$ (for $d = 2, 3$), 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 $O(n^{3/2})$ for general objects and polygons in $\mathbb{R}^2$.*

3.2 Piercing fat objects

Let $\mathcal{C}$ be a (static) set of n objects in $\mathbb{R}^d$. A set of points $\mathcal{P}$ in $\mathbb{R}^d$ is a *piercing set* for $\mathcal{C}$ if $\mathcal{P}$ intersects every object in $\mathcal{C}$. Finding a minimal piercing set is NP-complete for $d \geq 2$ [13], so it is natural to seek approximate solutions, in which the size of the computed piercing set is not much larger than the optimum size. The problem of finding a minimal piercing set is a special instance of the well known *set cover* problem, so we can apply the greedy algorithm for finding a set cover [9] to obtain, in polynomial time, a piercing set whose size is larger than the optimum size by a factor of $(1+\log l)$, where $l \leq n$ is the *depth* of the arrangement of $\mathcal{C}$. Brönnimann and Goodrich [8] presented a polynomial-time algorithm for computing a set cover in which the approximation factor depends both on the optimum cover size c 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 $O(c \log c)$.

In this subsection we present efficient approximate algorithms for the case of fat objects in two or three dimensions. The algorithms produce piercing sets whose size is within a constant factor off the optimum size.

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 algorithm works in stages, where the i-th 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 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 optimum piercing set is at least j, whereas the size of the output is $O(j)$, so the output size is indeed within a constant factor off the optimum. 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 $d = 2, 3$. 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 $\mathcal{C}$, as follows:*

Objects:	*polytopes*	*balls*	*general*	*polygons*
dimension	*3D*	*3D*	*2D*	*2D*
Running time	$O(n^{4/3+\varepsilon})$	$O(n^{4/3+\varepsilon})$	$O(n^{1+\varepsilon})$	$O(n \log^4 n)$

3.3 Segment shooting among fat objects

Let $\mathcal{C}$ be a set of n convex α-fat objects in the plane, and let δ be the smallest diameter of any object in $\mathcal{C}$. In the bounded-size arc shooting problem, we wish to preprocess $\mathcal{C}$, so that, for a given oriented query arc $\vec{r}$ of length at most $h\delta$, for some constant h, the first object of $\mathcal{C}$ hit by r (if such an object exists) can be found efficiently. More precisely, we search for an object $c \in \mathcal{C}$ for which there is a point $z \in \vec{r}$ such that $z \in c$ and the initial portion of $\vec{r}$ preceding z does not intersect any object of $\mathcal{C}$. An efficient solution to this problem is presented in [15] for the special case where $\mathcal{C}$ consists of either (constant-complexity) polygons or disks, and the query arc is either a segment or a circular arc. Here we present a solution for the general case, assuming the query arcs are segments. The data structure we describe is based on the data structure for point enclosure, 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]). Given an oriented query segment $\vec{r} = \overrightarrow{pq}$, we first check whether its first endpoint p lies in one (or more) of the objects of $\mathcal{C}$, using the data structure of Theorem 2.1. Assuming it does not, we proceed as follows. We obtain in $O(\log^2 n)$ time a collection of $O(\log^2 n)$ canonical subsets of $\mathcal{C}$, such that each object that is intersected by $\vec{r}$ appears in at least one of these subsets, and such that each subset $\mathcal{C}^*$ has a point p^* common to all its members. This is done by performing a point enclosure query for each of the points in a point set of constant size that is constructed as a function of $\vec{r}$, in its vicinity; see [23, 24].

We now perform a segment shooting query in each of these subsets $\mathcal{C}^*$ to obtain a collection of $O(\log^2 n)$ candidate objects from which we choose the one that is hit first by $\vec{r}$. We next show how to perform efficiently a segment shooting query in a subset $\mathcal{C}^*$ with a common point.

Assume for simplicity of exposition that the common point of $\mathcal{C}^*$ is the origin O, and let U denote the union of the objects in $\mathcal{C}^*$. Let ℓ be the oriented line containing $\vec{r}$ and oriented in the same direction. We will find v, the intersection point of ℓ with U, that lies ahead of p and is closest to p, if a such a point exists. By checking the relative position of p, q and v along ℓ, we discover whether v lies on $\vec{r}$. We divide U into two regions, U^u, the region above the x-axis, and U^d, the region below the x-axis. We find the first intersection point between ℓ and each of them, and select the one that is closer to p. Let us focus on U^u and let $\xi = \partial U^u$. We consider ξ as the graph of the upper envelope of the boundaries of the sets of $\mathcal{C}^*$, expressed as functions in polar coordinations about O. If we assume that each pair of the boundaries of the objects of $\mathcal{C}$ intersect in at most a constant number s of points, then the number of vertices of ξ (i.e. intersection pairs of boundaries that lie on ξ) is at most $(\lambda_s|\mathcal{C}^*|)$. We construct a binary tree $\mathcal{T}$ as follows. We compute the convex hull $CH(\xi)$, using the optimal algorithm of Nielsen and Yvinec [20]. Clearly, ℓ intersects U^u if and only if the straight line containing ℓ intersects $CH(\xi)$. We associate $root(\mathcal{T})$ with $CH(\xi)$. Next, we find a vertex $m \in \xi$ which divides ξ into two parts ξ_1 and ξ_2 having approximately the same number of vertices. Let the endpoints of these parts be t_1, m and m, t_2. We add edges that connect these endpoints to O, so that ξ_1 (resp. ξ_2) together with these edges form the boundary of a star-shaped region $\mathcal{K}_1$ (resp. $\mathcal{K}_2$), that contains the origin O. We compute $CH(\mathcal{K}_1)$ and $CH(\mathcal{K}_2)$, and associate the two children of $root(\mathcal{T})$ with $CH(\mathcal{K}_1)$ and $CH(\mathcal{K}_2)$. We continue to construct $\mathcal{T}$ recursively, and stop when $\mathcal{K}_v$, for a leaf v, consists of the boundary of a single object.

Answering a query. Assume none of the endpoints of $\vec{r}$ is in U. We check whether ℓ intersects $CH(\xi)$, the convex hull of U^u. If it does not then we deduce that $\vec{r}$ does not intersect U^u. If it does, we move on to the two children w_1, w_2 of the root of $\mathcal{T}$, storing the regions $CH(\mathcal{K}_1)$ and $CH(\mathcal{K}_2)$, respectively. Clearly ℓ intersects at least one of these regions. If it intersects only one of them, we continue recursively in the appropriate subtree. The interesting case is when ℓ intersects both regions at two respective openly-disjoint intervals I_1 and I_2. suppose, with no loss of generality, that I_1 precedes I_2 along ℓ. We distinguish between the following three cases:

(i) p precedes I_1 on ℓ: In this case, we recurse only at w_1,

(ii) p succeeds I_1 on ℓ: In this case, we recurse only at w_2. (If p also succeeds I_2, we can stop the recursion and conclude that $\vec{r}$ does not meet U^u.)

(iii) $p \in I_1$: In this case it is not clear whether the first hitting point of r with U^u occurs within $\mathcal{K}_1$ or within $\mathcal{K}_2$, since it is possible that $\vec{r}$ emerges from $CH(\mathcal{K}_1)$ without hitting $\mathcal{K}_1$ itself. (In the special case where q also lies in I_1, or precedes I_2 in ℓ, we know that $\vec{r}$ does not hit $\mathcal{K}_2$, so we recurse only at w_1. To resolve this problem, let p' denote the endpoint of I_1 lying ahead of p (between p and I_2). If $p' \in U$ then it suffices to recurse only on w_1.Otherwise, we perform a new shooting query with $\vec{r'} = \overrightarrow{p'p}$ (we shoot in the opposite direction, from p' towards p). However, we start the recursion from the node w_1 of $\mathcal{T}$. The important observation is that case (iii) will never arise in any successive step of the query, so the query will follow a simple path in $\mathcal{T}$ from w to some leaf, or else it stops earlier with the conclusion that $\vec{r}'$ does not hit U^u. If we reach a leaf Z, we check in constant time whether $\vec{r}'$ hits the single object associated with Z, to determine whether $\vec{r}'$ hits U^u.

(iii.a) If $\vec{r}'$ does not hit U^u, we recurse (with the original query) only at w_2. Note that case (iii) will not arise again, at any future recursive step.

(iii.b) If $\vec{r}'$ hits U^u, we recurse only at w_1. (Now it is possible for case (iii) to arise again.)

If we have not terminated the query with the conclusion that $\vec{r}$ does not hit U^u, we will end at some leaf Z of $\mathcal{T}$. As in case (iii) above, we find the first hitting point of $\vec{r}$ with the single object o whose boundary is stored at Z, and return this point (and o) as output. The cost of this process is $O(\log^3 n)$; The cost of intersecting ℓ with any convex hull is $O(\log n)$; the query follows a path of length $O(\log n)$ in $\mathcal{T}$, and at each node of the path we may need to execute a sub-query (if we are in case (iii)) that traces another path of length $O(\log n)$. We repeat this process on U^d, and apply it to each of the $O(\log^2 n)$ canonical sets $\mathcal{C}^*$. Thus we have:

Theorem 3.3 *Let $\mathcal{C}$ be a collection of convex α-fat objects in $\mathbb{R}^2$, each with constant description complexity. We can preprocess $\mathcal{C}$ in time $O(n^{1+\varepsilon})$, into a data structure of size $O(n^{1+\varepsilon})$, so that given an oriented segment $\vec{r}$ of length $\leq h\delta$, where h is a constant and δ is the smallest diameter of any object in $\mathcal{C}$, we can find the first object of $\mathcal{C}$ hit by $\vec{r}$ (if such an object exists) in time $O(\log^5 n)$.*

Dynamization and Generalization. In time $O(\log n)$ we can easily add objects to the data structure, while increasing the query time by $O(\log n)$, using the standard technique of [6]. Concerning deletions, we suspect that one can maintain in $O(\log^2 n)$ the convex hull of a set of convex objects, provided they all share a point. We believe that this is possible using extensions of the ideas of [14]. However, we are not aware of any work on this problem. The special case where the objects are fat polygons, can be handled trivially using the technique of [22], so that each insertion or deletions takes $O(\log^2 n)$, and the time for a query is the same.

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 and M. Sharir, Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications, *Proc. 11th ACM Symp. Comput. Geom.*, 1995, 39–50.

[3] P.K. Agarwal, M.J. Katz and M. Sharir, Computing depth orders and related problems, *Computational Geometry: Theory and Applications* 5 (1995), 187–206.

[4] P.K. Agarwal and J. Matoušek, Dynamic half-space range reporting and its applications, *Algorithmica* 14 (1995), 325–345.

[5] 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), 391–406.

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

[7] 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*, 1994, 61–72.

[8] H. Brönnimann and M.T. Goodrich, Almost optimal set covers in finite VC-Dimension, *Discrete and Computational Geometry* 14 (1995), 263–279.

[9] V. Chvatal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, 4 (1979), 233–235.

[10] K. L. Clarkson, K. Mehlhorn, and R. Seidel, Four results on randomized incremental constructions, *Computational Geometry: Theory and Applications* 3 (1993), 185–212.

[11] A. Efrat and A. Itai, Improvements on bottleneck matching and related problems, using geometry, *Proc. 12th ACM Symp. Comput. Geom.*, 1996, 301–310. See also: A. Efrat, M.J. Katz and A. Itai, Improvements on bottleneck matching and related problems, using geometry, in preparation.

[12] A. Efrat and M. Sharir, On the Complexity of the Union of Fat Objects in the Plane, *Proceedings 13 Annual Symposium on Computational Geometry*, 1997, to appear.

[13] R.J. Fowler, M.S. Paterson, and S.L. Tanimoto, Optimal packing and covering in the plane are NP-complete, *Information Processing Letters* 12 (3) (1981), 133–137.

[14] J. Hershberger and S. Suri, Applications of a semi-dynamic convex hull algorithm, *Proceedings 2 Scand. Workshop on Algorithms Theory, Lecture Notes in Computer Science*, 1990, vol. 447, Springer-Verlag, New York–Berlin–Heidelberg, 380–392.

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

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

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

[18] J. Matoušek, Efficient partition trees, *Discrete and Computational Geometry* 8 (1992), 315–334.

[19] J. Matoušek, Range searching with efficient hierarchical cuttings, *Discrete and Computational Geometry* 10 (1993), 157–182.

[20] F. Nielsen and M. Yvinec, Output-Sensitive Convex Hull Algorithms of Planar Convex Objects, *Research Report* 2575 INRIA, BP93, 06902 Sophia-Antipolis, France.

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

[22] H. Overmars and J. van Leeuwen, Maintenance of configurations in the plane, *J. Comput. Syst. Sci.* 23 (1981), 166–204.

[23] M.H. Overmars and A.F. van der Stappen, Range searching and point location among fat objects, *Proc. 2nd Annual European Symp. on Algorithms*, 1994, 240–253. (See also: Tech. Report UU-CS-1994-30, Dept. of Computer Science, Utrecht University.)

[24] O. Schwarzkopf and J. Vleugels, Range Searching in Low-Density Environments, Tech. Report UU-CS-1996-26, Dept. of Computer Science, Utrecht University.

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

[26] A.F. van der Stappen, *Motion Planning amidst Fat Obstacles*, Ph.D. thesis, Utrecht University, 1994.

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

[28] A.F. van der Stappen and M.H. Overmars, Motion planning amidst fat obstacles, *Proc. 10th ACM Symp. Comput. Geom.*, 1994, 31–40.