Finding pairwise intersections inside a query range

Finding Pairwise Intersections Inside a Query Range*

Mark de Berg[1], Joachim Gudmundsson[2], and Ali D. Mehrabi[1]

[1] Department of Computer Science, TU Eindhoven, the Netherlands
[2] Department of Computer Science, University of Sydney, Australia

Abstract. We study the following problem: preprocess a set $\mathcal{O}$ of objects into a data structure that allows us to efficiently report all pairs of objects from $\mathcal{O}$ that intersect inside an axis-aligned query range Q. We present data structures of size $O(n \operatorname{polylog} n)$ and with query time $O((k+1) \operatorname{polylog} n)$ time, where k is the number of reported pairs, for two classes of objects in the plane: axis-aligned rectangles and objects with small union complexity. For the 3-dimensional case where the objects and the query range are axis-aligned boxes in $\mathbb{R}^3$, we present a data structures of size $O(n\sqrt{n} \operatorname{polylog} n)$ and query time $O((\sqrt{n}+k) \operatorname{polylog} n)$. When the objects and query are fat, we obtain $O((k+1) \operatorname{polylog} n)$ query time using $O(n \operatorname{polylog} n)$ storage.

1 Introduction

The study of geometric data structures is an important subarea within computational geometry, and range queries form one of the most widely studied topics within this area [1,11]. In a range query, the goal is to report or count all points from a given set $\mathcal{O}$ that lie inside a query range Q. The more general version, where $\mathcal{O}$ contains other objects than just points and the goal is to report all objects intersecting Q, is often called intersection searching and it has been studied extensively as well.

A common characteristic of the range-searching and intersection-searching problems studied so far, is that whether an object $o_i \in \mathcal{O}$ should be reported (or counted) depends only on o_i and Q. In this paper we study a range-searching variant where we are interested in reporting *pairs* of objects that satisfy a certain criterion. In particular, we want to preprocess a set $\mathcal{O} = \{o_1, \ldots, o_n\}$ of n objects in the plane such that, given a query range Q, we can efficiently report all pairs of objects o_i, o_j that intersect inside Q. An obvious approach is to precompute all intersections between the objects and store the intersections in a suitable intersection-searching data structure. This may give fast query times, but in the worst case any two objects intersect, so $\Omega(n^2)$ is a lower bound on the storage for this approach. The main question is thus: can we achieve fast query times

* M. de Berg and A. D. Mehrabi were supported by the Netherlands Organization for Scientific Research (NWO) under grants 024.002.003 and 612.001.118, respectively.

with a data structure that uses subquadratic (and preferably near-linear) storage in the worst case?

We answer this question affirmatively when Q is an axis-aligned rectangle in the plane and the objects are either axis-aligned rectangles or objects with small union complexity. For axis-aligned rectangles our data structure uses $O(n \log n)$ storage and has $O((k+1) \log n \log^* n)$ query time,[3] where k is the number of reported pairs of objects. Our data structure for classes of objects with small union complexity—disks and other types of fat objects are examples—uses $O(U(n) \log n)$ storage, where $U(n)$ is maximum union complexity of n objects from the given class, and it has $O((k+1) \log^2 n)$ query time. We also consider a 3-dimensional extension of the planar case, where the range Q and the objects in $\mathcal{O}$ are axis-aligned boxes. Our data structures for this setting has size $O(n\sqrt{n} \log n)$ and query time $O((k+1) \log^2 \log^* n)$. For the special case where the query range and the objects are fat, we present a data structure of $O(n \log^2 n)$ size and $O((k+1) \log^2 n \log^* n)$ query time.

2 Axis-aligned objects

In this section we study the case where the set $\mathcal{O}$ is a set of n axis-aligned rectangles in the plane or boxes in $\mathbb{R}^3$. Our approach for these cases is the same and uses the following two-step query process.

1. Compute a *seed set* $\mathcal{O}^*(Q) \subseteq \mathcal{O}$ of objects such that the following holds: for any two objects o_i, o_j in $\mathcal{O}$ such that o_i and o_j intersect inside Q, at least one of o_i, o_j is in $\mathcal{O}^*(Q)$.
2. For each seed object $o_i \in \mathcal{O}^*(Q)$, perform an intersection query with the range $o_i \cap Q$ in the set $\mathcal{O}$, to find all objects $o_j \neq o_i$ intersecting o_i inside Q.

To make this approach efficient, we need that the seed set $\mathcal{O}^*(Q)$ does not contain too many objects that do not give an answer in Step 2. For the planar case our seed set will satisfy $|\mathcal{O}^*(Q)| = O(1+k)$, where k denotes the number of pairs of objects in $\mathcal{O}$ that intersect inside Q, while for the 3-dimensional case we will have $|\mathcal{O}^*(Q)| = O(\sqrt{n}+k)$.

2.1 The planar case

Axis-aligned segments. As a warm-up exercise we start with the case where $\mathcal{O}$ consists of axis-aligned segments. Let $\mathcal{O} = \{s_1, \ldots, s_n\}$ be a set of axis-aligned segments, and let $\mathsf{V}(\mathcal{O})$ and $\mathsf{H}(\mathcal{O})$ denote the set of vertical and horizontal segments in $\mathcal{O}$, respectively. We assume for simplicity that we are only interested in intersections between horizontal and vertical segments; the solution can easily be adapted to the case where we also want to report intersections between two horizontal (or two vertical) segments.

[3] Here $\log^* n$ denotes the iterated logarithm.

The key to our approach is to be able to efficiently find the seed set $\mathcal{O}^*(Q)$. To this end, during the preprocessing we compute an $O(n)$-sized subset W of the intersection points in $\mathcal{O}$. We call intersection points in W *witnesses*. The witness set W is defined as follows: for each line segment $s_i \in \mathsf{V}(\mathcal{O})$ we put the topmost and bottommost intersection points of s_i with a segment from $\mathsf{H}(\mathcal{O})$ (if any) into W; for each line segment $s_i \in \mathsf{H}(\mathcal{O})$ we put the leftmost and rightmost intersection points of s_i with a segment from $\mathsf{V}(\mathcal{O})$ (if any) into W. Since we take at most two witness points for each line segment, the size of W is clearly at most $2n$.

Our data structure to find the seed set $\mathcal{O}^*(Q)$ now consists of three components: First, we store W in a data structure $\mathcal{D}_1$ for 2-dimensional orthogonal range reporting. Second, we store $\mathsf{V}(\mathcal{O})$ in a data structure $\mathcal{D}_2$ that allows us to decide if there are any segments that completely cross the query rectangle Q from top to bottom, and that can report all such segments. Third, we store $\mathsf{H}(\mathcal{O})$ in a data structure $\mathcal{D}_3$ that allows us to decide if there are any segments that completely cross the query rectangle Q from left to right.

Step 1 of the query procedure, where we compute $\mathcal{O}^*(Q)$, proceeds as follows.

1(i) Perform a query in $\mathcal{D}_1$ to find all witness points inside Q. For each reported witness point, insert the corresponding segment into $\mathcal{O}^*(Q)$.

1(ii) Perform queries in $\mathcal{D}_2$ and $\mathcal{D}_3$ to decide if the number of segments crossing Q completely from top to bottom, and the number of segments crossing Q completely from left to right, are both non-zero. If so, report all segments crossing completely from top to bottom, and put them into $\mathcal{O}^*(Q)$.

Lemma 1. *Let s_i, s_j be two segments in $\mathcal{O}$ such that $s_i \cap s_j \in Q$. Then at least one of s_i, s_j is put into $\mathcal{O}^*(Q)$ by the above query procedure.*

Proof. If s_i crosses Q completely from left to right and s_j crosses Q completely from top to bottom (or vice versa), then one of them will be put into $\mathcal{O}^*(Q)$ in Step 1(ii). Otherwise at least one of the segments, say s_i, has an endpoint v inside Q. But then the intersection point on s_i closest to v, which is a witness point, must lie inside Q. Hence, s_i is put into $\mathcal{O}^*(Q)$ in Step 1(i). □

In Step 2 of the query procedure we need to report, for each segment s_i in the seed set $\mathcal{O}^*(Q)$, the segments $s_j \in \mathcal{O}$ intersecting $s_i \cap Q$. Thus we store $\mathcal{O}$ in a data structure $\mathcal{D}_4$ that can report all segments intersecting an axis-aligned query segment. Putting everything together we obtain the following theorem.

Theorem 1. *Let $\mathcal{O}$ be a set of n axis-aligned segments in the plane. Then there is a data structure that uses $O(n \log n)$ storage and can report, for any axis-aligned query rectangle Q, all pairs of segments s_i, s_j in $\mathcal{O}$ such that s_i intersects s_j inside Q in $O((k+1) \log n \log^* n)$ time, where k denotes the number of answers.*

Proof. For the data structure $\mathcal{D}_1$ on the set W we can take a standard 2-dimensional range tree [3], which uses $O(n \log n)$ storage. If we apply fractional cascading [3], reporting the witness points inside Q takes $O(\log n + \#\text{answers})$ time. For $\mathcal{D}_2$ (and, similarly, $\mathcal{D}_3$) we note that a vertical segment $s_i := x_i \times [y_i, y_i']$

crosses $Q := [x_Q, x'_Q] \times [y_Q, y'_Q]$ if and only if the point (x_i, y_i, y'_i) lies in the range $[x_Q, x'_Q] \times [-\infty, y_Q] \times [y'_Q, \infty]$. Hence, we can use the data structure of Subramanian and Ramaswamy [13], which uses $O(n \log n)$ storage and has $O(\log n \log^* n + \#\text{answers})$ query time. Hence, the supporting data structures for Step 1 use $O(n \log n)$ storage, and finding the seed set takes $O(\log n \log^* n + |\mathcal{O}^*(Q)|)$ time.

It remains to analyze Step 2 of the query procedure. First notice that the problem of finding for a given $s_i \in \mathcal{O}^*(Q)$ all $s_j \in \mathcal{O}$ such that $s_i \cap Q$ intersects s_j, is the same range-searching problem as Step 1(ii), except that the query range is a line segment this time. Hence, we again transform the problem to a 3D range-searching problem on points and use the data structure of Subramanian and Ramaswamy [13]. Thus the running time of Step 2 is $\sum_{s_i \in \mathcal{O}^*(Q)} O(\log \log^* n + k_i)$, where k_i denotes the number of segments in $\mathcal{O}$ that intersect s_i inside Q. Since $|\mathcal{O}^*(Q)| \leqslant 2k$ where k is the total number of reported pairs—each segment in $\mathcal{O}^*(Q)$ intersects at least one other segment inside Q and for every reported pair we put at most two segments into the seed set—the time for Step 2 is $O(|\mathcal{O}^*(Q)| \log n \log^* n + k) = O((k+1) \log n \log^* n)$. □

Axis-aligned rectangles. We now extend our approach to axis-aligned rectangles. Let $\mathcal{O} = \{r_1, \ldots, r_n\}$ be a set of axis-aligned rectangles in the plane. Similar to the case of axis-aligned segments we need to find the seed set $\mathcal{O}^*(Q)$ efficiently.

As before, we first define a witness set W. The witnesses in W are now axis-aligned segments rather than just points. For each rectangle $r_i \in \mathcal{O}$ we define at most ten witness segments, two for each edge of r_i and two in the interior of r_i, as follows—see also Fig. 1. Let e be an edge of r_i, and consider the set $S(e) := e \cap (\cup_{j \neq i} r_j)$, that is, the part of e covered by the other rectangles. The set $S(e)$ consists of a number of sub-edges of e. If e is vertical then we add the topmost and bottommost sub-edge from $S(e)$ (if any) to W; if e is horizontal we add the leftmost and rightmost sub-edge to W. The two witness segments in the interior of r_i are defined as follows. Suppose there are vertical edges (belonging to other rectangles r_j) completely crossing r_i from top to bottom. Then we put $e' \cap r_i$ into W, where e' is the rightmost such crossing edge. Similarly, we put into W the topmost horizontal edge e'' completely crossing r_i from left to right. Our data structure to find the seed set $\mathcal{O}^*(Q)$ now consists of the following components.

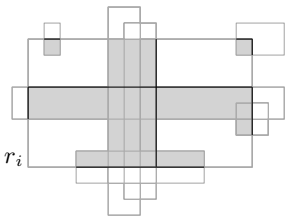


Fig. 1: Gray areas are intersections with r_i, black segments indicate witness segments.

- We store the witness set W in a data structure $\mathcal{D}_1$ that allows us to report the set of segments that intersect the query rectangle Q.
- We store the vertical edges of the rectangles in $\mathcal{O}$ in a data structure $\mathcal{D}_2$ that allows us to decide if the set $\mathsf{V}(Q)$ of edges that completely cross a query

rectangle Q from top to bottom, is non-empty. The data structure should also be able to report all (rectangles corresponding to) the edges in $\mathsf{V}(Q)$.
- We store the horizontal edges of the rectangles in $\mathcal{O}$ in a data structure $\mathcal{D}_3$ that allows us to decide if the set $\mathsf{H}(Q)$ of edges that completely cross a query rectangle Q from left to right, is non-empty.
- We store $\mathcal{O}$ in a data structure $\mathcal{D}_4$ that allows us to report the set of rectangles that contain a query point q.

Step 1 of the query procedure, where we compute $\mathcal{O}^*(Q)$, proceeds as follows.

1(i) Perform a query in $\mathcal{D}_1$ to find all witness segments intersecting Q. For each reported witness segment, insert the corresponding rectangle into $\mathcal{O}^*(Q)$.

1(ii) Perform queries in $\mathcal{D}_2$ and $\mathcal{D}_3$ to decide if the sets $\mathsf{V}(Q)$ and $\mathsf{H}(Q)$ are both non-empty. If so, report all rectangles corresponding to edges in $\mathsf{V}(Q)$ and put them into $\mathcal{O}^*(Q)$.

1(iii) For each corner point q of Q, perform a query in $\mathcal{D}_4$ to report all rectangles in $\mathcal{O}$ that contain q, and put them into $\mathcal{O}^*(Q)$.

The next lemma can be proved using a case analysis—see the Appendix A.

Lemma 2. *Let r_i, r_j be two rectangles in $\mathcal{O}$ such that $(r_i \cap r_j) \cap Q \neq \emptyset$. Then at least one of r_i, r_j is put into $\mathcal{O}^*(Q)$ by the above query procedure.*

In the second part of the query procedure we need to report, for each rectangle r_i in the seed set $\mathcal{O}^*(Q)$, the rectangles $r_j \in \mathcal{O}$ intersecting $r_i \cap Q$. Thus we store $\mathcal{O}$ in a data structure $\mathcal{D}_5$ that can report all rectangles intersecting a query rectangle. Putting everything together we obtain the following theorem.

Theorem 2. *Let $\mathcal{O}$ be a set of n axis-aligned rectangles in the plane. There is a data structure that uses $O(n \log n)$ storage and can report, for any axis-aligned query rectangle Q, all pairs of rectangles r_i, r_j in $\mathcal{O}$ such that r_i intersects r_j inside Q in $O((k+1) \log n \log^* n)$ time, where k denotes the number of answers.*

Proof. For the data structure $\mathcal{D}_1$ on the set W we use the data structure developed by Edelsbrunner *et al.* [9], which uses $O(n \log n)$ preprocessing time and storage, and has $O(\log n + \#\text{answers})$ query time.

Data structure $\mathcal{D}_2$ (and, similarly, $\mathcal{D}_3$) answers the same type of query we needed when $\mathcal{O}$ contains segments. Hence, we can use the same data structure [13] which uses $O(n \log n)$ space and has $O(\log n \log^* n + \#\text{answers})$ query time. For data structure $\mathcal{D}_4$ we use the point-enclosure data structure developed by Chazelle [4], which uses $O(n)$ storage and can be used to report all rectangles in $\mathcal{O}$ containing a query point in $O(\log n + \#\text{answers})$ time.

The analysis of Step 2 is similar to the analysis for the case of axis-aligned segments, except that we now have $|\mathcal{O}^*(Q)| \leqslant 2k+4$, where k is the total number of pairs of rectangles that will be reported; the extra term "+4" is because in Step 1(iii) we may report at most one rectangle per corner of Q that does not have an intersection inside Q. Again, finding the rectangles in $\mathcal{O}$ intersecting $r_i \cap Q$, for a given $r_i \in \mathcal{O}^*(Q)$, can be done in $O(\log n \log^* n + \#\text{answers})$, leading to an overall query time of $O((k+1) \log n \log^* n)$. □

2.2 The 3-dimensional case

We now study the case where the set $\mathcal{O}$ of objects and the query range Q are axis-aligned boxes in $\mathbb{R}^3$. We first present a solution for the general case, and then an improved solution for the special case where the input as well as the query are cubes. Both solutions use the same query strategy as above: we first find a seed set $\mathcal{O}^*(Q)$ that contains at least one object o_i from every pair that intersects inside Q and then we find all other objects intersecting o_i inside Q.

The general case. Let $\mathcal{O} := \{b_1, \ldots, b_n\}$ be a set of axis-aligned boxes. The pairs of boxes b_i, b_j intersecting inside Q come in three types: (i) $b_i \cap b_j$ fully contains Q, (ii) $b_i \cap b_j$ lies completely inside Q, (iii) $b_i \cap b_j$ intersects a face of Q.

Type (i) is easy to handle without using seeds sets: we simply store $\mathcal{O}$ in a data structure for 3-dimensional point-enclosure queries [4], which allows us to report all boxes $b_i \in \mathcal{O}$ containing a query point in $O(\log^2 n + \#\text{answers})$ time. If we query this structure with a corner q of Q and report all pairs of boxes containing q then we have found all intersecting pairs of Type (i).

Lemma 3. *We can find all intersecting pairs of boxes of Type (i) in $O(\log^2 n + k)$ time, where k is the number of such pairs, with a structure of size $O(n \log n)$.*

For Type (ii) we proceed as follows. Note that a vertex of $b_i \cap b_j$ is either a vertex of b_i or b_j, or it is the intersection of an edge e of one of these two boxes and a face f of the other box. To handle the first case we create a set W of witness points, which contains for each box b_i all its vertices that are contained in at least one other box. We store W in a data structure for 3-dimensional orthogonal range reporting [13]. In the query phase we then query this data structure with Q, and put all boxes corresponding to the witness vertices inside Q into the seed set $\mathcal{O}^*(Q)$. For the second case we show next how to find the intersecting pairs e, f where e is a vertical edge (that is, parallel to the z-axis) and f is a horizontal face (that is, parallel to the xy-plane); the intersecting pairs with other orientations can be found in a similar way.

Let E be the set of vertical edges of the boxes in $\mathcal{O}$ and let F be the set of horizontal faces. We sort F by z-coordinate—we assume for simplicity that all z-coordinates of the faces are distinct—and partition F into $O(\sqrt{n})$ *clusters*: the cluster F_1 contains the first $\sqrt{n}$ faces in the sorted order, the second cluster F_2 contains the next $\sqrt{n}$ faces, and so on. We call the range between the minimum and maximum z-coordinate in a cluster its *z-range*. For each cluster F_i we store, besides its z-range and the set F_i itself, the following information. Let $E_i \subseteq E$ be the subset of edges that intersect at least one face in F_i, and let $\overline{E_i}$ denote the set of points obtained by projecting the edges in E_i onto the xy-plane. We store $\overline{E_i}$ in a data structure $\mathcal{D}(\overline{E_i})$ for 2-dimensional orthogonal range reporting. Note that an edge $e \in E$ intersects at least one face $f \in F_i$ inside Q if and only if $e \in E_i$ and $\overline{e}$ lies in $\overline{Q}$, the projection of Q onto the xy-plane.

A query with a box $Q = [x_1 : x_2] \times [y_1 : y_2] \times [z_1 : z_2]$ is now answered as follows. We first find the clusters F_i and F_j whose z-range contains z_1 and z_2, respectively, and we put (the boxes corresponding to) the faces in these clusters into the seed set $\mathcal{O}^*(Q)$. Next we perform, for each $i < t < j$, a query with the

projected range $\overline{Q}$ in the data structure $\mathcal{D}(\overline{E_i})$. For each of the reported points $\overline{e}$ we put the box corresponding to the edge e into the seed set $\mathcal{O}^*(Q)$. Finally, we remove any duplicates from the seed set.

We obtain the following lemma, whose proof is in the Appendix A.

Lemma 4. *Using a data structure of size $O(n\sqrt{n}\log n)$ we can find in time $O(\log n \log^* n + k)$ a seed set $\mathcal{O}^*(Q)$ of $O(\sqrt{n}+k)$ boxes containing at least one box from every intersecting pair of Type (ii), where k is the number of such pairs.*

It remains to handle the Type (iii) pairs, in which $b_i \cap b_j$ intersects a face of Q. We describe how to find the pairs such that $b_i \cap b_j$ intersects the bottom face of Q; the pairs intersecting the other faces can be found in a similar way.

We first sort the z-coordinates of the horizontal faces of the boxes in $\mathcal{O}$. For $1 \leqslant i \leqslant 2\sqrt{n}$, let h_i be a horizontal plane containing the $i\sqrt{n}$-th horizontal face in the ordering. These planes partition $\mathbb{R}^3$ into $O(\sqrt{n})$ horizontal slabs $\Sigma_0, \ldots, \Sigma_{2\sqrt{n}+1}$. We call a box $b \in \mathcal{O}$ *short* for a slab Σ_i if it has a horizontal face inside Σ_i, and we call it *long* if it completely crosses Σ_i. For each Σ_i, we store the short boxes in a list. We store the projections of the long boxes onto the xy-plane in a data structure $\mathcal{D}(\Sigma_i)$ for the 2-dimensional version of the problem, namely the structure Theorem 2.

A query with the bottom face of Q is now answered as follows. We first find the slab Σ_i containing the face. We put all short boxes of Σ_i into our seed set $\mathcal{O}^*(Q)$. We then perform a query with $\overline{Q}$, the projection of Q onto the xy-plane, in the data structure $\mathcal{D}(\Sigma_i)$. For each answer we get from this 2-dimensional query—that is, each pair of projections intersecting inside $\overline{Q}$—we directly report the corresponding pair of long boxes. (There is no need to go through the seed set for these pairs.) This leads to the following lemma for the Type (iii) pairs.

Lemma 5. *Using a data structure of size $O(n\sqrt{n}\log n)$ we can find in time $O(\sqrt{n} + (k+1)\log^* n \log n)$ a seed set $\mathcal{O}^*(Q)$ of $O(\sqrt{n})$ boxes plus a collection $B(Q)$ of pairs of boxes intersecting inside Q such that, for each pair of Type (iii) boxes, either at least one of these boxes is in $\mathcal{O}^*(Q)$ or b_i, b_j is a pair in $B(Q)$.*

In the second step of our query procedure we need to be able to report all boxes $b_j \in \mathcal{O}$ intersecting a query box B of the form $Q \cap b_i$, where $b_i \in \mathcal{O}^*(Q)$. Note that B and b_j intersect if and only if their projections onto the z-axis intersect and their projections onto the xy-plane intersect. Hence, we can answer the queries with a data structure $\mathcal{D}^*$ whose main tree is a (hereditary) segment tree [6] and whose associated structures are the data structure of Subramanian and Ramaswamy [13]. This leads to a structure using $O(n\log^2 n)$ storage and $O(\log^2 n \log^* n + \#\text{answers})$ query time.

Putting everything together we obtain the following theorem.

Theorem 3. *Let $\mathcal{O}$ be a set of n axis-aligned boxes in $\mathbb{R}^3$. Then there is a data structure that uses $O(n\sqrt{n}\log n)$ storage and that allows us to report, for any axis-aligned query box Q, all pairs of boxes b_i, b_j in $\mathcal{O}$ such that b_i intersects b_j inside Q in $O(\sqrt{n} + (k+1)\log^2 n \log^* n)$ time, where k denotes the number of answers.*

Fat boxes. Next we obtain better bounds when the boxes in $\mathcal{O}$ and the query box Q are fat, that is, when their *aspect ratio*—the ratio between the length of the longest edge and the length of the shortest edge—is bounded by a constant α. First we consider the case of cubes.

Let $\mathcal{O} := \{c_1, \cdots, c_n\}$ be a set of n cubes in $\mathbb{R}^3$ and let Q be the query cube. We compute a set W of witness points for each cube c_i, as follows. Let e be an edge of c_i, and consider the set $S(e) := e \cap (\cup_{j \neq i} c_j)$, that is, the part of e covered by the other cubes. We put the two extreme points from $S(e)$—in other words, the two points closest to the endpoints of e—into W. Similarly, we assign each face f of c_i at most four witness points, namely points from $S(f) := f \cap (\cup_{j \neq i} c_j)$ that are extreme in the directions parallel to f. For example, if f is parallel to the xy-plane, then we take points of maximum and minimum x-coordinate in $S(f)$ and points of maximum and minimum y-coordinate in $S(f)$ as witnesses. We store W in a data structure $\mathcal{D}_1$ for orthogonal range queries, and we store $\mathcal{O}$ in a data structure $\mathcal{D}_2$ for point-enclosure queries.

To compute $\mathcal{O}^*(Q)$ in the first phase of the query procedure, we query $\mathcal{D}_1$ to find all witness points inside Q and for each reported witness point, we insert the corresponding cube into $\mathcal{O}^*(Q)$. Furthermore, for each corner point q of Q, we query $\mathcal{D}_2$ to find the cubes in $\mathcal{O}$ that contain q, and we put them into $\mathcal{O}^*(Q)$.

Lemma 6. *Let c_i, c_j be two cubes in $\mathcal{O}$ such that $(c_i \cap c_j) \cap Q \neq \emptyset$. Then at least one of c_i, c_j is put into $\mathcal{O}^*(Q)$ by the above query procedure.*

Proof. Suppose $c_i \cap c_j$ intersects Q, and assume without loss of generality that c_i is not larger than c_j. If c_i or c_j contains a corner q of Q then the corresponding cube will be put into the seed set when we perform a point-enclosure query with q, so assume c_i and c_j do not contain a corner. We have two cases.

Case A: c_i does not intersect any edge of Q. Because c_i and Q are cubes, this implies that c_i is contained in Q or c_i intersects exactly one face of Q. Assume that c_i intersects the bottom face of Q; the cases where c_i intersects another face and where c_i is contained in Q can be handled similarly. We claim that at least one of the vertical faces of c_i contributes a witness point inside Q. To see this, observe that c_j will intersect at least one vertical face, f, of c_i inside Q, since c_j intersects c_i inside Q and c_i is not larger than c_j. Hence, the witness point on f with maximum z-coordinate will be inside Q. Thus c_i will be put into $\mathcal{O}^*(Q)$.

Case B: c_i intersects one edge of Q. (If c_i intersects more than one edge of Q then it would contain a corner of Q.) Assume without loss of generality that c_i intersects the bottom edge of the front face of Q; see Fig. 2. Observe that if c_j intersects the top face of c_i then the witness point of the face with minimum x-coordinate is inside Q. Similarly, if c_j intersects the back face of c_i (the face parallel to the yz-plane and with minimum x-coordinate) then the witness point of the face with maximum z-coordinate is inside Q. Otherwise, as illustrated in Fig 3, c_j must have an edge e parallel to the y-axis that intersects c_i inside Q, and one of the witness points on e will be inside Q—note that e lies fully inside Q because c_j does not contain a corner of Q. □

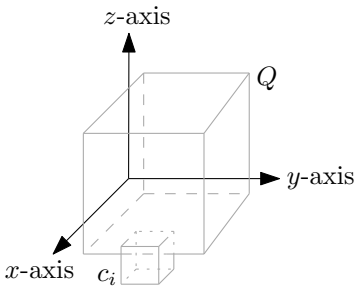


Fig. 2: Case B in the proof of Lemma 6; c_j is not shown.

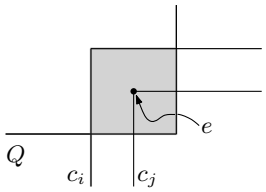


Fig. 3: Cross-section of Q, c_i, and c_j with a plane parallel to the xz-plane. The gray area indicates $Q \cap c_i$ in the cross-section.

To adapt the above solution to boxes of aspect ratio at most α, we cover each box $b_i \in \mathcal{O}$ by $O(\alpha^2)$ cubes, and preprocess the resulting collection $\widetilde{\mathcal{O}}$ of cubes as described above, making sure we do not introduce witness points for pairs of cubes used in the covering of the same box b_i. To perform a query, we cover Q by $O(\alpha^2)$ query cubes and compute a seed set for each query cube. We take the union of these seed sets, replace the cubes from $\widetilde{\mathcal{O}}$ in the seed set by the corresponding boxes in $\mathcal{O}$, and filter out duplicates. This gives us our seed set $\mathcal{O}^*(Q)$ for the second phase of the query procedure.

In the second phase we take each $b_i \in \mathcal{O}^*(Q)$ and report all $b_j \in \mathcal{O}$ intersecting $b_i \cap Q$, using the data structure $\mathcal{D}^*$ described in Subsection 2.2. We obtain the following theorem.

Theorem 4. *Let $\mathcal{O}$ be a set of n axis-aligned boxes in $\mathbb{R}^3$ of aspect ratio at most α. Then there is a data structure that uses $O(\alpha^2 n \log^2 n)$ storage and that allows us to report, for any axis-aligned query box Q of aspect ratio at most α, all pairs of cubes c_i, c_j in $\mathcal{O}$ such that c_i intersects c_j inside Q in $O(\alpha^2(k+1)\log^2 \log^* n)$ time, where k denotes the number of answers.*

Proof. The data structures $\mathcal{D}_1$ and $\mathcal{D}_2$ can be implemented such that they use $O(n \log n)$ storage, and have $O(\log n \log^* n + \#\text{answers})$ and $O(\log^2 n + \#\text{answers})$ query time, respectively [13,4]. In Step 2 of the query procedure we use the data structure $\mathcal{D}^*$ of Subsection 2.2, which uses $O(n \log^2 n)$ storage and has $O(\log^2 \log^* n + \#\text{answers})$ query time. The conversion of boxes of aspect ratio α to cubes give an additional factor $O(\alpha^2)$. □

3 Objects with small union complexity in the plane

In the previous section we presented efficient solutions for the case where $\mathcal{O}$ consists of axis-aligned rectangles. In this section we obtain results for classes of constant-complexity objects (which may have curved boundaries) with small

union complexity. More precisely, we need that $U(n)$, the maximum union complexity of any set of n objects from the class, is small. This is for instance the case for disks (where $U(m) = O(m)$ [12]) and for locally fat objects (where $U(m) = m2^{O(\log^* m)}$ [2]).

In Step 2 of the query algorithm of the previous section, we performed a range query with $o_i \cap Q$ for each $o_i \in \mathcal{O}^*(Q)$. When we are dealing with arbitrary objects, this will be expensive, so we modify our query procedure.

1. Compute a seed set $\mathcal{O}^*(Q) \subseteq \mathcal{O}$ of objects such that, for any two objects o_i, o_j in $\mathcal{O}$ intersecting inside Q, both o_i and o_j are in $\mathcal{O}^*(Q)$.
2. Compute all intersecting pairs of objects in the set $\{o_i \cap Q : o_i \in \mathcal{O}^*(Q)\}$ by a plane-sweep algorithm.

Next we describe how to efficiently find $\mathcal{O}^*(Q)$, which should contain all objects intersecting at least one other object inside Q, when the union complexity $U(n)$ is small. For each object $o_i \in \mathcal{O}$ we define $o_i^* := \bigcup_{o_j \in \mathcal{O}, j \neq i}(o_i \cap o_j)$ as the union of all intersections between o_i and all other objects in $\mathcal{O}$. Let $|o_i^*|$ denote the complexity (that is, number of vertices and edges) of o_i^*.

Lemma 7. $\sum_{i=1}^{n} |o_i^*| = O(U(n))$.

Proof. Consider the arrangement induced by the objects in $\mathcal{O}$. We define the *level* of a vertex v in this arrangement as the number of objects from $\mathcal{O}$ that contain v in their interior. We claim that every vertex of any o_i^* is a level-0 or level-1 vertex. Indeed, a level-k vertex for $k > 1$ is in interior of more than one object, which is easily seen to imply that it cannot be a vertex of any o_i^*.

Since the level-0 vertices are exactly the vertices of the union of $\mathcal{O}$, the total number of level-0 vertices is $U(n)$. It follows from the Clarkson-Shor technique [7] that the number of level-1 vertices is $O(U(n))$ as well. The lemma now follows, because each level-0 or level-1 vertex contributes to at most two different o_i^*'s. □

Our goal in Step 1 is to find all objects o_i such that o_i^* intersects Q. To this end consider the connected components of o_i^*. If o_i^* intersects Q then one of these components lies completely inside Q or an edge of Q intersects o_i^*.

Lemma 8. *We can find all o_i^* that have a component completely inside Q in $O(\log n + k)$ time, where k is the number of pairs of objects that intersect inside Q, with a data structure that uses $O(U(n) \log n)$ storage.*

Proof. For each o_i, take an arbitrary representative point inside each component of o_i^*, and store all the representative points in a structure for orthogonal range reporting. By Lemma 7 we store $O(U(n))$ points, and so the structure for orthogonal range reporting uses $O(U(n) \log n)$ storage.

The query time is $O(\log n + t)$, where t is the number of representative points inside Q. This implies the query time is $O(\log n + k)$, because if o_i^* has t_i representative points inside Q then o_i intersects $\Omega(t_i)$ other objects inside Q. This is true because the objects have constant complexity, so a single object o_j cannot generate more than a constant number of components of o_i^*. □

Next we describe a data structure for reporting all o_i^* intersecting a vertical edge of Q; the horizontal edges of Q can be handled similarly. The data structure is a balanced binary tree $\mathcal{T}$, whose leaves are in one-to-one correspondence to the objects in $\mathcal{O}$. For an (internal or leaf) node ν in $\mathcal{T}$, let $\mathcal{T}(\nu)$ denote the subtree rooted at ν and let $\mathcal{O}(\nu)$ denote the set of objects corresponding to the leaves of $\mathcal{T}(\nu)$. Define $\mathcal{U}(\nu) := \cup_{o_i \in \mathcal{O}(\nu)} o_i^*$. At node ν, we store a point-location data structure [8] on the trapezoidal map of $\mathcal{U}(\nu)$. (If the objects are curved, then the "trapezoids" may have curved top and bottom edges.)

Lemma 9. *The tree $\mathcal{T}$ uses $O(U(n) \log n)$ storage and allows us to report all o_i^* intersecting a vertical edge s of Q in $O((t+1)\log^2 n)$ time, where t is the number of answers.*

Proof. To report all o_i^* intersecting s we walk down $\mathcal{T}$, only visiting the nodes ν such that s intersects $\mathcal{U}(\nu)$. This way we end up in the leaves corresponding to the o_i^* intersecting s. To decide if we have to visit a child ν of an already visited node, we do a point location with both endpoints of s in the trapezoidal map of $\mathcal{U}(\nu)$. Now s intersects $\mathcal{U}(\nu)$ if and only if one of these endpoints lies in a trapezoid inside $\mathcal{U}(\nu)$ and/or the two endpoints lie in different trapezoids. Thus we spend $O(\log n)$ time for the decision. Since we visit $O(k \log n)$ nodes, the total query time is as claimed.

To analyze the storage we claim that the sum of the complexities of $\mathcal{U}(\nu)$ over all nodes ν at any fixed height of $\mathcal{T}$ is $O(U(n))$. The bound on the storage then follows because the point-location data structures take linear space [8] and the height of $\mathcal{T}$ is $O(\log n)$. It remains to prove the claim. Consider a node ν at a given height h in $\mathcal{T}$. Lemma 5 in Appendix A proves that each vertex in $\mathcal{U}(\nu)$ is either a level-0 or level-1 vertex of the arrangement induced by the objects in $\mathcal{O}(\nu)$, or a vertex of o_i^*, for some o_i in $\mathcal{O}(\nu)$. The proof of the claim then follows from the following two facts. First, the number of vertices of the former type is $O(U(|\mathcal{O}(\nu)|))$, which sums to $O(U(n))$ over all nodes at height h. Second, by Lemma 7 the number of vertices of the latter type over all nodes at height h sums to $O(U(n))$. □

Theorem 5. *Let $\mathcal{O}$ be a set of n constant-complexity objects in the plane from a class of objects such that the maximum union complexity of any m objects from the class is $U(m)$. Then there is a data structure that uses $O(U(n)\log n)$ storage and that allows us to report for any axis-aligned query rectangle Q, in $O((k+1)\log^2 n)$ time all pairs of objects o_i, o_j in $\mathcal{O}$ such that o_i intersects o_j inside Q, where k denotes the number of answers.*

4 Concluding remarks

We presented data structures for finding intersecting pairs of objects inside a query rectangle. An obvious open problem is whether our bounds can be improved. In particular, one would hope that better solutions are possible for

3-dimensional boxes, where we obtained $O((k+\sqrt{n})\operatorname{polylog} n)$ query time with $O(n\sqrt{n}\log n)$ storage. (It is possible to reduce the query time in our solution to $O((k+m)\operatorname{polylog} n)$, for any $1 \leqslant m \leqslant \sqrt{n}$, but at the cost of increasing the storage to $O((n^2/m)\operatorname{polylog} n)$.)

Two settings where we have not been able to obtain efficient solutions are when $\mathcal{O}$ is a set of balls in $\mathbb{R}^3$, and when $\mathcal{O}$ is a set of arbitrary segments in the plane. Especially the latter setting seems challenging. Indeed, consider the special case where $\mathcal{O}$ consist of $n/2$ horizontal lines and $n/2$ lines of slope 1. Suppose furthermore that the query is a vertical line ℓ and that we only want to check if ℓ contains at least one intersection. A data structure for this setting could be used to solve the following 3Sum-hard problem: given three sets of parallel lines, decide if there is a triple intersection [10]. Thus it is unlikely that we can obtain a solution with (significantly) sublinear query time and (significantly) subquadratic preprocessing time in the setting just described. However, storage is not the same as preprocessing time. This raises the following question: is it possible to obtain sublinear query time with subquadratic storage?

References

1. P. K. Agarwal, and J. Erickson. Geometric Range Searching and Its Relatives. *Contemporary Mathematics.* 223:1-56 (1999).
2. B. Aronov, M. de. Berg, E. Ezra, and M. Sharir. Improved bounds for the union of locally fat objects in the plane. *SIAM J. Comput.* 43(2):543–572 (2014).
3. M. de. Berg, O. Cheong, M. v. Kreveld, and M. Overmars. *Computational Geometry: Algorithms and Applications (3rd edition).* Springer-Verlag, 2008.
4. B. Chazelle. Filtering search: A new approach to query-answering. *SIAM J. Comput.* 15:703–724 (1986).
5. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.* 17:427–462 (1988).
6. B. Chazelle, H. Edelsbrunner, L.J. Guibas, and M. Sharir. Algorithms for bichromatic line-segment problems and polyhedral terrains. *Algorithmica* 11: 116–132 (1994).
7. K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discr. Comput. Geom.* 4:387–421 (1989).
8. H. Edelsbrunner, L. J. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM J. Comput.* 15:317-340 (1986).
9. H. Edelsbrunner, M. H. Overmars, and R. Seidel. Some methods of computational geometry applied to computer graphics. *Comput. Vision, Graphics and Image Proc.* 28:92–108 (1984).
10. A. Gajentaan and M.H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom. Theory Appl.* 5: 165–185 (1995).
11. J. E. Goodman and J. O'Rourke. Range Searching. Chapter 36 of *Handbook of Discrete and Computational Geometry (2nd edition)*, 2004.
12. K. Keden, R. Livne, J. Pach, and M. Sharir. On the union of Jordan regions and collision-free translational motion amidst polygonal obstacles. *Discr. Comput. Geom.* 1:59-71 (1986).
13. S. Subramanian, and S. Ramaswamy. The P-range tree: A new data structure for range searching in secondary memory. In *Proc. 6th ACM-SIAM Symp. Discr. Alg.*, pages 378–387, 1995.

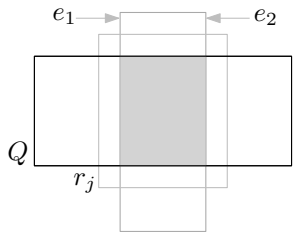


Fig. 4: A possible situation in Case B-3-I.

A Omitted proofs

Lemma 2. *Let r_i, r_j be two rectangles in $\mathcal{O}$ such that $(r_i \cap r_j) \cap Q \neq \emptyset$. Then at least one of r_i, r_j is put into $\mathcal{O}^*(Q)$ by the above query procedure.*

Proof. Let $I := (r_i \cap r_j) \cap Q$. Each edge of I is either contributed by r_i or r_j, or by Q. Let $E(I)$ denote the set of edges of r_i and r_j that contribute an edge to I. We distinguish two cases, with various subcases.

Case A: At least one edge $e \in E(I)$ has an endpoint, v, inside Q. Now the witness sub-edge on e closest to v must intersect Q and, hence, the corresponding rectangle will be put into $\mathcal{O}^*(Q)$ in Step 1(i).

Case B: All edges in $E(I)$ cross Q completely. We now have several subcases.

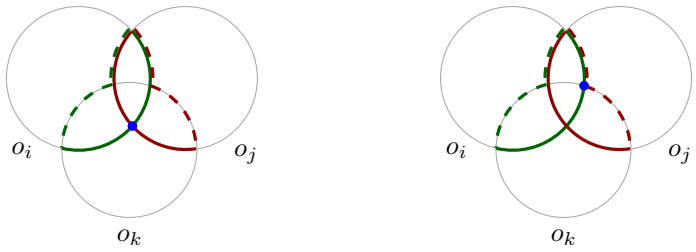
Case B-1: $|E(I)| \leqslant 1$. Now Q contributes at least three edges to I, so at least one corner of I is a corner of Q. Hence, both r_i and r_j are put into $\mathcal{O}^*(Q)$ in Step 1(iii).

Case B-2: $|E(I)| \geqslant 3$. Since each edge of $E(I)$ crosses Q completely and $|E(I)| \geqslant 3$, both $\mathsf{V}(Q)$ and $\mathsf{H}(Q)$ are non-empty. Thus at least one of r_i and r_j is put into $\mathcal{O}^*(Q)$ in Step 1(ii).

Case B-3: $|E(I)| = 2$. Let e_1 and e_2 denote the segments in $E(I)$. If one of e_1, e_2 is vertical and the other is horizontal, we can use the argument from Case B-2. It remains to handle the case where e_1 and e_2 have the same orientation, say vertical.

Case B-3-i: Edges e_1 and e_2 belong to the same rectangle, say r_i, as in Fig. 4. If e_1 has an endpoint, v, inside r_j, then e_1 has a witness sub-edge starting at v that intersects Q, so r_i is put into $\mathcal{O}^*(Q)$ in Step 1(i). If r_j contains a corner of Q then r_j will be put into $\mathcal{O}^*(Q)$ in Step 1(iii). In the remaining case the right edge of r_j crosses Q and there are vertical edges completely crossing r_j (namely e_1 and e_2). Hence, the rightmost edge completely crossing r_j, which is a witness for r_j, intersects Q. Thus r_j is put into $\mathcal{O}^*(Q)$ in Step 1(i).

Case B-3-ii: Edge e_1 is an edge of r_i and e_2 is an edge of r_j (or vice versa). Assume without loss of generality that the y-coordinate of the top endpoint of e_1 is less than or equal to the y-coordinate of the top endpoint of e_2. Then the top endpoint, v, of e_1 must lie in r_j, and so e_1 has a witness sub-edge starting at v that intersects Q. Hence, r_i is put into $\mathcal{O}^*(Q)$ in Step 1(i). □



(a) Case A in the proof of Lemma 5.

(b) Case B in the proof of Lemma 5.

Fig. 5: Different cases in the proof of Lemma 5. To simplify the presentation we assumed the objects are disks. o_i^* and o_j^* are surrounded by dark green and dark red, respectively. Regular arcs are in solid and irregular arcs are in dashed. The blue vertex refers to vertex u in the proof.

Lemma 4. *Using a data structure of size $O(n\sqrt{n}\log n)$ we can find in time $O(\log n \log^* n + k)$ a seed set $\mathcal{O}^*(Q)$ of $O(\sqrt{n}+k)$ boxes containing at least one box from every intersecting pair of Type (ii), where k is the number of such pairs.*

Proof. The Type (ii) intersections $b_i \cap b_j$ either have a vertex that is a vertex of b_i or b_j inside Q, or they have an edge-face pair intersecting inside Q. To find seed objects for the former pairs we used $O(n \log n)$ storage and $O(\log n \log^* n +$ #answers) query time, and we put $O(k)$ boxes into the seed set. For the latter pairs, we used an approach based on clusters. For each cluster F_i we have a data structure $\mathcal{D}(\overline{E_i})$ that uses $O(n \log n)$ storage, giving $O(n\sqrt{n}\log n)$ storage in total. Besides the $O(\sqrt{n})$ boxes in the two clusters F_i and F_j, we put boxes into the seed set for the clusters F_t with $i < t < j$, namely when querying the data structures $\mathcal{D}(\overline{E_i})$. This means that the same box may be put into $\mathcal{O}^*(Q)$ up to $\sqrt{n}$ times. (Note that these duplicates are later removed.) However, each copy we put into the seed set corresponds to a different intersecting pair. Together with the fact that the query time in each $\mathcal{D}(\overline{E_t})$ is $O(\log n \log^* n +$ #answers) this means the total query time and size of the seed set are as claimed. □

Lemma 5. *Each vertex in $\mathcal{U}(\nu)$ is either a level-0 or level-1 vertex of the arrangement induced by the objects in $\mathcal{O}(\nu)$, or a vertex of o_i^*, for some o_i in $\mathcal{O}(\nu)$.*

Proof. Define $\mathcal{O}^*(\nu) := \{o_i^* : o_i \in \mathcal{O}(\nu)\}$. Any vertex u of $\mathcal{U}(\nu)$ that is not a vertex of some $o_i^* \in \mathcal{O}^*(\nu)$ must be an intersection of the boundaries of some $o_i^*, o_j^* \in \mathcal{O}(\nu)$. Note that the boundary ∂o_i^* of an object o_i^* consists of two types of pieces: *regular arcs*, which are parts of the boundary of o_i itself, and *irregular arcs*, which are parts of the boundary of some other object o_k. To bound the number of vertices of $\mathcal{U}(\nu)$ of the form $\partial o_i^* \cap \partial o_j^*$ we now distinguish three cases.

Case A: Intersections between two regular arcs. In this case u is either a level-0 vertex of the arrangement defined by $\mathcal{O}(\nu)$ (namely when u is contained in no other object $o_k \in \mathcal{O}(\nu)$), or a level-1 vertex of that arrangement (when u is contained in a single object $o_k \in \mathcal{O}(\nu)$). Note that u cannot be contained in two objects from $\mathcal{O}(\nu)$, because then u would be in the interior of some $o_k^* \in \mathcal{O}^*(\nu)$, contradicting that u is a vertex of $\mathcal{U}(\nu)$. See Fig 5a.

Case B: Intersections between a regular arc and an irregular arc. Without loss of generality, assume that u is the intersection of a regular arc of ∂o_i^* and an irregular arc of ∂o_j^*. Note that this implies that u lies in the interior of o_j. If there is no other object $o_k \in \mathcal{O}$ containing u then u would be a vertex of o_j^*, and if there is at least one object $o_k \in \mathcal{O}$ containing u then u would not lie on ∂o_j^*. So, under the assumption that u is not already a vertex of o_j^*, Case B does not happen. See Fig 5b.

Case C: Intersections between two irregular arcs. In this case u lies in the interior of both o_i and o_j. But then u should also be in the interior of o_i^* and o_j^*, so this case cannot happen. □