3-D Vertical Ray Shooting and 2-D Point Enclosure, Range Searching, and Arc Shooting Amidst Convex Fat Objects

Matthew Katz

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

3-D Vertical Ray Shooting and 2-D Point Enclosure, Range Searching, and Arc Shooting Amidst Convex Fat Objects

Matthew Katz

N° 2583

Juin 1995

PROGRAMME 4

ISSN 0249-6399

3-D Vertical Ray Shooting and 2-D Point Enclosure, Range Searching, and Arc Shooting Amidst Convex Fat Objects

Matthew Katz*

Programme 4 — Robotique, image et vision
Projet Prisme

Rapport de recherche n° 2583 — Juin 1995 — 23 pages

Abstract: We present a new data structure for a set of n convex simply-shaped fat objects in the plane, and use it to obtain efficient and rather simple solutions to several problems including (i) *vertical ray shooting* — preprocess a set $\mathcal{K}$ of n non-intersecting convex simply-shaped flat objects in 3-space, whose xy-projections are fat, for efficient vertical ray shooting queries, (ii) *point enclosure* — preprocess a set $\mathcal{C}$ of n convex simply-shaped fat objects in the plane, so that the k objects containing a query point p can be reported efficiently, (iii) *bounded-size range searching* — preprocess a set $\mathcal{C}$ of n convex fat polygons, so that the k objects intersecting a 'not-too-large' query polygon can be reported efficiently, and (iv) *bounded-size segment shooting* — preprocess a set $\mathcal{C}$ as in (iii), so that the first object (if exists) hit by a 'not-too-long' oriented query segment can be found efficiently. For the first three problems we construct data structures of size $O(\lambda_s(n)\log^3 n)$, where s is the maximum number of intersections between the boundaries of the (xy-projections) of any pair of objects, and $\lambda_s(n)$ is the maximum length of (n,s) Davenport-Schinzel sequences. The data structure for the fourth problem is of size $O(\lambda_s(n)\log^2 n)$. The query time in the first problem is $O(\log^4 n)$, the query time in the second and third problems is $O(\log^3 n + k\log^2 n)$, and the query time in the fourth problem is $O(\log^3 n)$.

We also present a simple algorithm for computing a depth order for a set $\mathcal{K}$ as in (i), that is based on the solution to the vertical ray shooting problem. (A depth order for $\mathcal{K}$, if exists, is a linear order of $\mathcal{K}$, such that, if $K_1, K_2 \in \mathcal{K}$ and K_1 lies vertically above K_2, then K_1 precedes K_2.)

Key-words: computational geometry, data structures, output-sensitive algorithms, visibility

(Résumé : tsvp)

Supported in part by ESPRIT Basic Research Action No. 7141 (project ALCOMII).

*INRIA, B.P.93, 06902 Sophia-Antipolis cedex (France), Phone: +33 93 65 77 72 E-mail: Matthew.Katz@sophia.inria.fr.

Unité de recherche INRIA Sophia-Antipolis
2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex (France)
Téléphone : (33) 93 65 77 77 – Télécopie : (33) 93 65 77 65

Lancer de rayons verticaux 3-D et inclusion de points 2-D, requêtes de domaines, et lancer d'arcs dans un ensemble d'objets convexes et épais

Résumé : Nous présentons une nouvelle structure de données pour un ensemble de n objets épais de complexité bornée dans le plan. Nous l'utilisons pour obtenir des solutions simples et efficaces pour plusieurs problèmes, parmi lesquels (i) *lancer de rayons verticaux* — un ensemble K de n objets convexes de complexité bornée et plats en 3-D dont les projections horizontales sont épaisses, peut être prétraité pour le lancer de rayons verticaux. (ii) *inclusion de points* — un ensemble $\mathcal{C}$ de n objets du plan, épais, convexes et de complexité bornée peut être prétraité pour que les k objets contenant un point donné soient déterminés efficacement. (iii) *requêtes de domaines de taille bornée* — un ensemble $\mathcal{C}$ de n polygones épais, convexes peut être prétraité pour que les k objets ayant une intersection avec un polygone requête 'pas trop grand' soient déterminés rapidement. (iv) *lancer de segments de longueur bornée* — un ensemble $\mathcal{C}$ défini comme ci dessus en (iii) peut être prétraité pour que le premier objet (si il existe) rencontré par un segment orienté 'pas trop grand' soit trouvé efficacement. Pour les trois premiers problèmes, nous construisons une structure de données de taille $O(\lambda_s(n)\log^3 n)$, où s est le nombre maximal d'intersections entre deux objets et $\lambda_s(n)$ la longueur maximale d'une (n, s) séquence de Davenport-Schinzel. Pour le quatrième problème, la taille de la structure est $O(\lambda_s(n)\log^2 n)$. Le temps de requête pour le premier problème est $O(\log^4 n)$, $O(\log^3 n + k\log^2 n)$ pour les problèmes (ii) et (iii), et $O(\log^3 n)$ pour le dernier.

Nous proposons également un algorithme simple pour calculer un ordre de profondeur pour un ensemble défini comme en (i), basé sur une solution du problème du lancer de rayons verticaux. (Un ordre de profondeur, est un ordre total sur les objets de $\mathcal{K}$ tel que si K_1 est au dessus de K_2 alors K_1 est plus petit que K_2). L'algorithme est capable de déterminer si un tel ordre existe, contrairement à l'algorithme d'Agarwal et al [2] qui peut produire un ordre incorrect si un ordre correct n'existe pas, notre algorithme est plus efficace en pratique que celui d'Agarwal et al.

Mots-clé : géométrie algorithmique, structures de données, algorithmes adaptatifs, visibilité

1 Introduction

Informally, an object c in the plane is 'fat' if it is not too long and thin and it does not have long and thin parts. For example, any disk is fat. In the last few years it was recognized that fat objects possess several desirable properties. This has led many authors to reconsider various known problems in computational and combinatorial geometry under the assumption that the input objects are fat, which resulted in an array of more efficient solutions than in the general case. See [2, 4, 9, 11, 18, 22, 25, 26, 30, 31, 32, 34] for a small sample of these results. In practice, the fatness assumption often holds, thus the search for more efficient solutions for collections of fat objects is definitely justified.

We use the following definition of fatness (also used in [2]). An object c in the plane is α-fat, for some parameter $\alpha > 1$, if there exists an axis-parallel square s^+ containing c and an axis-parallel square s^- that is contained in c, such that the ratio between the edge lengths of s^+ and s^- is at most α. Lemma 1.1 below, which is used in Section 2, follows immediately from a result of van der Stappen [30, Chap. 2, Theorem 2.6]. It shows that when the underlying objects are convex, our definition implies an alternative definition of fatness that was introduced by van der Stappen et al. [31] for objects that are not necessarily convex. (van der Stappen et al. actually use disks rather than axis-parallel squares in their definition, but the two definitions are clearly equivalent, and squares are more convenient for our applications. The converse of Lemma 1.1 is also true. That is, if c is a convex α-fat object by the definition of van der Stappen et al., then it is $O(\alpha)$-fat by our definition. This was proved implicitly in [26].)

Lemma 1.1 *Let c be a convex α-fat object in the plane. Then, for every axis-parallel square s that does not entirely contain c and whose center p lies in c, we have*

$$\frac{area(s)}{area(c \cap s)} \leq \beta ,$$

where $\beta = O(\alpha^2)$.

In this paper we present a new data structure for a set of n (possibly intersecting) convex simply-shaped α-fat objects in the plane whose size is nearly linear in n, and use several variants of it to obtain efficient and rather simple solutions to several basic problems, as stated in the abstract and in more details below. We also present a simple algorithm for computing a depth order for an appropriate set of objects in 3-space (see below), that is based on the solution to the vertical ray shooting

problem. In many practical situations α is a small constant. In these situations the solutions that are presented here are both more efficient and simpler than the corresponding general solutions (that are based on the theory of epsilon-nets), and thus, we believe, are more attractive. In this introduction, we state our results and compare them with previous related results, assuming, as usual, that α is a constant. In the subsequent sections, however, we will treat α as a not necessarily constant parameter, and show that the results are efficient also when α is quite large. (For example, our solution to the point enclosure problem is faster than the best known general solutions using nearly linear storage for any $\alpha = O(n^{1/4-\varepsilon})$, for $\varepsilon > 0$.)

Vertical ray shooting. Let $\mathcal{K}$ be a set of n non-intersecting convex simply-shaped (not necessarily horizontal) flat objects in 3-space, and assume that the xy-projections of the objects in $\mathcal{K}$ are α-fat. We want to preprocess the set $\mathcal{K}$ for efficient *vertical ray shooting*, that is, we want to preprocess $\mathcal{K}$ so that, for a given query point p, the object of $\mathcal{K}$ lying immediately below p (if such an object exists) can be found quickly.

Consider first the following simple and well known solution to the vertical ray shooting problem for a set $\mathcal{D}$ of n *horizontal* disks. (We present this solution only to emphasize the two subtle differences between a set such as $\mathcal{D}$ and the set $\mathcal{K}$. These differences make the vertical ray shooting problem for the set $\mathcal{K}$ significantly more difficult.) Sort the disks in $\mathcal{D}$ by their z-coordinate and store them, in sorted order, at the leaves of a balanced binary tree T. For a node v of T, let $\mathcal{D}_v$ denote the set of disks that are stored at the leaves of the subtree of T rooted at v, and let U_v denote the union of the xy-projections of the disks in $\mathcal{D}_v$. Associate with v the region U_v and preprocess U_v for efficient point location queries. The combinatorial complexity of the boundary of the union of m disks in the plane is only $O(m)$ [19], and this union can be computed in $O(m \log m)$ time [5]. Moreover, the union can also be preprocessed for logarithmic-time point location in $O(m)$ space and $O(m \log m)$ time. It follows that the whole structure requires $O(n \log n)$ space, and it can be computed in $O(n \log^2 n)$ time.

Now, given a query point p in space, we can easily detect the disk lying immediately below it, if such a disk exists, in $O(\log^2 n)$ time, as follows. First search in T with the z-coordinate (i.e., height) of p to obtain the set of all disks of $\mathcal{D}$ whose height is less than the height of p, as a collection of $O(\log n)$ disjoint canonical sets. Now consider these sets one by one, beginning with the set whose height range is the highest, until a set $\mathcal{D}_v$ for which the xy-projection of p lies in the corresponding region U_v is encountered. Finally, perform $O(\log n)$ point location queries in the sub-

tree of v to detect the highest disk of $\mathcal{D}_v$ whose projection contains the projection of p. Clearly the total time spent is $O(\log^2 n)$.

The above solution is based on the following two properties of the set $\mathcal{D}$: (i) The combinatorial complexity of the union of the xy-projections of the objects in $\mathcal{D}$ is small (linear in this case), and (ii) a *depth order* of $\mathcal{D}$ exists (i.e., a linear order of $\mathcal{D}$ such that if $D_1, D_2 \in \mathcal{D}$ and D_1 lies vertically above D_2 then D_1 precedes D_2), and can be computed efficiently (in $O(n \log n)$ time in this case). Thus, the above solution can be applied also to other classes of objects satisfying these conditions, e.g., horizontal fat triangles, where the first property follows from [22] (see [10]).

However, in the general case considered here, if a depth order for $\mathcal{K}$ does not exist, then this solution is not applicable. Even if a depth order exists, the best known algorithm for computing it, as given in [2], is still inefficient; it runs in roughly $O(n^{3/2})$ time, and requires, in addition, that the xy-projections of the given objects be more or less of the same size. Moreover, the size of the union of the xy-projections of the objects in $\mathcal{K}$ is not necessarily close to linear. (A counter example, where the union has quadratic complexity, is given in [22]. However, in this construction the objects do not have constant description complexity. It appears to be an open problem whether the size of the union of convex simply-shaped α-fat objects is close to linear.)

In Section 2 we present an efficient solution to the general case, that is, to the vertical ray shooting problem for the set $\mathcal{K}$. Our solution requires $O(\lambda_s(n) \log^3 n)$ storage and $O(\lambda_s(n) \log^4 n)$ preprocessing time, and a vertical ray shooting query can be answered in $O(\log^4 n)$ time, where s is the maximum number of intersections between the boundaries of the xy-projections of any pair of objects in $\mathcal{K}$. [1]

Point enclosure. In Section 3 we present efficient solutions to two versions of the point enclosure problem. Let $\mathcal{C}$ be a set of n convex simply-shaped α-fat objects in the plane. The first (standard) version, referred to as the *point enclosure* problem, is to preprocess $\mathcal{C}$ so that, for a query point p, one can efficiently report the objects of $\mathcal{C}$ containing p. For the second version, we also assume that the objects of $\mathcal{C}$ are colored by a set of $1 \leq m \leq n$ colors. In this problem, which is sometimes called the *generalized point enclosure* problem, we want to preprocess $\mathcal{C}$ so that, for a query point p, one can efficiently report the distinct colors u for which there exists an object of $\mathcal{C}$ of color u containing p.

[1] $\lambda_s(n)$ is the maximum length of (n, s) Davenport-Schinzel sequences, which is nearly linear in n for any fixed s [15, 29].

Efficient solutions for these problems for a collection of disks or fat triangles are known. In [28] Sharir has presented a randomized algorithm for the point enclosure problem for a collection of disks in the plane. It requires $O(n \log^2 n)$ expected preprocessing time and $O(n \log n)$ expected storage, and the query cost is $O(\log n + k \log n)$, where k is the output size. His method can also be applied to the case of fat triangles, with a slight increase in the preprocessing time and storage. Another algorithm is an appropriate simplification of the algorithm presented above for the case of disks. Its query cost is $O(\log n + k \log^2 n)$. This method may also be applied for the case of fat triangles.

However, both the method of Sharir and the latter method are not appropriate in our case, since we can not assume that the combinatorial complexity of the union of the objects in $\mathcal{C}$ is small (see discussion above), as required by these methods. We are not aware of any other solutions to the point enclosure problem with nearly linear preprocessing time and a query time which is of the form $O(\text{polylog}(n) + k\,\text{polylog}(n))$, except, of course, for axis-parallel rectangles (using standard orthogonal range searching techniques [23]). When removing the fatness assumption, the best known solutions using nearly linear storage (see e.g. [8, 20, 21]) have roughly $O(\sqrt{n} + k)$ query time.

The generalized problem was studied in a series of papers by Janardan and Lopez [17] and by Gupta et al. [12, 13, 14]. They present a solution for the case of disks [14] which requires $O(n \log^2 n)$ preprocessing time and $O(n \log n)$ storage, and the query cost is $O(\log n + k \log n)$, where k is the output size. Their solution for the case of fat triangles requires $O(n \log^2 n)$ storage and the query cost is $O(\log^3 n + k \log n)$. Another solution for the case of fat triangles with query cost $O(\log n + k \log^2 n)$ requires $O(n \log n \log\log n)$ storage, and is based on the second method above for the standard version. However, as in the standard version, these methods are not appropriate in our case.

Our solutions to the point enclosure problem and to the generalized point enclosure problem both require $O(\lambda_s(n) \log^3 n)$ storage and $O(\lambda_s(n) \log^4 n)$ preprocessing time, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$. A point enclosure query can be answered in $O(\log^3 n + k \log^2 n)$ time, and a generalized point enclosure query can be answered in $O(\log^3 n + k \log^4 n)$ time, where in the former expression k is the number of objects containing the query point, and in the latter expression k is the number of distinct colors of objects containing the query point.

Bounded-size range searching and arc shooting. In Section 4 we present a solution to the *bounded-size range searching* problem. Let $\mathcal{C}$ be as above, and let ρ be the diameter of the object in $\mathcal{C}$ with smallest diameter. In this problem, we wish to preprocess $\mathcal{C}$ so that, for a given query range R whose diameter is at most $h\rho$, one can efficiently report all objects of $\mathcal{C}$ intersecting R, where h is some constant. Overmars and van der Stappen [26] present an efficient solution to this problem assuming the objects in $\mathcal{C}$ are *disjoint*. Their solution requires $O(n \log n)$ storage to answer a query in $O(\log n)$ time. (In their setting, i.e., disjoint objects and bounded-size ranges, the output set of a query is necessarily of constant size.) Their solution is also appropriate for higher dimensions, and it can also handle non-convex (fat) polytopes. However, the solution of Overmars and van der Stappen does not apply to sets of intersecting objects. Moreover, in this problem, unlike in the point enclosure problem, it does not seem to help to assume that the underlying objects belong to some class of objects for which the combinatorial complexity of the union is small (e.g., fat triangles or disks). When removing the fatness assumption, the best known solutions using nearly linear storage (see e.g. [8, 20, 21, 33]) require roughly $O(\sqrt{n} + k)$ time.

We present efficient solutions to the bounded-size range searching problem when the objects of $\mathcal{C}$ are either (convex α-fat) polygons or disks, and the query ranges are either not necessarily convex and fat polygons or disks. Our solutions require nearly linear preprocessing time and storage, and a range query can be answered in $O(\text{polylog}(n) + k\,\text{polylog}(n))$ time. See Table 1 for the exact bounds.

input objects	query object	preprocessing	storage	query
polygons	polygon	$O(\lambda_s(n) \log^4 n)$	$O(\lambda_s(n) \log^3 n)$	$O(\log^3 n + k \log^2 n)$
polygons	disk	$O(\lambda_s(n) \log^6 n)$	$O(\lambda_s(n) \log^6 n)$	$O(\log^6 n + k \log^5 n)$
triangles	polygon	$O(n \log^4 n)$	$O(n \log^3 n)$	$O(\log^3 n + k \log^2 n)$
triangles	disk	$O(n \log^6 n)$	$O(n \log^6 n)$	$O(\log^6 n + k \log^5 n)$
disks	polygon or disk	$O(n \log^6 n)$	$O(n \log^6 n)$	$O(\log^6 n + k \log^5 n)$

Table 1: Summary of range searching results

In the *bounded-size arc shooting* problem, we wish to preprocess $\mathcal{C}$ so that, for a given oriented query arc r of length at most $h\rho$, the first object of $\mathcal{C}$ that is hit by r (i.e., the object whose intersection with r is nearest to a designated endpoint of

r) can be found efficiently. We present efficient solutions to this problem when the objects of $\mathcal{C}$ are either (convex α-fat) polygons or disks and the query arc is either a line segment or a circular arc. Our solutions require nearly linear preprocessing time and storage, and a shooting query can be answered in $O(\text{polylog}(n))$ time. See Table 2 for the exact bounds.

input objects	query object	preprocessing	storage	query
polygons	line segment	$O(\lambda_s(n)\log^3 n)$	$O(\lambda_s(n)\log^2 n)$	$O(\log^3 n)$
polygons	circular arc	$O(\lambda_s(n)\log^5 n)$	$O(\lambda_s(n)\log^5 n)$	$O(\log^6 n)$
triangles	line segment	$O(n\log^3 n)$	$O(n\log^2 n)$	$O(\log^3 n)$
triangles	circular arc	$O(n\log^5 n)$	$O(n\log^5 n)$	$O(\log^6 n)$
disks	line segment or circular arc	$O(n\log^5 n)$	$O(n\log^5 n)$	$O(\log^6 n)$

Table 2: Summary of arc shooting results

Computing a depth order. Let $\mathcal{K}$ be a set of n non-intersecting convex simply-shaped (not necessarily horizontal) flat objects in 3-space, and assume that the xy-projections of the objects in $\mathcal{K}$ are α-fat. We would like to determine whether the 'above-below' relation over $\mathcal{K}$ contains cycles, and, if it does not, to compute a depth order for $\mathcal{K}$ (i.e., a linear order of $\mathcal{K}$ such that if $K_1, K_2 \in \mathcal{K}$ and K_1 lies vertically above K_2 then K_1 precedes K_2).

In [2] Agarwal et al. present a roughly $O(n^{3/2}\log^4 n)$ algorithm for computing a depth order for $\mathcal{K}$, assuming that (i) such an order exists, and (ii) the xy-projections of the objects in $\mathcal{K}$ are more or less of the same size, more precisely, the ratio between the largest diameter and the smallest diameter is at most α. The first assumption is problematic, since it is very likely that we do not know in advance whether there exists a depth order for $\mathcal{K}$. In this case, if we apply the algorithm of [2], we might obtain an order of $\mathcal{K}$ which we will believe is a depth order while such an order does not even exist.

In Section 5 we present a simple algorithm that overcomes this problem. It either determines that the above-below relation contains cycles, or computes a depth order for $\mathcal{K}$. The second assumption above is not necessary as well. However, our algorithm is efficient only when there exists a 'good' upper bound β (that is known to us) for

the value

$$\max \left\{ \frac{area(c_i)}{area(c_i \cap c_j)} \mid K_i, K_j \in \mathcal{K},\ c_i \cap c_j \neq \emptyset \right\} ,$$

where c_i and c_j are the xy-projections of K_i and K_j. This condition is quite strong (although it is in some sense weaker than the second assumption above), but, still, it often holds in practical situations, and it allows us to remove the problematic assumption above.

The algorithm runs in $O((\beta^2 \log^2 \beta) n \log^5 n)$ time. Thus, if β is some small constant implied, e.g., by the underlying drawing package, then we get an expression that is nearly linear in n. In general, the algorithm is more efficient than the previous algorithm of Agarwal et al. when $\beta = O(n^{1/4-\varepsilon})$. Even if β is larger, i.e., $\beta = O(n^{1/2-\varepsilon})$, our algorithm may still be the preferred choice (when it is not known whether a depth order exists). The algorithm is based on the solution of Section 2 to the vertical ray shooting problem, and on an observation that allows us to maintain the underlying data structure while objects obeying a certain condition are deleted from it.

2 Vertical Ray Shooting

Using the terminology of [2], we say that a set $\mathcal{C}$ of objects in the plane has the *common point property* if all the objects in $\mathcal{C}$ have a point in common. In [2] we have established the following simple and useful results, which we state here as a lemma for the reader's convenience.

Lemma 2.1 *Let $\mathcal{C}$ be a set of n convex simply-shaped objects in the plane and assume that $\mathcal{C}$ has the common point property. Then the combinatorial complexity of the boundary of the union of the objects in $\mathcal{C}$ is $O(\lambda_s(n))$, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$, and under an appropriate model of computation, the union can be computed in time $O(\lambda_s(n) \log n)$. Moreover, it is possible to determine in $O(\log n)$ time (after $O(\lambda_s(n) \log n)$ preprocessing) whether a query point p lies within the union, and, if it does, also to obtain a witness, i.e., an object of $\mathcal{C}$ containing p.*

Since our objects are also α-fat, we can obtain in some cases better results than those implied by the above lemma. For example, in the case of fat triangles the union size is only linear in n, and the union can be computed in $O(n \log n)$ time (as shown in [2]).

Let $\mathcal{K} = \{K_1, \ldots, K_n\}$ be a set of n non-intersecting convex simply-shaped flat objects in 3-space, and assume that the xy-projections of the objects in $\mathcal{K}$ are α-fat. Let c_i denote the xy-projection of K_i, and put $\mathcal{C} = \{c_1, \ldots, c_n\}$. Clearly the objects in $\mathcal{C}$ are all convex, simply-shaped, and α-fat. For each object $c_i \in \mathcal{C}$, let s_i be a smallest axis-parallel square containing c_i, and put $\mathcal{S} = \{s_1, \ldots, s_n\}$.

In the first stage we construct a data structure for $\mathcal{S}$ so that, given a query point p in the plane, one can report all squares of $\mathcal{S}$ containing p as a disjoint union of canonical pre-stored subsets. This structure is simply a two-dimensional segment tree T (see, e.g., [27]). It can be built in $O(n \log^2 n)$ time, it uses $O(n \log^2 n)$ storage, and a query takes $O(\log^2 n)$ time, which is also the number of canonical sets constituting the output of the query.

In the second stage we process each node v of T separately. We associate with v the (non empty) rectangle s_v, equal to the intersection of all the squares in the canonical set $\mathcal{S}_v \subseteq \mathcal{S}$ that is associated with v. Denote by $\mathcal{C}_v$ the subset of $\mathcal{C}$ corresponding to $\mathcal{S}_v$, and denote by $\mathcal{K}_v$ the subset of $\mathcal{K}$ corresponding to $\mathcal{C}_v$. Clearly, any query point that has $\mathcal{S}_v$ as one of the sets in its output must necessarily lie inside s_v. We can therefore discard from $\mathcal{C}_v$ all objects that do not intersect s_v. This is justified by observing that if p is a query point in 3-space whose xy-projection has $\mathcal{S}_v$ as one of its canonical output sets, then none of the objects of $\mathcal{K}_v$, corresponding to the discarded objects in $\mathcal{C}_v$, can lie below (or above) p. For notational convenience, we continue to denote the set of projections in $\mathcal{C}_v$ that intersect s_v by $\mathcal{C}_v$. We now wish to partition the set $\mathcal{C}_v$ into a small number (dependent on α) of subsets, such that each subset has the common point property. For this we first need the following lemma. Denote by s_{min} the smallest square in $\mathcal{S}_v$ and let ρ_{min} be its edge length. We know that each of the objects in C_v intersects s_{min} (since $s_{min} \supseteq s_v$). Expand s_{min} by a factor of two about its center and denote by s^*_{min} the expanded square.

Lemma 2.2 *For each $c \in \mathcal{C}_v$, we have*

$$\frac{area(s^*_{min})}{area(c \cap s^*_{min})} \leq \gamma \ ,$$

where $\gamma = O(\alpha^2)$.

Proof. Let p be some point in $c \cap s_{min}$, and draw an axis-parallel square s centered at p with edge length $\rho_{min}/2$ (see Figure 1). Then it is impossible that s entirely contains c, since the smallest axis-parallel square containing an object in $\mathcal{C}_v$ has edge length ρ_{min}. Therefore, since c is convex and α-fat, we have, by Lemma 1.1,

$$\frac{area(s)}{area(c \cap s)} \leq \beta \ .$$

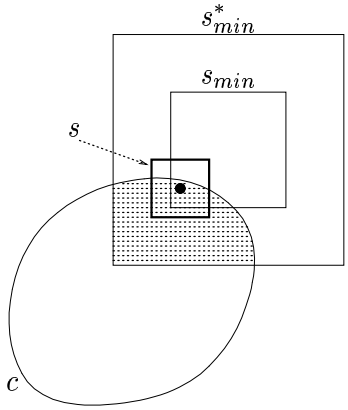

Figure 1: The area of $c \cap s^*_{min}$ must be 'large'

Moreover, s is fully contained in s^*_{min}, so

$$\frac{area(s^*_{min})}{area(s)} = 16 \ .$$

Combining these two observations, we obtain

$$\frac{area(s^*_{min})}{area(c \cap s^*_{min})} \leq \frac{area(s^*_{min})}{area(c \cap s)} = 16\frac{area(s)}{area(c \cap s)} \leq 16\beta = O(\alpha^2) \ .$$

□

Lemma 2.2 allows us to use a simple construction, due to Fredman, that produces $l = O(\gamma \log \gamma) = O(\alpha^2 \log \alpha)$ points within s^*_{min} so that each object of $\mathcal{C}_v$ contains at least one of these points (the set produced by Fredman's construction 'stabs' any convex object that covers some fraction of a given convex region). We thus partition $\mathcal{C}_v$ into l subsets, $\mathcal{C}^1_v, \ldots, \mathcal{C}^l_v$, each having the common point property. Consider one such subset, $\mathcal{C}^i_v$, and put $n^i_v = |\mathcal{C}^i_v|$. Since this subset has the common point property, we may sort the objects in $\mathcal{K}^i_v$, where $\mathcal{K}^i_v$ is the set of original 3-D objects corresponding to $\mathcal{C}^i_v$, by their vertical depth order, in $O(n^i_v \log n^i_v)$ time. Moreover, by Lemma 2.1, we can compute the union of any subset of m objects in $\mathcal{C}^i_v$ in $O(\lambda_s(m) \log m)$ time, where s is the maximum number of intersections between

the boundaries of any pair of objects in $\mathcal{C}$. We thus compute, for the set $\mathcal{K}_v^i$, a tree τ_v^i, as described in the introduction for the case of horizontal disks. This requires $O(\lambda_s(n_v^i)\log n_v^i)$ storage and $O(\lambda_s(n_v^i)\log^2 n_v^i)$ preprocessing time. The overall storage required for processing the node v is thus $O(\lambda_s(|\mathcal{S}_v|)\log|\mathcal{S}_v|)$, and the overall preprocessing time of node v is $O(\lambda_s(|\mathcal{S}_v|)\log^2|\mathcal{S}_v|)$. Since the sum of the sizes of all canonical subsets of T is $O(n\log^2 n)$, the overall storage required for our structure is $O(\lambda_s(n)\log^3 n)$ and the total preprocessing time is $O(\lambda_s(n)\log^4 n)$.

We next describe how a query is answered using the above structure. Let p be a query point in 3-space. Our goal is to locate the object of $\mathcal{K}$ lying immediately below p, if such an object exists. We first perform a query with the xy-projection of p in the two-dimensional segment tree T that was constructed in the first stage. The cost of this query is $O(\log^2 n)$ and its output consists of $O(\log^2 n)$ canonical subsets of $\mathcal{S}$, each associated with a unique node of T.

For each of these nodes v we will find the object of $\mathcal{K}_v$ lying immediately below p (if such an object exists). By computing the heights in which the vertical ray emanating from p and directed downwards intersects these objects, and by selecting the object with the highest such height, we can determine the output to the original query. Recall that at the node v we have $l = O(\alpha^2\log\alpha)$ associated trees, for the sets $\mathcal{K}_v^1,\ldots,\mathcal{K}_v^l$. We query each of these trees separately, obtaining a candidate object from each tree. We then select from these candidates, as above, the one with the highest intersection with the downward-directed ray from p.

It remains to describe the query procedure applied to the tree τ_v^i of a set $\mathcal{K}_v^i$. The simple procedure that is used in the case of horizontal disks, described in the introduction, does not apply in our case, since we can not compare p with the objects of $\mathcal{K}_v^i$ based on their heights over the common point of $\mathcal{C}_v^i$. If p did lie above or below each object in $\mathcal{K}_v^i$, then we could use an appropriate binary search procedure for this task, but, in general, p need not have this property. Instead, we apply here the procedure that was described in [2] for 'merging' the objects in a set $\mathcal{R}$ with the objects in a set $\mathcal{B}$, whose corresponding set of xy-projections has the common point property. In our case, the set $\mathcal{R}$ consists of a single object, namely the point p, the set $\mathcal{B}$ is $\mathcal{K}_v^i$, and we wish to find the object K' of $\mathcal{K}_v^i$ lying immediately below p and the object K'' of $\mathcal{K}_v^i$ lying immediately above p (if they exist). For the reader's convenience, we now describe this merging procedure in the context of our somewhat simpler setting.

The search in τ_v^i proceeds by levels, beginning at the root level. At each level at most four nodes are visited, so that K' and K'' are stored at the leaves of (one or two) of their at most four corresponding subtrees. At the leaves level, K' and K''

are selected from the (at most) four objects of $\mathcal{K}_v^i$ that are stored at the (at most) four visited leaves. At each internal node that is visited, we need to determine if the xy-projection of p intersects the union (of a subset of $\mathcal{C}_v^i$) that is associated with that node, and, if so, to find a witness object containing the xy-projection of p. This can be done in $O(\log n)$ time (see Lemma 2.1).

Let U_{root} denote the region associated with the root of τ_v^i. If the xy-projection of p does not lie in U_{root}, then p does not lie above or below any object of $\mathcal{K}_v^i$. If, however, the xy-projection of p does lie in U_{root}, then we must determine the nodes of the next level that should be visited. Let K be the object of $\mathcal{K}_v^i$ corresponding to the witness that was found at the root, and assume that K lies above p. If K is stored in a leaf of the subtree of the node $left(root)$, then we only have to visit the node $left(root)$ in the level of the children of the root (assuming the lowest object in $\mathcal{K}_v^i$ is stored at the leftmost leaf of τ_v^i). This is because, in this case, both K' and K'' are necessarily stored at the leaves of the subtree of $left(root)$, since they are stabbed by the vertical line through the common point of $\mathcal{C}_v^i$ at points than are lower than the stabbing point of K (where the stabbing point of K' is lower than that of K''). This follows from our assumptions about the objects in $\mathcal{K}$, i.e., they are non-intersecting and convex. Otherwise, if K is stored in the subtree of a leaf of $right(root)$, both the children of the root must be visited. The case where K lies below p is treated symmetrically. The invariant maintained by the procedure (at most four visited nodes per level) clearly holds at the level of the children of the root. At a general level j, we first visit the at most four nodes identified at this level, and select at most two of them (the rightmost of these nodes for which a below witness of p was found, and the leftmost of these nodes for which an above witness was found), whose children we still may need to visit. We determine these at most four children, based on a criterion similar to that used at the root. The validity of the invariant property follows easily by induction on j. Thus we spend $O(\log^2 n)$ time in finding the object of $\mathcal{K}_v^i$ lying immediately below p. Since we repeat this procedure $O(\alpha^2 \log \alpha)$ times, the time spent at node v is $O((\alpha^2 \log \alpha) \log^2 n)$ and the overall cost of the query is thus $O((\alpha^2 \log \alpha) \log^4 n)$.

The following theorem summarizes the result of this section.

Theorem 2.3 *Let $\mathcal{K}$ be a set of n non-intersecting convex simply-shaped flat objects in 3-space, and assume that the xy-projections of the objects in $\mathcal{K}$ are all α-fat. It is possible to construct a data structure of size $O(\lambda_s(n) \log^3 n)$ in $O(\lambda_s(n) \log^4 n)$ time, where s is the maximum number of intersections between the boundaries of the xy-projections of any pair of objects in $\mathcal{K}$, so that, for a given query point p, the object of $\mathcal{K}$ lying immediately below p (if such an object exists) can be found in*

$O((\alpha^2 \log \alpha) \log^4 n)$ *time. If the objects in* $\mathcal{K}$ *are fat triangles or disks or belong to any other class of objects for which the factor* $\lambda_s(n)$ *in Lemma 2.1 can be replaced by* n*, then the above bounds on the storage and preprocessing time can be replaced by* $O(n \log^3 n)$ *and* $O(n \log^4 n)$*, respectively.*

The following observation is needed for Section 5. Assume we have constructed the data structure above for the set $\mathcal{K}$, but now some of the objects in $\mathcal{K}$ may 'disappear' (one by one), so that whenever an object $K \in \mathcal{K}$ disappears it is the highest object in all the $O(\log^2 n)$ subsets $\mathcal{K}_v^i$ in which it is stored. We would like to maintain our data structure during this process. This is easily done by storing with each tree τ_v^i of the data structure a pointer to the leaf storing the current highest object in $\mathcal{K}_v^i$. Initially, this pointer points to the extremal leaf of τ_v^i storing the highest object in $\mathcal{K}_v^i$. When an object $K \in \mathcal{K}$ is deleted, we update the pointers that are associated with the $O(\log^2 n)$ trees τ_v^i in which K is stored. This takes $O(\log^2 n)$ time. Now, when searching with a query point p in some tree τ_v^i whose pointer is not pointing to one of its extremal leaves, we would like to consider only the active objects of $\mathcal{K}_v^i$, that is, only the objects that lie below the current highest object (including the current highest object). By traversing the path in τ_v^i from the leaf storing the previous highest object to the root, we obtain the set of active objects as a collection of $O(\log n_v^i)$ disjoint subtrees. We search with p, as before, in these subtrees beginning with the subtree with the highest height range, until one of these searches yields a candidate object. This candidate object is taken as the candidate object of τ_v^i. Notice that in the worst case this may increase the query cost by a logarithmic factor. Thus, the cost of a query during this process is $O((\alpha^2 \log \alpha) \log^5 n)$.

3 Point Enclosure

In this section we present efficient solutions to the standard and generalized point enclosure problems. Let $\mathcal{C}$ be a set of n convex simply-shaped α-fat objects in the plane. In the standard version, the goal is to preprocess the set $\mathcal{C}$ so that, for a given query point p, all k objects of $\mathcal{C}$ containing p can be reported efficiently. Our solution to this problem is very similar to the solution presented in the previous section for vertical ray shooting. The data structure that we construct here differs from the one constructed above only in the final step of the construction, where we do not have to store the objects of $\mathcal{C}_v^i$ at the leaves of its corresponding tree in any particular order.

The query proceeds also in a very similar manner. The difference again is in the procedure applied to the tree τ_v^i of a set $\mathcal{C}_v^i$. This procedure now has to report all k_v^i objects of $\mathcal{C}_v^i$ containing the query point p. Based on Lemma 2.1, this can be done in $O(\log n + k_v^i \log^2 n)$ time: Simply traverse τ_v^i in a top-down fashion, visiting all nodes whose corresponding subtrees store an object that contains p, and spending $O(\log n)$ time at each such node. The above bound on the query time follows by observing that the number of visited nodes is $O(k_v^i \log n)$. By repeating this to all the relevant sets $\mathcal{C}_v^i$, we obtain the following theorem that summarizes our result on the point enclosure problem.

Theorem 3.1 *Let $\mathcal{C}$ be a set of n convex simply-shaped α-fat objects in the plane. It is possible to construct a data structure of size $O(\lambda_s(n) \log^3 n)$ in $O(\lambda_s(n) \log^4 n)$ time, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$, so that, for a given query point p, the k objects of $\mathcal{C}$ containing p can be reported in $O((\alpha^2 \log \alpha) \log^3 n + k \log^2 n)$ time. If the objects in $\mathcal{C}$ belong to a class of objects for which the factor $\lambda_s(n)$ in Lemma 2.1 can be replaced by n, then the above bounds on the storage and preprocessing time can be replaced by $O(n \log^3 n)$ and $O(n \log^4 n)$, respectively.*

In the generalized version of the point enclosure problem, we assume that the objects of $\mathcal{C}$ are colored. The goal here is to preprocess $\mathcal{C}$ so that, for a given query point p, all k distinct colors u for which there exists an object of $\mathcal{C}$ of color u containing p can be reported efficiently. Again the only difference between our construction for this problem and the construction for the vertical ray shooting problem is in the final step. Consider a set $\mathcal{C}_v^i$, and let m_v^i be the number of distinct colors the objects of $\mathcal{C}_v^i$ have. We build a balanced binary tree for $\mathcal{C}_v^i$. The tree has m_v^i leaves, and at each leaf we store the union of all objects of some color. At an internal node ζ we store the union of all objects whose colors are associated with the leaves of the subtree rooted at ζ. The procedure that is applied to this tree when processing a query p is similar to the one used above for the standard point enclosure problem, and outputs all k_v^i colors u for which there exists an object of $\mathcal{C}_v^i$ of color u containing p. The cost of the application of this procedure to a single set $\mathcal{C}_v^i$ is, as above, $O(\log n + k_v^i \log^2 n)$. By repeating this to all the relevant sets $\mathcal{C}_v^i$, and by noticing that, unlike in the standard version, a member of the output set (i.e., a color) may be reported more than once (actually $O((\alpha^2 \log \alpha) \log^2 n)$ times), we obtain the following theorem that summarizes our result for the generalized point enclosure problem (note, in view of the above comment, the difference between the query times in Theorem 3.1 and Theorem 3.2).

Theorem 3.2 *Let $\mathcal{C}$ be a set of n convex simply-shaped α-fat objects in the plane, and assume that each object of $\mathcal{C}$ is assigned some color. It is possible to construct a data structure of size $O(\lambda_s(n)\log^3 n)$ in $O(\lambda_s(n)\log^4 n)$ time, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$, so that, for a given query point p, the k colors u, for which there exists an object of $\mathcal{C}$ of color u containing p, can be reported in $O(\alpha^2\log\alpha(\log^3 n + k\log^4 n))$ time. If the objects in $\mathcal{C}$ belong to a class of objects for which the factor $\lambda_s(n)$ in Lemma 2.1 can be replaced by n, then the above bounds on the storage and preprocessing time can be replaced by $O(n\log^3 n)$ and $O(n\log^4 n)$, respectively.*

4 Bounded-Size Range Searching and Arc Shooting

Let $\mathcal{C}$ be a set of n convex simply-shaped α-fat objects in the plane, and let ρ be the diameter of the object in $\mathcal{C}$ with smallest diameter. In the bounded-size range searching problem, we wish to preprocess $\mathcal{C}$, so that, for a given query range R with diameter at most $h\rho$, for some constant h, all k objects of $\mathcal{C}$ intersecting R can be reported efficiently. Our solution to this problem is based on a result of Overmars and van der Stappen [26]. It is efficient whenever the following test can be performed efficiently (after some preprocessing of $\mathcal{C}$). Given a star-shaped region U, which is the union of the objects in some subset of $\mathcal{C}$ with the common point property, determine whether U intersects a query range R. We are able to perform this test efficiently, using known results, only when the set $\mathcal{C}$ consists of either polygons (i.e., convex α-fat polygons, each with a constant number of edges) or disks, and the query range is either a polygon with a constant number of edges that is not necessarily convex and α-fat or a disk.

Overmars and van der Stappen [26] show that given a bounded-size query range R, it is possible to construct a small set of points Φ, such that the set of all objects in $\mathcal{C}$ that contain at least one of the points of Φ is a superset of the output set of the query. The size of Φ depends only on α and on h; more precisely it is $O(\alpha^4 h^2)$, and Φ can be constructed in $O(|\Phi|)$ time. They use this set to answer the range searching query in the following way. They perform $|\Phi|$ point location queries, and then determine, for each of the candidate objects that were found, whether it really intersects the range R. In their setting this approach is possible, since they assume that the underlying objects are disjoint, and hence the number of candidates is at most $|\Phi|$. However, since we do not make this assumption, we can not afford to simply perform $|\Phi|$ point enclosure queries, since we may collect as many as $\Theta(n)$ candidate objects, while the final output set may be of only constant size.

For our solution we employ the data structure of the previous section. In addition, we preprocess each region associated with a node w of a tree τ_v^i for efficient ray shooting queries, using the methods of [3, 6, 7, 16].

We next describe the query procedure. Given a query range R, we first compute the appropriate point set Φ. Then, for each point $p \in \Phi$ we perform the following procedure. We first identify, as in the preceding sections, the $O(\log^2 n)$ relevant nodes of the segment tree T. Consider such a node v. We now replace p by the original query range R, and, for each of the trees τ_v^i associated with v, we report all k_v^i objects in $\mathcal{C}_v^i$ that intersect R. It remains to describe how we determine for a node w of τ_v^i whether the star-shaped region U_w that is associated with w intersects R. First check, in $O(\log n)$ time, whether one of the vertices of R lies in U_w, by performing a constant number of point location queries (if R is a disk, we take the leftmost and rightmost points of its boundary). If none of the vertices of R lies in U_w, then pick any point in U_w, and check, in constant time, whether it lies in R. If the answer is negative again, then U_w intersects R if and only if an edge of R intersects the boundary of U_w. Thus perform for each of the edges e of R a shooting query using the methods of [3, 6, 7, 16], to determine whether e intersects the boundary of U_w. If R is a polygon, shoot with a ray emanating from one of the vertices of e and containing e, if R is a disk, simply shoot with e (which is a half circle).

Assuming the underlying objects of $\mathcal{C}$ are polygons, preprocessing the region U_w for ray shooting queries takes $O(|w|)$ time [6, 7, 16], and for circular arc shooting queries $O(|w| \log^3 |w|)$ [3]. A ray shooting query can be answered in $O(\log |w|)$ time, and a circular arc shooting query in $O(\log^4 |w|)$ time. If the underlying objects of $\mathcal{C}$ are disks, then U_w can be preprocessed for both ray shooting queries and circular arc shooting queries in $O(|w| \log^3 |w|)$ time [1, 3], and a shooting query can be answered in $O(\log^4 |w|)$ time.

The following theorem summarizes our results (see also Table 1).

Theorem 4.1 *Let $\mathcal{C}$ be a set of n convex α-fat polygons in the plane, with minimum diameter greater than ρ.*

(i) It is possible to construct a data structure of size $O(\lambda_s(n) \log^3 n)$ in $O(\lambda_s(n) \log^4 n)$ time, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$, so that, for a given query polygon R (that is not necessarily convex and α-fat) with diameter less than $h\rho$, for some constant h, the k objects of $\mathcal{C}$ intersecting R can be reported in $O((\alpha^6 \log \alpha) \log^3 n + (\alpha^4) k \log^2 n)$ time.

(ii) It is possible to construct a data structure of size $O(\lambda_s(n)\log^6 n)$ in $O(\lambda_s(n)\log^6 n)$ time, so that, for a given query disk R of diameter less than $h\rho$, for some constant h, the k objects of $\mathcal{C}$ intersecting R can be reported in $O((\alpha^6\log\alpha)\log^6 n + (\alpha^4)k\log^5 n)$ time.

(iii) If the polygons in $\mathcal{C}$ are triangles or belong to any other class of polygons for which the factor $\lambda_s(n)$ in Lemma 2.1 can be replaced by n, then the factor $\lambda_s(n)$ can be replaced by n in the above bounds on the storage and preprocessing time.

Let $\mathcal{C}$ be a set of n disks in the plane, with minimum diameter greater than ρ. It is possible to construct a data structure of size $O(n\log^6 n)$ in $O(n\log^6 n)$ time, so that, for a given query range R with diameter less than $h\rho$, which is either a polygon (that is not necessarily convex and α-fat) or a disk, the k objects of $\mathcal{C}$ intersecting R can be reported in $O((\alpha^6\log\alpha)\log^6 n + (\alpha^4)k\log^5 n)$ time.

In the bounded-size arc shooting problem, we wish to preprocess $\mathcal{C}$, so that, for a given oriented query arc r of length at most $h\rho$, for some constant h, the first object of $\mathcal{C}$ hit by r (if such an object exists) can be found efficiently. Our solution to this problem follows immediately from the solution above to the bounded-size range searching problem. We use the same data structure, however, here, we do not need to construct the trees τ_v^i beyond their roots, so we can save a logarithmic factor in the storage and preprocessing.

Given an oriented query arc r of length less than $h\rho$, for some constant h, we first compute the set Φ, as above, for some axis-parallel square with, say, edge length $2h\rho$ that contains r. Then, for each of the points $p \in \Phi$ we perform the following procedure. We first identify, as in the preceding sections, the $O(\log^2 n)$ relevant nodes of the segment tree T. Then, for each of these nodes v, we perform a shooting query with r in each of the degenerate trees τ_v^i that are associated with v. The final answer is simply the object of $\mathcal{C}$ that is hit by r at a point that is the nearest to the designated endpoint of r among all the hitting points that were found throughout the entire process.

The following theorem summarizes our results (see also Table 2).

Theorem 4.2 *Let $\mathcal{C}$ be a set of n convex α-fat polygons in the plane, with minimum diameter greater than ρ.*

(i) It is possible to construct a data structure of size $O(\lambda_s(n)\log^2 n)$ in $O(\lambda_s(n)\log^3 n)$ time, where s is the maximum number of intersections between the boundaries of any pair of objects in $\mathcal{C}$, so that, for a given oriented query line segment r

of length less than $h\rho$, for some constant h, the first object of $\mathcal{C}$ hit by r (if such an object exists) can be found in $O((\alpha^6 \log \alpha) \log^3 n)$ time.

(ii) It is possible to construct a data structure of size $O(\lambda_s(n) \log^5 n)$ in $O(\lambda_s(n) \log^5 n)$ time, so that, for a given oriented query circular arc r of length less than $h\rho$, for some constant h, the first object of $\mathcal{C}$ hit by r (if such an object exists) can be found in $O((\alpha^6 \log \alpha) \log^6 n)$ time.

(iii) If the polygons in $\mathcal{C}$ are triangles or belong to any other class of polygons for which the factor $\lambda_s(n)$ in Lemma 2.1 can be replaced by n, then the factor $\lambda_s(n)$ can be replaced by n in the above bounds on the storage and preprocessing time.

Let $\mathcal{C}$ be a set of n disks in the plane, with minimum diameter greater than ρ. It is possible to construct a data structure of size $O(n \log^5 n)$ in $O(n \log^5 n)$ time, so that, for a given oriented query arc r of length less than $h\rho$, which is either a segment or a circular arc, the first object of $\mathcal{C}$ hit by r (if such an object exists) can be found in $O((\alpha^6 \log \alpha) \log^6 n)$ time.

5 Computing a Depth Order

Let $\mathcal{K}$ be a set of n non-intersecting convex simply-shaped flat objects in 3-space, and assume that the xy-projections of the objects in $\mathcal{K}$ are α-fat. Let β be an upper bound for the value

$$\max \left\{ \frac{area(c_i)}{area(c_i \cap c_j)} \;\mid\; K_i, K_j \in \mathcal{K},\ c_i \cap c_j \neq \emptyset \right\} ,$$

where c_i and c_j are the xy-projections of K_i and K_j. We would like to determine whether the 'above-below' relation over $\mathcal{K}$ contains cycles, and, if it does not, to compute a depth order for $\mathcal{K}$ (i.e., a linear order of $\mathcal{K}$ such that if $K_1, K_2 \in \mathcal{K}$ and K_1 lies vertically above K_2 then K_1 precedes K_2).

Our algorithm is based on the solution of Section 2 to the vertical ray shooting problem, and on the observation of the last paragraph of that section that allows us to efficiently maintain the underlying data structure while objects obeying a certain condition are deleted from it. The algorithm tries to compute a depth order for $\mathcal{K}$ by successively finding an object of $\mathcal{K}$ that is completely visible (from $z = \infty$) and removing it from $\mathcal{K}$. If, at some point, there is no such object (and $\mathcal{K} \neq \emptyset$), then the above-below relation over $\mathcal{K}$ contains cycles. Otherwise, the order in which the objects were removed is a depth order for $\mathcal{K}$.

Let D be the data structure of Section 2 for downwards-directed vertical ray shooting amidst $\mathcal{K}$. Notice that whenever an object is removed from $\mathcal{K}$ it satisfies the condition that is mentioned in the last paragraph of Section 2 concerning its position in D (since it is completely visible). Therefore, according to the observation of that paragraph, we can efficiently maintain D during this process. More precisely, a currently completely visible object can be deleted from D in $O(\log^2 n)$ time, and a vertical ray shooting query can be answered in $O((\alpha^2 \log \alpha) \log^5 n)$ time.

We now describe the procedure that determines in $O((\beta \log \beta)(\alpha^2 \log \alpha) \log^5 n)$ time whether an object $K \in \mathcal{K}$ is currently completely visible. Since the area of the intersection between the xy-projection c of K and the xy-projection of any other object in $\mathcal{K}$ is either zero or at least $1/\beta$ of the area of c, we can compute, as above, a set P_K of $O(\beta \log \beta)$ points within c, so that each of the xy-projections (of objects in $\mathcal{K}$) that intersects c contains at least one of these points. We now lift the points in P_K to the plane $z = \infty$, and perform with each of the lifted points a vertical ray shooting query using D. Clearly K is completely visible if and only if all the answers that are obtained are K itself.

In the initial stage of the algorithm, we compute, using the procedure above, the set $\mathcal{X}$ of all objects in $\mathcal{K}$ that are completely visible. The complexity of this stage is $O((\beta \log \beta)(\alpha^2 \log \alpha) n \log^4 n)$ time (since in this stage we can still use the static version of D which has query time $O((\alpha^2 \log \alpha) \log^4 n)$).

Now assume that an object K that is currently completely visible is about to be removed. We wish to find all the objects that are revealed by this operation, i.e., all the objects that become completely visible after K is removed. To this end, we lift each of the points in P_K to the (not necessarily horizontal) plane containing K, and perform a ray shooting query with each of the lifted points. Clearly, if K' is an object that becomes completely visible after the removal of K, then K' was obtained as the answer to at least one of these queries. Thus, in this way we obtain a set $\mathcal{Y}_K$ of $O(\beta \log \beta)$ candidates that we now need to test (after removing K), as above, to determine which of them are completely visible.

The main stage of the algorithm consists of the following loop.

while $\mathcal{X} \neq \emptyset$ do
1. pick any object $K \in \mathcal{X}$
2. find the appropriate set of candidates $\mathcal{Y}_K$
3. delete K from D (and from $\mathcal{K}$)
4. check for each of the objects in $\mathcal{Y}_K$, whether it is currently completely visible; if it is, add it to $\mathcal{X}$

If at the end of this stage $\mathcal{K}$ is empty, then we have found a depth order for $\mathcal{K}$, otherwise, the above-below relation over $\mathcal{K}$ contains cycles. The complexity of a single iteration in the above loop is $O((\beta^2 \log^2 \beta)(\alpha^2 \log \alpha) \log^5 n)$ time. Thus, the complexity of the entire algorithm is $O((\beta^2 \log^2 \beta)(\alpha^2 \log \alpha) n \log^5 n)$.

Acknowledgement

I would like to thank Pankaj Agarwal and Micha Sharir for helpful discussions on the contents of this paper.

References

[1] P.K. Agarwal, personal communication.

[2] P.K. Agarwal, M.J. Katz and M. Sharir, Computing depth orders and related problems, *Proc. 4th Scandinavian Workshop on Algorithm Theory*, 1994, 1–12.

[3] P.K. Agarwal and M. Sharir, Circle shooting in a simple polygon, *J. Algorithms* 14 (1993), 68–87.

[4] 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.

[5] F. Aurenhammer, Improved algorithms for discs and balls using power diagrams, *J. Algorithms* 9 (1988), 151–161.

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

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

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

[9] 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.

[10] M. de Berg, M. van Kreveld and J. Snoeyink, Two- and three-dimensional point location in rectangular subdivisions, *J. Algorithms* 18 (1995), 256–277.

[11] A. Efrat, G. Rote and M. Sharir, On the union of fat wedges and separating a collection of segments by a line, *Comp. Geom. Theory and Appls* 3 (1993), 277-288.

[12] P. Gupta, R. Janardan and M. Smid, Further results on generalized intersection searching problems: counting, reporting and dynamization, *Proc. 3rd Workshop on Algorithms and Data Structures*, 1993, 361–372.

[13] P. Gupta, R. Janardan and M. Smid, Generalized halfspace range searching, segment intersection searching, and fat-triangle range searching and stabbing, manuscript, 1994. (See also: Efficient algorithms for generalized intersection searching on non-iso-oriented objects, *Proc. 10th ACM Symp. on Computational Geometry*, 1994, 369–378.)

[14] P. Gupta, R. Janardan and M. Smid, On intersection searching problems involving curved objects, *Proc. 4th Scandinavian Workshop on Algorithm Theory*, 1994, 183–194.

[15] S. Hart and M. Sharir, Nonlinearity of Davenport–Schinzel sequences and of generalized path compression schemes, *Combinatorica* 6 (2) (1986), 151–177.

[16] J. Hershberger and S. Suri, A pedestrian approach to ray shooting: Shoot a ray, take a walk, *Proc. 4th ACM-SIAM Symp. Discrete Algorithms*, 1993, 54–63.

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

[18] M.J. Katz, M.H. Overmars, M. Sharir, Efficient hidden surface removal for objects with small union size, *Comp. Geom. Theory and Appls* 2 (1992), 223–234.

[19] K. Kedem, R. Livne, J. Pach and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discrete Comput. Geom.* 1 (1986), 59–71.

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

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

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

[23] K. Mehlhorn, *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*, Springer-Verlag, Berlin, 1984.

[24] N. Miller and M. Sharir, Efficient randomized algorithms for constructing the union of fat triangles and of pseudodiscs, manuscript, 1991.

[25] M.H. Overmars, Point location in fat subdivisions, *Inf. Proc. Letters* 44 (1992), 261–265.

[26] 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.

[27] F.P. Preparata and M.I. Shamos, *Computational Geometry – an Introduction*, Springer-Verlag, New York, 1985.

[28] M. Sharir, On k-sets in arrangements of curves and surfaces, *Discrete Comput. Geom.* 6 (1991), 593–613.

[29] M. Sharir and P.K. Agarwal, *Davenport-Schinzel Sequences and Their Geometric Applications*, Cambridge University Press, to appear.

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

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

[32] F. van der Stappen and M. Overmars, Motion planning amidst fat obstacles, *Proc. 10th ACM Symp. on Computational Geometry*, 1994, 31–40.

[33] M. van Kreveld, *New Results on Data Structures*, Ph.D. thesis, Utrecht University, 1992.

[34] M. van Kreveld, On fat partitioning, fat covering, and the union size of polygons, *Proc. 3rd Workshop on Algorithms and Data Structures*, 1993, 452–463.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399