Intersection queries in sets of disks

Marc van Kreveld, Mark Overmars and Pankaj Agarwal

RUU-CS-90-13
April 1990

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Intersection queries in sets of disks

Marc van Kreveld, Mark Overmars and Pankaj Agarwal

Technical Report RUU-CS-90-13
April 1990

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
the Netherlands

Intersection Queries in Sets of Disks*

Mark van Kreveld[1] Jan Vahrenhold[2] Rolf Klein[1]

[1] Dept. of Computer Science, Utrecht Univ.,
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands.
[2] DIMACS Center, Rutgers University,
P.O. Box 1179, Piscataway, NJ 08855-1179, USA
and
Computer Science Department, Duke University,
Durham, NC 27706, USA

Abstract

In this paper we develop some new data structures for storing sets of disks such that different types of queries can be answered efficiently. In particular we study intersection queries with lines and line segments and stabbing queries. In the case of non-intersecting disks we obtain structures that use linear storage and have a query time of $O(n^{\beta+\epsilon}+k)$ for intersection queries, where n is the number of disks, $\beta = \log_2(1+\sqrt{5})-1 \approx 0.695$, ϵ is an arbitrary small positive constant, and k is the number of disks reported. For sets of intersecting disks we obtain a structure that uses $O(n \log n)$ storage and answers a query in time $O(n^{0.695} \log^2 n)$. For ray shooting we obtain a structure that uses linear storage and has $O(n^{0.695})$ query time for arbitrary ϵ.

1 Introduction

Data structures for storing geometric objects, allowing for a variety of different types of queries, play an important role in computational geometry. Many different data structures have been designed for different types of objects and queries. An important type of query is the intersection query. Here we ask which objects in the data structure intersect a given query object. This problem has particularly been studied for the case in which the structure stores a set of points and the query object is an axis-parallel rectangle, the so-called range searching problem. Other

*The research of the first two authors was supported by the ESPRIT II Basic Research Action Program of the EC under contract No. 3075 (project ALCOM). The work of the third author was supported by DIMACS (Center for Discrete Mathematics and Theoretical Computer Science, a National Science Foundation Science and Technology Center - NSF STC 88-09648).

ISSN:0924-3275

data structures exist that store axis-parallel objects like horizontal line segments or rectangles.

Only recently structures have been designed that store sets of arbitrarily oriented line segments and allow for intersection queries with arbitrary line segments (see [1, 6, 10, 13]).

Almost no work has been done on intersection searching in curved objects. The only result we are aware of is a forthcoming paper by Sharir for searching with a point in a set of disks [14]. In this paper we will give some first data structures for storing sets of disks such that intersection queries with lines and line segments can be performed efficiently.

The results in this paper use as underlying data structure the so-called conjugation tree designed by Edelsbrunner and Welzl [9]. This structure is based on an elegant way to partition the plane recursively. (The structure is also called a ham-sandwich tree in [7].) We augment this structure by associating data structures to internal nodes that store subsets of the disks. These associated structures are based on hierarchical convex hulls of disks.

The rest of this paper is organized as follows.

In Section 2 we briefly recall the main properties of the conjugation tree we use. Moreover we indicate the basic method of storing disks in the conjugation tree.

In Section 3 we restrict ourselves to sets of non-intersecting disks. First we describe how intersection queries with a line can be performed in time $O(n^\beta + k)$ using only linear storage, where $\beta = \log_2(1+\sqrt{5})-1 < 0.695$ and k is the number of reported answers. Next, we extend this result to queries with line segments instead of lines. The query time becomes $O(n^{\beta+\epsilon} + k)$ for any $\epsilon > 0$, while still using only $O(n)$ storage.

In Section 4 we handle intersection queries in arbitrary (i.e., intersecting) disks. In this case we use a quite different approach. We transform the problem to a three-dimensional problem, mapping the disks to points in $\mathbb{R}^3$. The structure we obtain uses $O(n \log n)$ storage and has a query time of $O(n^{2/3} \log^2 n + k)$.

In Section 5, we look at the related problem of ray shooting. Here we are given a query ray emanating from a point q in direction d, and we ask for the first disk we hit as we move from q along the ray. We give a data structure for this problem that uses linear storage (for arbitrary disks) and has a query time of $O(n^{\beta+\epsilon})$, for any $\epsilon > 0$.

Finally, in Section 6, we conclude with some extensions and open problems. In particular, we briefly indicate how the query time of the algorithms, described in Section 3 and Section 5, can be improved to $O(n^{2/3+\epsilon})$, for any $\epsilon > 0$, without increasing the space requirement.

2 The conjugation tree

The basic data structure for most solutions presented in this paper is the conjugation tree (also called the ham-sandwich tree), introduced by Edelsbrunner and Welzl in [9] (see also [7]). The tree was originally used for storing a set of points in the plane, and for solving half planar range queries: how many (or which) points lie in a query halfplane. The structure can also be used to determine how many (or which) points lie in a triangle, or on a query line. The conjugation tree is a binary tree in which each node corresponds to a convex region of the plane. The root corresponds to the whole plane. The region corresponding to a node δ is divided by a straight line l_δ, called *bisecter,* into two subregions, each containing about the same number of points. These subregions correspond to the sons of δ. The important property of conjugation trees is that any line intersects with the region of at most $O(n^\beta)$ nodes, where $\beta = \log_2(1+\sqrt{5})-1 < 0.695$. As a result one can efficiently determine regions that are completely contained inside the query halfplane or lie completely outside the halfplane. In this way only $O(n^\beta)$ nodes have to be visited when performing a query. A conjugation tree can be constructed in $O(n \log n)$ time. (For details see [9].)

In this paper, we will use conjugation trees to store disks rather than points. Let $\mathcal{D} = \{D_1, \ldots, D_n\}$ be a set of disks, and let $\mathcal{P} = \{p_1, \ldots, p_n\}$ be the set of their centers. We construct a conjugation tree T on $\mathcal{P}$. For each disk D, we search with its center p in the tree T and locate the highest node δ on the path, such that the bisecter l_δ intersects D. We store D in some type of associated structure S_δ at this node. This associated structure S_δ depends on the type of query we want to answer. Each disk is stored in the structure S_δ for exactly one node δ in the tree.

Storing the disks in the way described above has an important property: Whenever a line l intersects a disk D stored in S_δ, then l intersects the region associated with δ. This easily follows from the fact that D is stored at the highest possible node. As a result D will lie completely inside the region associated with δ. Hence, l must intersect the region if D intersects l. Thus, all disks intersecting a line l can be determined by querying the S_δ structures at only $O(n^\beta)$ nodes. Let $R(n)$ denote the query time of the associated structure storing n disks. It easily follows from [9] that the total query time of the structure is given by the recurrence $Q(n) \leq Q(n/2) + Q(n/4) + R(n)$. Also, if the storage requirement of an associated structure is $M(n)$, then the whole structure needs $O(n) + O(M(n))$ storage.

The following lemma will be useful in estimating time bounds for the data structures we obtain:

Lemma 1 *Let T be a data structure storing n objects, having a query time of $Q(n)$. If $Q(n)$ satisfies $Q(n) \leq Q(n/2) + Q(n/4) + c \cdot n^\eta$ for constants $c > 0$ and $\eta < \beta$, then $Q(n) = O(n^\beta)$ (where $\beta = \log_2(1+\sqrt{5}) - 1$).*

Proof. Claim: $Q(n) \leq c'(n^\beta - n^\mu)$, where $\eta < \mu < \beta = \log_2(1+\sqrt{5}) - 1$, and c' is

some fixed constant.

$$\begin{aligned}
Q(n) &\leq Q(n/2) + Q(n/4) + c \cdot n^{\eta} \\
&\leq c'((n/2)^{\beta} - (n/2)^{\mu}) + c'((n/4)^{\beta} - (n/4)^{\mu}) + c \cdot n^{\eta} \\
&\leq c' \cdot n^{\beta}((1/2)^{\beta} + (1/4)^{\beta}) - c' \cdot n^{\mu}((1/2)^{\mu} + (1/4)^{\mu}) + c \cdot n^{\eta} \\
&\leq c' \cdot n^{\beta} - c' \cdot n^{\mu} - c' \cdot \zeta \cdot n^{\mu} + c \cdot n^{\eta} \quad \text{(for a positive constant } \zeta) \\
&\leq c'(n^{\beta} - n^{\mu}) \quad \text{(for } c' > c/\zeta).
\end{aligned}$$

□

3 Intersection queries in disjoint disks

We will first consider intersection queries with a line in a set of disjoint disks. We are given a set $\mathcal{D} = \{D_1, \ldots, D_n\}$ of n non-intersecting disks of arbitrary size. We wish to preprocess $\mathcal{D}$ for the following type of query: Given a query line l, report all disks in $\mathcal{D}$ that intersect l.

To solve the problem, we use the conjugation tree for disks, as described in the previous section. So we only have to describe the associated structure S_δ. Note that all disks stored in S_δ intersect the bisecter l_δ. Let $p = l_\delta \cap l$ be the intersection point of the query line and l_δ. We first describe a solution to a special case in which all disks intersect l_δ on the same side of p, and then extend it to the general case.

Suppose a set $\mathcal{D}'$ of disks $D_1, \ldots, D_m$ is given, all of which intersect the fixed line l_δ (which we assume to be horizontal), and we perform an intersection query with a line l which intersects l_δ in a point p, left of all intersections of disks with l_δ (see Figure 1). Let l_δ^+ (resp. l_δ^-) denote the halfplane lying above (resp. below) l_δ. Define the sets $\mathcal{D}'^+ = \{D_i^+ \mid D_i^+ = D_i \cap l_\delta^+, 1 \leq i \leq m\}$ and $\mathcal{D}'^- = \{D_i^- \mid D_i^- = D_i \cap l_\delta^-, 1 \leq i \leq m\}$. $D_i^+ \neq D_i$ (or D_i^-) is the intersection of a disk with a halfplane, which we will call a *disk part*. We describe how to preprocess $\mathcal{D}'^+$; $\mathcal{D}'^-$ can be preprocessed similarly.

Compute the convex hull H_1 of $\mathcal{D}'^+$. The boundary of H_1 consists of a sequence of circular arcs and connecting line segments. For all disk parts of $\mathcal{D}'^+$ that do not contribute to H_1, again compute the convex hull H_2, and repeat it until all disk parts contribute to some convex hull. The convex hulls $H_1, \ldots, H_j$ are called *convex layers* of $D_1^+, \ldots, D_m^+$.

Lemma 2 *The convex layers $H_1, \ldots, H_j$ have the following properties with respect to a line l:*

- *if $l \cap H_i = \emptyset$, then $l \cap H_j = \emptyset$ for all $j \geq i$,*
- *the disk parts of H_i that intersect l are consecutive in the ordering along the boundary of H_i,*

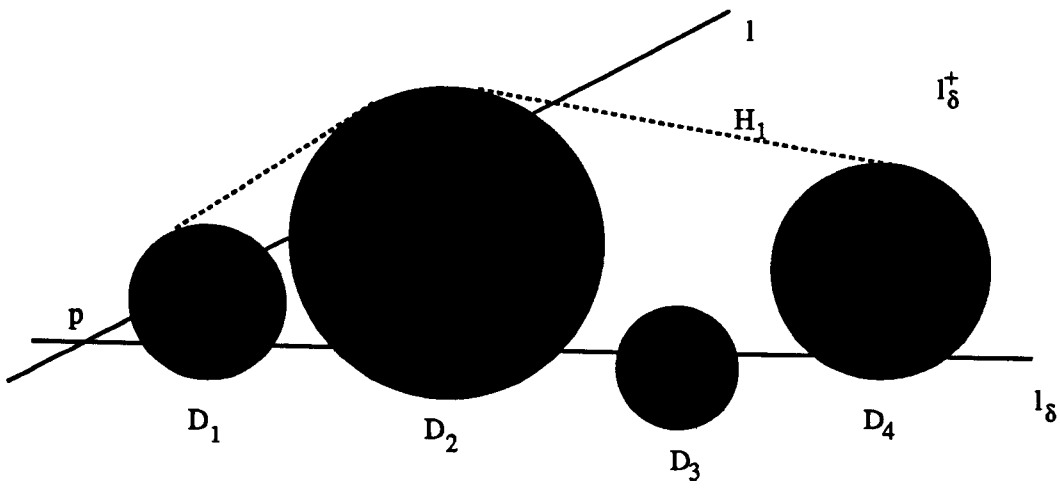

Figure 1: Disjoint disks intersected by a line.

- *if $l \cap H_i \neq \emptyset$, then there is a disk part contributing to H_i which intersects l.*

Proof. The first part follows because H_j, for all $j > i$, lies in the interior of H_i.

To prove the second part, suppose that there are three disk parts D_1^+, D_2^+, D_3^+ in some layer H_i, which intersect l_δ in this order from left to right, and that l intersects only D_1^+ and D_3^+. Then D_2^+ lies inside the convex hull of D_1^+ and D_3^+, and therefore does not contribute to H_i, a contradiction.

For the third part, assume that l intersects two edges h_1 and h_2 of H_i. If h_1 or h_2 is the boundary of some disk part, we are done, so assume that both h_1 and h_2 are line segments connecting circular arcs of H_i. Obviously, $h_1, h_2 \subset l_\delta^+$, $h_1 \neq h_2$, and there must be a circular arc γ between h_1 and h_2 on H_i. Since the disk part D^+, whose boundary contains γ, is also incident upon l_δ, l has to intersect D^+. □

In view of the above lemma, the disk parts of $\mathcal{D}'^+$ intersecting l can be reported as follows. Let $i = 1$. If l does not intersect H_i, we are done, otherwise report the answers of H_i, and continue with $i = i+1$. Similarly, we can search in $\mathcal{D}'^-$. Observe that no disk is reported twice.

A straightforward implementation of the search algorithm leads to $O((k+1)\log m)$ query time, where k is the number of answers to the query. An application of fractional cascading (see [3]), using the ordering on the slopes of lines touching the convex hulls, leads to an improved query time of $O(k + \log m)$. The space required to store the convex layers is easily seen to be $O(m)$.

Now we drop the assumption that p lies on the same side of all disks. Let $D_1, \ldots, D_m$ be the disks intersecting l_δ from left to right. Partition the disks into subsets $\{D_1, \ldots, D_{\sqrt{m}}\}, \ldots, \{D_{i\sqrt{m}+1}, \ldots, D_m\}$, where $i \leq \sqrt{m}$. Hence, each subset consists of at most $\sqrt{m}$ disks. For each subset, build a structure as described above.

When performing a query with a line l, determine in $O(\log m)$ time, the subset in which disks intersect l_δ on both sides of $p = l_\delta \cap l$. Traverse this subset completely and report all disks that intersect l. For all the other subsets, p lies completely to one side of the disks in that subset, and they are queried as described above. This gives us an $O(m)$ space data structure that solves the associated query problem in $O(\sqrt{m}\log m + k)$ time.

Theorem 1 *Let $\mathcal{D}$ be a set of n non-intersecting disks in the plane. There exists a linear space data structure that stores $\mathcal{D}$, such that all k disks in $\mathcal{D}$ intersecting a query line, can be reported in time $O(n^\beta + k)$.*

Proof. Construct the conjugation tree for disks and the associated structures, as described above. It uses linear space, as each disk is stored in the associated structure S_δ of only one node, and S_δ requires only linear space.

All associated structures that may return non-empty sets are visited, because a query line that avoids some region of the conjugation tree will never intersect a disk lying completely in that region. Correctness in the associated structures follows from the first and the second part of Lemma 2, and the above discussion.

The bound on query time follows from the third part of Lemma 2, the above discussion, and Lemma 1. □

Let us now consider queries with a line segment rather than with a line. The key to our solution is the following lemma.

Lemma 3 *Let s be an arbitrary line segment in the plane, and let R be the infinite strip containing s, whose bounding lines are perpendicular to s, and contain the endpoints of s. Then any disk D that intersects s, either contains one of the endpoints of s, or has its center in R, or both.*

Proof. Let s and R be as in the lemma. Suppose there is a disk D that intersects s, but does not have its center m in R. Let q_1 and q_2 be the endpoints of s, and let q be an intersection point of s and D. Since m is not in R, it is easy to see that the point of s closest to m is one of its endpoints, say say q_1. Then the distance from m to q_1 is not greater than the distance from m to any point on s, in particular, to q. Thus q_1 is contained in D. □

The lemma shows that if we want to report all disks in a set that intersect a line segment s, it suffices to report the disks that: (i) contain one of the endpoints of s, and (ii) intersect the line containing the segment s and have their centers in the strip R. Notice that these two sets of disks are not disjoint, but it is easy to circumvent reporting a disk more than once, and we leave the details to the reader.

The set $\mathcal{D}$ of non-intersecting disks $D_1, \ldots, D_n$ partitions the plane into $n+1$ regions. Add to this partition n horizontal line segments, starting at the leftmost point of every disk, and extending leftward until it hits another disk. Similarly,

add horizontal segments at the rightmost points of the disks. This results in a subdivision of the plane into $O(n)$ regions, which is monotone in the horizontal direction. Preprocess this subdivision for efficient point location, choosing one of the methods that works for circular arcs too, such as Cole's [5], or the structure of Edelsbrunner et al. [8]. This solves the problem of finding the disks that contain the endpoints of the query segment s in $O(n)$ space and $O(\log n)$ query time (there are at most two answers).

To find the other disks that intersect s, the following structure is used. Let T be a standard conjugation tree on the centers of the disks to be stored, such that every center of a disk is stored with one leaf. Every internal node corresponds to a set of centers that lie in the subtree rooted at this node. Choose a small constant $\epsilon > 0$. For every internal node δ on the level $i \cdot \lfloor \frac{1}{2}\epsilon \log n \rfloor$, $1 \leq i \leq \lfloor 2/\epsilon \rfloor$, associate with δ a conjugation tree for the disjoint disks to which δ corresponds, as described above. This main tree is used to extract the disks with their center in the strip R. The other internal nodes have no associated structure. (This technique of building hierarchical structures is based on ideas of Dobkin and Edelsbrunner [6].)

A query with the line segment $s = \overline{q_1 q_2}$ is performed as follows. Let l_1 and l_2 be the lines perpendicular to s, containing q_1 and q_2 respectively. Search with l_1 and l_2, by recursively continuing the search in the children of a node δ if the corresponding region is intersected by l_1 or l_2. If the region associated with δ lies completely on one side of both l_1 and l_2, then return from recursion. If the region lies completely between l_1 and l_2, then descend from δ to the first (highest) level in the tree where associated structures are stored. Search in these associated structures with the line containing s as the query line.

Theorem 2 *Let $\mathcal{D}$ be a set of n non-intersecting disks in the plane. $\mathcal{D}$ can be stored in a linear size data structure, such that all k disks in $\mathcal{D}$ intersecting a query line segment can be reported in $O(n^{\beta+\epsilon} + k)$ time, for any $\epsilon > 0$.*

Proof. The tree described above, together with the point location structure, yields the bounds. The point location structure takes care of the disks containing an endpoint of the query segment, and the tree T is used to report the other disks. In the main tree all disks are selected for which the center lies between l_1 and l_2. For these disks, searching with the line segment or the line containing this segment gives the same answers (cf. Lemma 3).

To prove the space bound, observe that every $\lfloor \frac{1}{2}\epsilon \log n \rfloor^{th}$ level of the tree stores each disk in exactly one associated structure. Consequently, the total space requirement is $O(2n/\epsilon) = O(n)$.

The query with the lines l_1 and l_2 on the main tree select $O(n^{\beta}) \cdot O(2^{\lfloor \frac{1}{2}\epsilon \log n \rfloor}) = O(n^{\beta+\epsilon/2})$ nodes of the main tree in which the associated structure is searched. The total query time (excluding the number of answers) $Q(n)$ satisfies $Q(n) \leq Q(n/2) + Q(n/4) + O(n^{\beta+\epsilon/2}) = O(n^{\beta+\epsilon})$ (cf. Lemma 1), for any $\epsilon > 0$. $\square$

4 Intersection queries in intersecting disks

We will now generalize our algorithms to sets of arbitrary (possibly intersecting) disks. Again we will first consider the case of queries with a line and then extend it to line segments. The solutions we obtain are based on a quite different approach. We will use transformations that map disks to points in three-dimensional space, and then apply known techniques to answer the query. We are given a set $\mathcal{D}$ of n possibly intersecting disks in the plane, and we want to report all intersections between $\mathcal{D}$ and a query line l.

Let φ be a function that maps a disk D in the plane with center (a, b) and radius r to a point in $\mathbb{R}^3$ such that

$$\varphi(D) = (a, b, r).$$

Let $\mathcal{D}^* = \{\varphi(D) \mid D \in \mathcal{D}\}$ be the set of n points in $\mathbb{R}^3$. For a given line l define the wedge W_l as follows. If l is the vertical line $x = c$, then

$$W_l = \{(x, y, z) \mid z \geq |x - c|\}.$$

On the other hand, if l is the line $y = mx + c$, then

$$W_l = \left\{(x, y, z) \mid z \geq \left|\frac{mx - y + c}{\sqrt{m^2 + 1}}\right|\right\}.$$

Lemma 4 *A disk D intersects a line l if and only if $D^* \in W_l$.*

Proof. Follows from elementary geometry. □

Thus the problem reduces to reporting the points of $\mathcal{D}^*$ lying inside the wedge W_l. Chazelle and Welzl [4] have shown that a given set of n points in $\mathbb{R}^3$ can be preprocessed into a data structure of size $O(n \log n)$ so that all k points lying inside a query tetrahedron can be reported in time $O(n^{2/3} \log^2 n + k)$. Clearly, the wegde is a special case of a tetrahedron. Therefore we have

Theorem 3 *Given a set $\mathcal{D}$ of n disks in the plane, one can preprocess $\mathcal{D}$ into a data structure of size $O(n \log n)$, so that given a query line l one can report all k disks intersecting l in time $O(n^{2/3} \log^2 n + k)$.*

Let us now consider queries with a line segment s. Using Lemma 3 we can again split the problem into two subproblems: (i) find the disks that contain an endpoint of s, and (ii) find the disks with center in the strip R perpendicular to s, which intersect the line l containing s. To solve the second subproblem, we transform the problem to $\mathbb{R}^3$, and let the line segment s lie in the xy-plane. Let H_s be the vertical slab in $\mathbb{R}^3$ that contains the segment s, and is bounded by vertical planes normal to s and passing through the endpoints of s. Define $\Delta_s = W_l \cap H_s$. It is easy to see that a disk D whose center lies in R intersects s if and only if $\varphi(D) \in \Delta_s$. Since Δ_s

is bounded by four planes, we can use Chazelle and Welzl's result [4] to report all k_1 points of $\mathcal{D}^* \cap \Delta_s$ in time $O(n^{2/3}\log^2 n + k_1)$ using $O(n \log n)$ space.

Next we describe how to solve the first subproblem: given a set $\mathcal{D}$ of disks, store them so that for a query point p, the disks of $\mathcal{D}$ containing p can be reported efficiently. Let φ' be a function that maps a disk D with center (a, b) and radius r to the point $(a, b, a^2 + b^2 - r^2)$ in $\mathbb{R}^3$. Let $\hat{\mathcal{D}} = \{\varphi'(D) | D \in \mathcal{D}\}$. Also, map a point $p = (c, d)$ to the plane $\hat{p} : z = 2cx + 2dy - (c^2 + d^2)$. Now the following lemma holds:

Lemma 5 *A point $p \in D$ if and only if the point $\varphi'(D)$ lies below the plane $\hat{p}$.*

Using the results of Chazelle and Welzl [4], we obtain a data structure of size $O(n \log n)$ that reports, in time $O(n^{2/3}\log^2 n + k_2)$, all k_2 points of $\hat{\mathcal{D}}$ lying below a query plane. Combining this with the result for subproblem (ii) we obtain

Theorem 4 *Given a set $\mathcal{D}$ of n (possibly intersecting) disks in the plane, we can construct a data structure of $O(n \log n)$ size, such that for a query segment s, all k disks of $\mathcal{D}$ intersecting s can be reported in time $O(n^{2/3}\log^2 n + k)$.*

5 Ray shooting in a set of disks

A ray shooting query in a set $\mathcal{D}$ of disks asks for the first disk hit by a ray in a given direction d, emanating from a given point q. Such a query is well-defined only when the starting point q does not lie in any disk of $\mathcal{D}$. So in the sequel we assume that the starting point q for the shooting query lies outside the union of the disks. This can easily be tested in $O(\log n)$ time by building a point location structure on the union of the disks, which has linear size (see [8], [12]). We use the following lemma to simplify the problem:

Lemma 6 *Given a set $\mathcal{D}$ of (possibly intersecting) disks, and a ray shooting query with point q and direction d, then the first disk that is hit (i.e., the answer to the query) has its center in the halfplane H, which is bounded by the line through q and perpendicular to d, and contains the query ray.*

Proof. Immediate by Lemma 3. □

To solve the ray shooting problem, we construct a two-level data structure. The first level is used to extract the disks that have their centers in the correct halfplane. The second level is used to determine the first disk hit when we shoot in these disks from 'minus infinity' along the line containing the query ray.

As in the previous sections, we build a conjugation tree T on the set of centers of the disks, and every node δ on a level $i \cdot \lfloor \frac{1}{2}\epsilon \log n \rfloor$ has an associated structure, to be described below, for shooting from infinity in the disks whose centers lie in the subtree rooted at δ. To perform a shooting query we first determine the correct

nodes whose associated structures contain the disks with center in the halfplane H and perform a shooting query from infinity in the associated structures.

Shooting from infinity asks for the first disk intersected by a directed line l. To solve this problem we store all disks in a conjugation tree as described in Section 2, that is, we construct a conjugation tree on the set of centers and a disk D is associated with the highest node δ where l_δ intersects D. Let $\mathcal{D}_\delta$ be the set of disks associated with the node δ. We now visit each node δ whose region intersects l and, using the associated structure S_δ, determine the first disk of $\mathcal{D}_\delta$ hit by l.

It remains to describe the structures S_δ. We will show that S_δ requires linear storage and answers a query in time $O(\log^2 n)$. A similar analysis as in Section 3 will show that this leads to a linear size data structure with query time of $O(n^{\beta+\epsilon})$, for any $\epsilon > 0$.

All disks in $\mathcal{D}_\delta$ intersect l_δ. Without loss of generality we assume that l_δ is horizontal. Let l be the query line. Assume that l intersects l_δ in a point p and that the starting point q of the ray lies above l_δ. (The special case that l does not intersect l_δ can easily be dealt with.) Now there are two cases: either l intersects a disk above l_δ or not. To determine the first case, let C be the outer-contour of the set of disks plus the halfplane below l_δ. C is a (infinite) polygon with circular arcs and straight line segments (also called a splinegon) of linear complexity. See Figure 2 for an example. Shooting inside a splinegon can be done relatively easy in time $O(\log^2 n)$ using $O(n \log n)$ storage by adapting the shooting algorithm for simple polygons by Chazelle and Guibas [2]. Using the special nature of the splinegon involved, the storage can be reduced to linear. Details are left to the reader. If we hit a circle arc, the corresponding disk is the required answer. On the other hand, if we hit l_δ (in point p), then l does not intersect any disk of $\mathcal{D}_\delta$ above l_δ.

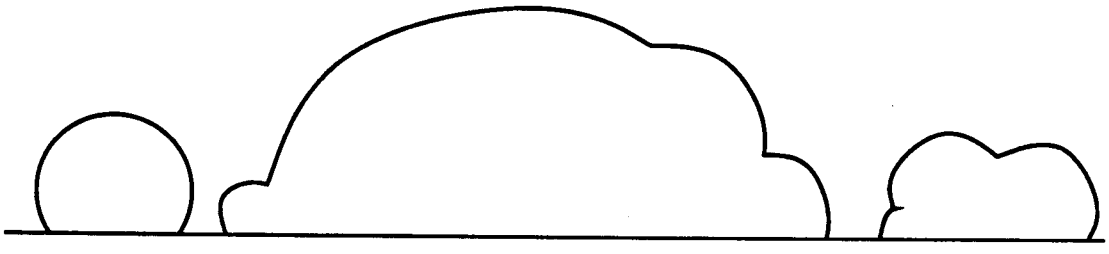

Figure 2: The top splinegon.

Note that the contour of the disks below l_δ can be viewed as a number of 'pockets'. The pockets are the connected components of the boundary of the outside of the union of the disks, intersected by the halfplane l_δ^-. See Figure 3. Each of these pockets is a splinegon itself. Each of these splinegons we preprocess for shooting. We first determine the pocket p belongs to. This can easily be done in time $O(\log n)$

by binary search on the line l_δ. Next we shoot from p in the corresponding splinegon. In this way we find in time $O(\log^2 n)$ either an answer or know that no answer exists (when we don't hit anything). Because the total size of all the splinegons is linear (because the contour of a set of circles has linear size) S_δ uses linear storage.

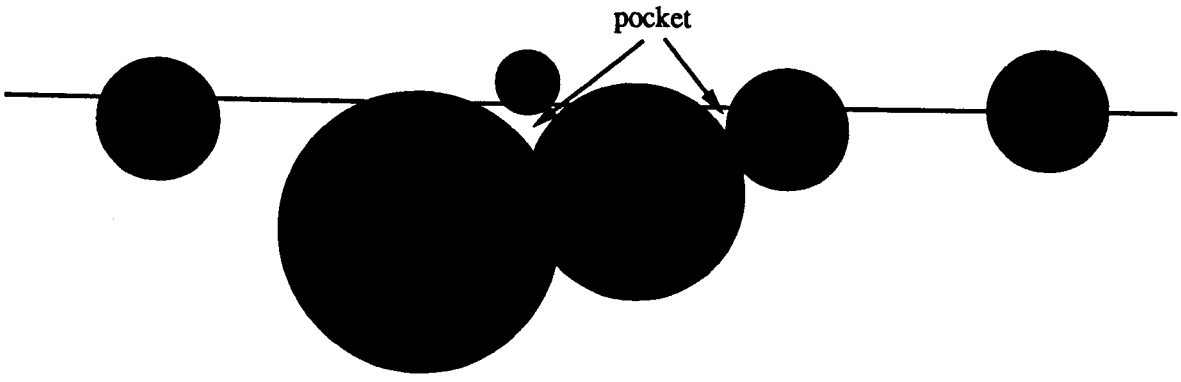

Figure 3: The pockets.

Using the special properties of the splinegons obtained the shooting can in fact be done in time $O(\log n)$. Because this does not influence the total time bound we did not describe this technique. Putting all of it together, using a similar analysis as in Section 3 we conclude:

Theorem 5 *Given a set $\mathcal{D}$ of n (possibly intersecting) disks in the plane, it can be stored in a data structure of linear size, such that the first disk hit by a query ray can be determined in $O(n^{\beta+\epsilon})$ time, for any $\epsilon > 0$.*

6 Conclusions

In this paper we have presented some new data structures for storing sets of disks, which allow for efficient intersection queries. For sets of disjoint disks we obtained structures that use only linear storage and answer a query in time $O(n^{\beta+\epsilon} + k)$, where $\beta = \log_2(1 + \sqrt{5}) - 1$, k is the number of answers, and $\epsilon > 0$ an arbitrarily small positive constant. For arbitrary disks, we constructed data structures of size $O(n \log n)$ with query time $O(n^{2/3} \log^2 n + k)$. We also presented efficient algorithms for shooting queries.

The query time of algorithms described in Section 3 and Section 5 can in fact be improved. To this end we replace the conjugation tree by a structure based on ϵ-nets as described by Haussler and Welzl [11]. This structure is also based on convex decomposition of the plane. Structures can be associated to internal nodes in a similar way as described above, although the details are a bit more involved.

This improves all time bounds to $O(n^{2/3+\epsilon})$ rather than $O(n^{\beta+\epsilon})$ without increasing the space requirement.

In the preceding sections we have never mentioned preprocessing time bounds for the data structures and only talked about query time and amount of storage required. All structures however can be constructed deterministically in polynomial time.

Many directions for further research remain open. At the moment we are working on improved data structures based on spanning trees with low stabbing number (see e.g. [1]). Query times in such structures lie close to $O(\sqrt{n})$ but the storage consumption is normally a bit higher. An important open problem concerns the extension to more general objects than disks. Although some of the results presented can be extended to more general objects (in particular the intersection queries with lines), most results depend on the fact that we deal with disks. Structures for intersection queries that store more general objects are still unavailable. Another direction for further research involves queries with curved objects, e.g. circle arcs.

References

[1] Agarwal, P.K., Ray shooting and other applications of spanning trees with low stabbing number, *Proc. 5th ann. symp. on Comp. Geometry* (1989), pp. 315-325.

[2] Chazelle, B., and L.J. Guibas, Visibility and intersection problems in plane geometry, *Proc. 1st ann. symp. on Comp. Geometry* (1985), pp. 135-146.

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

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

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

[6] Dobkin, D.P., and H. Edelsbrunner, Space searching for intersecting objects, *J. of Algorithms 8* (1987), pp. 348-361.

[7] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

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

[9] Edelsbrunner, H., and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inf. Proc. Lett. 23* (1986), pp. 289-293.

[10] Guibas, L., M. Overmars, and M. Sharir, *Ray shooting, implicit point location, and related queries in arrangements of segments*, Techn. Rep. No. 433, New York University, 1989.

[11] Haussler, D., and E. Welzl, ε-nets and simplex range queries, *Discr. & Comp. Geometry 2* (1987), pp. 127-151.

[12] Kedem, K., R. Livne, J. Pach, and M. Sharir, On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles, *Discr. & Comp. Geometry 1* (1986), 59-71.

[13] Overmars, M.H., H. Schipper, and M. Sharir, Storing line segments in partition trees, *BIT*, to appear.

[14] Sharir, M., *Efficient algorithm for reporting disc stabbing by points in the plane*, preliminary note, 1989.