Orthogonal Point Location and Rectangle Stabbing Queries in 3-d

Timothy M. Chan
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
tmc@illinois.edu

Yakov Nekrich
Cheriton School of Computer Science, University of Waterloo, Canada
yakov.nekrich@googlemail.com

Saladi Rahul
Dept. of Computer Science, University of Illinois at Urbana-Champaign, USA
saladi@uiuc.edu

Konstantinos Tsakalidis[1]
Dept. of Computer and Information Science, Tandon School of Engineering, New York University, USA
kt79@nyu.edu

Abstract

In this work, we present a collection of new results on two fundamental problems in geometric data structures: *orthogonal point location* and *rectangle stabbing*.

- **Orthogonal point location.** We give the first linear-space data structure that supports 3-d point location queries on n disjoint axis-aligned boxes with *optimal* $O(\log n)$ query time in the (arithmetic) pointer machine model. This improves the previous $O\left(\log^{3/2} n\right)$ bound of Rahul [SODA 2015]. We similarly obtain the first linear-space data structure in the I/O model with optimal query cost, and also the first linear-space data structure in the word RAM model with sub-logarithmic query time.
- **Rectangle stabbing.** We give the first linear-space data structure that supports 3-d 4-sided and 5-sided rectangle stabbing queries in *optimal* $O(\log_w n + k)$ time in the word RAM model. We similarly obtain the first optimal data structure for the closely related problem of 2-d top-k rectangle stabbing in the word RAM model, and also improved results for 3-d 6-sided rectangle stabbing.

For point location, our solution is simpler than previous methods, and is based on an interesting variant of the van Emde Boas recursion, applied in a round-robin fashion over the dimensions, combined with bit-packing techniques. For rectangle stabbing, our solution is a variant of Alstrup, Brodal, and Rauhe's grid-based recursive technique (FOCS 2000), combined with a number of new ideas.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases geometric data structures, orthogonal point location, rectangle stabbing, pointer machines, I/O model, word RAM model

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.31

Related Version A full version of the paper is available at `https://arxiv.org/abs/1805.08602`.

[1] Partially supported by NSF grants CCF-1319648 and CCF-1533564

45th International Colloquium on Automata, Languages, and Programming (ICALP 2018).
Editors: Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella;
Article No. 31; pp. 31:1–31:14
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

In this work we present a plethora of new results on two fundamental problems in geometric data structures: (a) *orthogonal point location* (where the input rectangle or boxes are non-overlapping), and (b) *rectangle stabbing* (where the input rectangles or boxes are overlapping).

1.1 Orthogonal point location

Point location is among the most central problems in the field of computational geometry, which is covered in textbooks and has countless applications. In this paper we study the *orthogonal point location* problem. Formally, we want to preprocess a set of n *disjoint* axis-aligned boxes (hyperrectangles) in $\mathbb{R}^d$ into a data structure, so that the box in the set containing a given query point (if any) can be reported efficiently. There are two natural versions of this problem, for (a) *arbitrary disjoint boxes* where the input boxes need not fill the entire space, and (b) a *subdivision* where the input boxes fill the entire space.

Arbitrary disjoint boxes. Historically, the point location problem has been studied in the pointer machine model and the main question has been the following:

"Is there a linear-space structure with $O(\log n)$ query time?"

In 2-d this question has been successfully resolved: there exists a linear-space structure with $O(\log n)$ query time [19, 18, 14, 27, 30] (actually this result holds for nonorthogonal point location). In 3-d there has been work on this problem [15, 17, 2, 23], but the question has not yet been resolved. The currently best known result on the pointer machine model is a linear-space structure with $O(\log^{3/2} n)$ query time by Rahul [23]. In this paper,

- we obtain the first linear-space structure with $O(\log n)$ query time for 3-d orthogonal point location for arbitrary disjoint boxes. The structure works in the (arithmetic) pointer machine model and is *optimal* in this model.

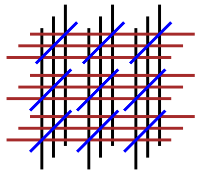
The orthogonal point location problem has been studied in the I/O-model and the word RAM as well (please see the full version for a brief description of these models). In the I/O model, an optimal solution is known in 2-d [16, 6]: a linear-space structure with $O(\log_B n)$ query time, where B is the block size (this result holds for nonorthogonal point location). However, in 3-d the best known result is a linear-space structure with a query cost of $O(\log_B^2 n)$ I/Os by Nekrich [20] (for orthogonal point location for disjoint boxes).

- In the I/O model, we obtain the first linear-space structure with $O(\log_B n)$ query cost for 3-d orthogonal point location for arbitrary disjoint boxes. This result is *optimal*.

In the word RAM model, an optimal solution in 2-d was given by Chan [10] with a query time of $O(\log \log U)$, assuming that input coordinates are in $[U] = \{0, 1, \ldots, U-1\}$. However, in 3-d the best known result for arbitrary disjoint boxes is a linear-space structure with $O(\log n \log \log n)$ query time: this result was not stated explicitly before but can obtained by an interval tree augmented with Chan's 2-d orthogonal point location structure [10] at each node. Our above new result with logarithmic query time is already an improvement even in the word RAM, but we can do slightly better still:

- In the w-bit word RAM model, we obtain the first linear-space structure with *sub-logarithmic* query time for 3-d orthogonal point location for arbitrary disjoint boxes. The time bound is $O(\log_w n)$. (We do not know whether this result is optimal, however.)

Subdivisions. In the plane, the two versions of the problem are equivalent in the sense that any arbitrary set of n disjoint rectangles can be converted into a subdivision of $\Theta(n)$ rectangles via the vertical decomposition. In 3-d, the two versions are no longer equivalent, since there exist sets of n disjoint boxes that need $\Omega\left(n^{3/2}\right)$ boxes to fill the entire space. See figure on the right.



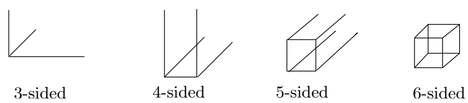
In 3-d the special case of a subdivision is potentially easier than the arbitrary disjoint boxes setting, as the former allows for a fast $O(\log^2 \log U)$ query time in the word RAM model with $O(n \log\log U)$ space, as shown by de Berg, van Kreveld, and Snoeyink [13] (with an improvement by Chan [10]).

- In the word RAM model, we further improve de Berg, van Kreveld, and Snoeyink's method to achieve a *linear*-space structure with $O(\log^2 \log U)$ query time for 3-d orthogonal point location on subdivisions.

1.2 Rectangle stabbing

Rectangle stabbing is a classical problem in geometric data structures [1, 3, 7, 12, 23], which is as old, and as equally natural, as orthogonal range searching—in fact, it can be viewed as an "inverse" of orthogonal range searching, where the input objects are boxes and query objects are points, instead of vice versa. Formally, we want to preprocess a set S of n axis-aligned boxes (possibly overlapping) in $\mathbb{R}^d$ into a data structure, so that the boxes in S containing a given query point q can be reported efficiently. (As one of many possible applications, imagine a dating website, where each lady is interested in gentlemen whose salary is in a range $[S_1,S_2]$ and age is in a range $[A_1,A_2]$; suppose that a gentleman with salary x_q and age y_q wants to identify all ladies who might be potentially interested in him.)

Throughout this paper, we will assume that the endpoints of the rectangles lie on the grid $[2n]^3$ (this can be achieved via a simple rank-space reduction). In the word RAM model, Pǎtraşcu [21] gave a lower bound of $\Omega(\log_w n)$ query time for any data structure which occupies at most $n \log^{O(1)} n$ space to answer the 2-d rectangle stabbing query. Shi and Jaja [29] presented an optimal solution in 2-d which occupies linear space with $O(\log_w n + k)$ query time, where k is the number of rectangles reported.



We introduce some notation to define various types of rectangles in 3-d. (We will use the terms "rectangle" and "box" interchangably throughout the paper.) A rectangle in 3-d is called $(3+t)$-*sided* if it is bounded in t out of the 3 dimensions and unbounded (on one side) in the remaining $3-t$ dimensions.

In the word RAM model, an optimal solution in 3-d is known *only* for the 3-sided rectangle stabbing query: a linear-space structure with $O(\log\log_w n + k)$ query time (by combining the work of Afshani [1] and Chan [10]; this is optimal due to the lower bound of Pǎtraşcu and Thorup [22]). Finding an optimal solution for 4-, 5-, and 6-sided rectangle stabbing has remained open.

3-d 4- and 5-sided rectangle stabbing. Currently, the best-known result for 4-sided and 5-sided rectangle stabbing queries by Rahul [23] occupies $O(n \log^* n)$ space with $O(\log n + k)$ and $O(\log n \log\log n + k)$ query time, respectively. This result holds in the pointer machine model. For 4-sided rectangle stabbing, adapting Rahul's solution to the word RAM model does not lead to any improvement in the query time (the bottleneck is in answering $\log n$

3-d dominance reporting queries). For 5-sided rectangle stabbing, even if we assume the existence of an optimal 4-sided rectangle stabbing structure, plugging it into Rahul's solution can improve the query time to only $O(\log n + k)$, which is still suboptimal. In this paper,

- we obtain the first *optimal* solution for 3-d 4-sided and 5-sided rectangle stabbing in the word RAM model: a linear-space structure with $O(\log_w n + k)$ query time.

2-d top-k rectangle stabbing. Recently, there has been a lot of interest in top-k range searching [4, 8, 9, 24, 25, 26, 28, 31]. Specifically, in the 2-d top-k rectangle stabbing problem, we want to preprocess a set of *weighted* axis-aligned rectangles in 2-d, so that given a query point q and an integer k, the goal is to report the k largest-weight rectangles containing (or stabbed by) q. This problem is closely related to the 5-sided rectangle stabbing problem (by treating the weight as a third dimension, a rectangle r with weight $w(r)$ can be mapped to a 5-sided rectangle $r \times (-\infty, w(r)]$).

- By extending the solution for 3-d 5-sided rectangle stabbing problem, we obtain the first *optimal* solution for the 2-d top-k rectangle stabbing problem: a linear-space structure with $O(\log_w n + k)$ query time.

3-d 6-sided rectangle stabbing. Our new solution to 3-d 5-sided rectangle stabbing, combined with standard interval trees, immediately implies a solution to 3-d 6-sided rectangle stabbing with a query time of $O(\log_w n \cdot \log n + k)$, which is already new. But we can do slightly better still:

- We obtain a linear-space structure with $O(\log_w^2 n + k)$ query time for 3-d 6-sided rectangle stabbing problem in the word RAM model. We conjecture this to be optimal (the analogy is the lower bound of $\Omega(\log^2 n + k)$ query time for linear-space pointer machine structures [3]).

Back to orthogonal point location. Our solution for orthogonal point location uses rectangle stabbing as a subroutine: if there is an $S(n)$-space data structure with $Q(n) + O(k)$ query time to answer the rectangle stabbing problem in $\Re^d$, then one can obtain a data structure for orthogonal point location in $\Re^{d+1}$ with $O(S(n))$-space and $O(Q(n))$ time. By plugging in our new results for 3-d 6-sided rectangle stabbing, we obtain a linear-space word RAM structure which can answer any orthogonal point location query in 4-d in $O(\log_w^2 n)$ time, improving the previously known $O(\log^2 n \log\log n)$ bound [10].

1.3 Our techniques

Our results are obtained using a number of new ideas (in addition to existing data structuring techniques), which we feel are as interesting as the results themselves.

3-d orthogonal point location. To better appreciate our new 3-d orthogonal point location method, we first recall that the current best word-RAM method had $O(\log n \log\log n)$ query time, and was obtained by building an interval tree over the x-coordinates, and at each node of the tree, storing Chan's 2-d point location data structure on the yz-projection of the rectangles. Interval trees caused the query time to increase by a logarithmic factor, while Chan's 2-d structures achieved $O(\log\log n)$ query time via a complicated van-Emde-Boas-like recursion. We can thus summarize this approach loosely by the following recurrence for the query time (superscripts refer to the dimension):

$$Q^{(3)}(n) = O(Q^{(2)}(n) \log n) \text{ and } Q^{(2)}(n) = Q^{(2)}(\sqrt{n}) + O(1)$$
$$\implies Q^{(3)}(n) = O(\log n \log\log n).$$

(Note that naively increasing the fan-out of the interval tree could reduce the query time but would blow up the space usage.)

In the pointer machine model, the current best data structure by Rahul [23], with $O(\log^{3/2} n)$ query time, required an even more complicated combination of interval trees, Clarkson and Shor's random sampling technique, 3-d rectangle stabbing, and 2-d orthogonal point location.

To avoid the extra $\log\log n$ factor, we cannot afford to use Chan's 2-d orthogonal point location structure as a subroutine; and we cannot work with just yz-projections, which intuitively cause loss of efficiency. Instead, we propose a more direct solution based on a new van-Emde-Boas-like recursion, aiming for a new recurrence of the form

$$Q^{(3)}(n) = Q^{(3)}(\sqrt{n}) + O(\log n).$$

The $O(\log n)$ term arises from the need to solve 2-d rectangle stabbing subproblems, on projections along all three directions (the yz-, xz-, and xy-plane), applied in a *round-robin* fashion. The new recurrence then solves to $O(\log n)$—notice how $\log\log$ disappears, unlike the usual van Emde Boas recursion! In the word RAM model, we can even use known sub-logarithmic solutions to 2-d rectangle stabbing to get $O(\log_w n)$ query time.

We emphasize that our new method is much *simpler* than the previous, slower methods, and is essentially self-contained except for the use of a known data structure for 2-d rectangle stabbing emptiness (which reduces to standard 2-d orthogonal range counting).

One remaining issue is space. In our new method, a rectangle is stored $O(\log\log n)$ times, due to the depth of the recursion. To achieve linear space, we need another idea, namely, *bit-packing* tricks, to compress the data structure. Because of the rapid reduction of the universe size in the round-robin van-Emde-Boas recursion, the amortized space in words per input box satisfies a recurrence of the form

$$s(n) = s(\sqrt{n}) + O\left(\frac{\log n}{w}\right) \implies s(n) = O\left(\frac{\log n}{w}\right) = O(1).$$

Our new result on the subdivision case is obtained by a similar space-reduction trick.

3-d 5-sided rectangle stabbing. For 3-d rectangle stabbing, the previous solution by Rahul [23] was based on a grid-based, $\sqrt{n}$-way recursive approach of Alstrup, Brodal, and Rauhe [5], originally designed for 2-d orthogonal range searching. The fact that the approach can be adapted here is nontrivial and interesting, since our input objects are now more complicated (rectangles instead of points) and the target query time is quite different (near logarithmic rather than $\log\log$). More specifically, Rahul first solved the 4-sided case via a complicated data structure, and then applied Alstrup et al.'s technique to reduce 5-sided rectangles to 4-sided rectangles, which led to a query-time recurrence similar to the following (subscripts denote the number of sides, and output cost related to k is ignored):

$$Q_4(n) = O(\log n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_4(n)) \implies Q_5(n) = O(\log n \log\log n).$$

Intuitively, the reduction from the 5-sided to the 4-sided case causes loss of efficiency. To avoid the extra $\log\log n$ factor, we propose a new method that is also based on Alstrup *et* *al.*'s recursive technique, but reduces 5-sided rectangles directly to 3-sided rectangles, aiming for a new recurrence of the form

$$Q_3(n) = O(\log\log_w n) \text{ and } Q_5(n) = 2Q_5(\sqrt{n}) + O(Q_3(n)).$$

During recursion, we do not put 4-sided rectangles in separate structures (which would slow down querying), but instead use a common tree for both 4-sided and 5-sided rectangles. The new recurrence then solves to $Q_5(n) = O(\log_w n)$ with an appropriate base case—notice how $\log\log$ again disappears, and notice how this gives a new result even for the 4-sided case!

One remaining issue is space. Again, we can compress the data structure by incorporating bit-packing tricks (which was also used in Alstrup *et al.*'s original method). For 4- and 5-sided rectangle stabbing, the space recurrence then solves to linear.

However, with space compression, a new issue arises. The cost of reporting each output rectangle in a query increases to $O(\log\log n)$ (the depth of the recursion), because of the need to *decode* the coordinates of a compressed rectangle. In other words, the query cost becomes $O(\log_w n + k\log\log n)$ instead of $O(\log_w n + k)$. This extra decoding overhead also occurred in previous work on 2-d orthogonal range searching by Alstrup *et al.* [5] and Chan *et al.* [11], and it is open how to avoid the overhead for that problem without sacrificing space (this is related to the so-called *ball inheritance problem* [11]).

We observe that for the 4- and 5-sided rectangle stabbing problem, a surprisingly simple idea suffices to avoid the overhead: instead of keeping pointers between consecutive levels of the recursion tree, we just keep pointers directly from each level to the *leaf* level.

3-d 6-sided rectangle stabbing. We can solve 6-sided rectangle stabbing by using our result for 5-sided rectangle stabbing as a subroutine. However, the naive reduction via interval trees increases the query time by a $\log n$ factor instead of $\log_w n$. To speed up querying, the standard idea is to use a tree with a larger fan-out w^ε. This leads to various colored generalizations of 2-d rectangle stabbing with a small number w^ε of colors. Much of our ideas can be extended to solve these colored subproblems in a straightforward way, but a key subproblem, of answering colored 2-d dominance searching queries in $O(\log\log_w n + k)$ time with linear space, is nontrivial. We solve this key subproblem via a clever use of 2-d shallow cuttings, combined with a grouping trick, which may be of independent interest.

2 Orthogonal Point Location in 3-d

Preliminaries. Our solution to 3-d orthogonal point location will require known data structures for *2-d orthogonal point location* and *2-d rectangle stabbing emptiness.* The proofs are presented in the full version.

▶ **Lemma 1.** *Given n disjoint axis-aligned rectangles in $[U]^2$ ($n \le U \le 2^w$), there are data structures for point location with $O\left(\frac{n\log U}{w}\right)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\min\{\log\log U, \log_w n\})$ query time in the word RAM model.*

▶ **Lemma 2.** *Given n (possibly overlapping) axis-aligned rectangles in $[U]^2$ ($n \le U \le 2^w$), there are data structures for rectangle stabbing emptiness with $O\left(\frac{n\log U}{w}\right)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\log_w n)$ query time in the word RAM model.*

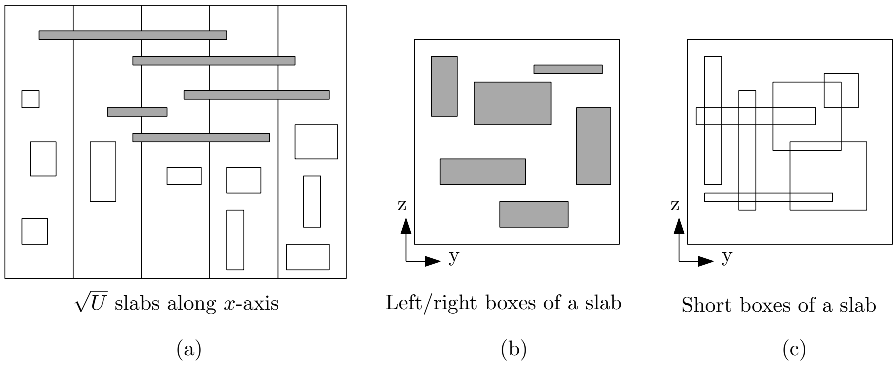


Figure 1 Boxes obtained after partitioning along the x-direction.

Data structure. We are now ready to describe our data structure for 3-d orthogonal point location. We focus on the pointer machine model first. At the beginning, we apply a rank space reduction (replacing input coordinates by their ranks) so that all coordinates are in $[2n]^3$, where n is the global number of input boxes. Given a query point, we can initially find the ranks of its coordinates by three predecessor searches (costing $O(\log n)$ time in the pointer machine model).

We describe our preprocessing algorithm recursively. The input to the preprocessing algorithm is a set of n disjoint boxes that are assumed to be aligned to the $[U_x] \times [U_y] \times [U_z]$ grid. (At the beginning, $U_x = U_y = U_z = 2n$.)

Without loss of generality, assume that $U_x \geq U_y, U_z$. We partition the $[U_x] \times [U_y] \times [U_z]$ grid into $\sqrt{U_x}$ equal-sized vertical slabs perpendicular to the x-direction. See Figure 1. (In the symmetric case $U_y \geq U_x, U_z$ or $U_z \geq U_x, U_y$, we partition along the y- or z-direction instead.) We classify the boxes into two categories:

- *Short boxes.* For each slab, define its short boxes to be those that lie completely inside the slab.
- *Long boxes.* Long boxes intersect the boundary (vertical plane) of at least one slab. Each long box $\mathcal{B}$ is broken into three disjoint boxes:
  - *Left box.* Let s_L be the slab containing the left endpoint (with respect to the x-axis) of $\mathcal{B}$. The left box is defined as $\mathcal{B} \cap s_L$.
  - *Right box.* Let s_R be the slab containing the right endpoint of $\mathcal{B}$. The right box is defined as $\mathcal{B} \cap s_R$.
  - *Middle box.* The remaining portion of box $\mathcal{B}$ after removing its left and right box, i.e. $\mathcal{B} \setminus ((\mathcal{B} \cap s_L) \cup (\mathcal{B} \cap s_R))$.

We build our data structure as follows:

1. *Planar point location structure.* For each slab, we project its left boxes onto the yz-plane. The projected boxes remain disjoint, since they intersect a common boundary. We store them in a data structure for 2-d orthogonal point location by Lemma 1. We do this for the slab's right boxes as well.
2. *Rectangle stabbing structure.* For each slab, we project its short boxes onto the yz-plane. The short boxes are not necessarily disjoint. We store them in a data structure for 2-d rectangle stabbing emptiness by Lemma 2.
3. *Recursive middle structure.* We recursively build a *middle structure* on all the middle boxes.
4. *Recursive short structures.* For each slab, we recursively build a *short structure* on all the short boxes inside the slab.

By translation or scaling, these recursive short structures or middle structure can be made aligned to the $\left[\sqrt{U_x}\right] \times [U_y] \times [U_z]$ grid. In addition, we store the mapping from left/right/middle boxes to their original boxes, as a list of pairs (sorted lexicographically) packed in $O\left(\frac{n \log(U_x U_y U_z)}{w}\right)$ words.

Query algorithm. The following lemma is crucial for deciding whether to query recursively the middle or the short structure.

▶ **Lemma 3.** *Given a query point (q_x, q_y, q_z), if the query with (q_y, q_z) on the rectangle stabbing emptiness structure of the slab that contains q_x returns*

- NON-EMPTY*, then the query point cannot lie inside a box stored in the middle structure, or*
- EMPTY*, then the query point cannot lie inside a box stored in the slab's short structure.*

Proof. If NON-EMPTY is returned, then the query point is stabbed by the extension (along the x-direction) of a box in the slab's short structure and cannot be stabbed by any box stored in the middle structure, because of disjointness of the input boxes. If EMPTY is returned, then obviously the query point cannot lie inside a box stored in the short structure. ◀

To answer a query for a given point (q_x, q_y, q_z), we proceed as follows:

1. Find the slab that contains q_x by predecessor search over the slab boundaries.
2. Query with (q_y, q_z) the planar point location structures at this slab. If a left or a right box returned by the query contains the query point, then we are done.
3. Query with (q_y, q_z) the rectangle stabbing emptiness structure at this slab. If it returns NON-EMPTY, query recursively the slab's short structure, else query recursively the middle structure (after appropriate translation/scaling of the query point).

In step 3, to decode the coordinates of the output box, we need to map from a left/right/middle box to its original box; this can be done naively by another predecessor search in the list of pairs we have stored.

Query time analysis. Let $Q(U_x, U_y, U_z)$ denote the query time for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. Observe that the number of boxes n is trivially upper-bounded by $U_x U_y U_z$ because of disjointness. The predecessor search in step 1, the 2-d point location query in step 2, and the 2-d rectangle stabbing query in step 3 all take $O(\log n) = O(\log(U_x U_y U_z))$ time by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$Q(U_x, U_y, U_z) = Q\left(\sqrt{U_x}, U_y, U_z\right) + O(\log(U_x U_y U_z)).$$

If $U_x = U_y = U_z = U$, then three rounds of recursion will partition along the x-, y-, and z-directions and decrease U_x, U_y, and U_z in a round-robin fashion, yielding

$$Q(U, U, U) = Q\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O(\log U),$$

which solves to $Q(U, U, U) = O(\log U)$. As $U = 2n$ initially, we get $O(\log n)$ query time.

Space analysis. Let $s(U_x, U_y, U_z)$ denote the *amortized* number of words of space needed per input box for our data structure in the $[U_x] \times [U_y] \times [U_z]$ grid. The amortized number of words per input box for the 2-d point location and rectangle stabbing structures is $O\left(\frac{\log(U_x U_y U_z)}{w}\right)$ by Lemmata 1 and 2. We thus obtain the following recurrence, assuming that $U_x \geq U_y, U_z$:

$$s(U_x, U_y, U_z) = s\left(\sqrt{U_x}, U_y, U_z\right) + O\left(\frac{\log(U_x U_y U_z)}{w}\right).$$

Three rounds of recursion yield

$$s(U, U, U) = s\left(\sqrt{U}, \sqrt{U}, \sqrt{U}\right) + O\left(\frac{\log U}{w}\right),$$

which solves to $s(U, U, U) = O\left(\frac{\log U}{w}\right)$. As $U = 2n$ initially, the total space in words is $O\left(n\frac{\log n}{w}\right) \leq O(n)$. Note that the above analysis ignores an overhead of $O(1)$ words of space per node of the recursion tree, but by shortcutting degree-1 nodes, we can bound the number of nodes in the recursion tree by $O(n)$. To summarize, we claim the following results:

▶ **Theorem 4.** *Given n disjoint axis-aligned boxes in 3-d, there are data structures for point location with $O(n)$ words of space and $O(\log n)$ query time in the pointer machine model, $O(\log_B n)$ query cost in the I/O model, and $O(\log_w n)$ query time in the word RAM model.*

Proof. The proof for the I/O model and the word RAM model can be found in the full version. ◀

Further applications of this framework to subdivisions, 4-d and higher dimensions can be found in the full version.

3 Rectangle Stabbing

3.1 Preliminaries

▶ **Lemma 5.** *(Rahul [23]) There is a data structure of size $O(n)$ words which can answer a 5-sided 3-d rectangle stabbing query in $O(\log^2 n \cdot \log\log n + k)$ time.*

▶ **Lemma 6.** *(Leaf structure.) For a set of size $O(w^{1/4})$, there is a data structure of size $O(w^{1/4})$ words which can answer a 5-sided 3-d rectangle stabbing query in $O(1 + k)$ time.*

3.2 3-d 5-sided rectangle stabbing

Skeleton of the structure. Consider the projection of the rectangles of S on to the xy-plane and impose an orthogonal $\left[\!\left[2\sqrt{\frac{n}{\log^4 n}}\right]\!\right] \times \left[\!\left[2\sqrt{\frac{n}{\log^4 n}}\right]\!\right]$ grid such that each horizontal and vertical slab contains the projections of $\sqrt{n\log^4 n}$ sides of S. This grid is the root node of our tree $\mathcal{T}$. For each vertical and horizontal slab, we recurse on the rectangles of S which are *sent* to that slab. At each node of the recursion tree, if we have m rectangles in the subproblem, the grid size changes to $\left[\!\left[2\sqrt{\frac{m}{\log^4 m}}\right]\!\right] \times \left[\!\left[2\sqrt{\frac{m}{\log^4 m}}\right]\!\right]$. We stop the recursion when a node has less than $w^{1/4}$ rectangles.

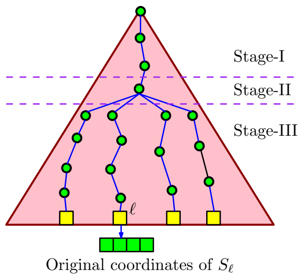


Figure 2

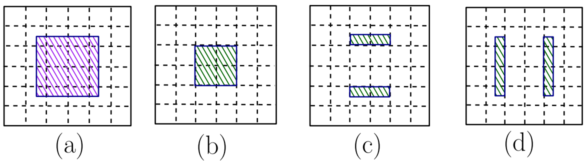


Figure 3

Breaking the rectangles. The solution of Rahul [23] *breaks* only one side to reduce 5-sided rectangles to 4-sided rectangles, and then uses the solution for 4-sided rectangle stabbing as a black box. Unlike the approach of Rahul [23], we will break two sides of each 5-sided rectangle to obtain $O(\log\log n)$ 3-sided rectangles.

For a node in the tree, the intersection of every pair of horizontal and vertical grid line defines a *grid point*. A rectangle $r \in S$ is associated with four root-to-leaf paths (as shown in Figure 2). Any node (say, v) on these four paths is classified w.r.t. r into one of the three stages as follows:

Stage-I. The xy-projection of r intersects none of the grid points. Then r is not stored at v, and sent to the child corresponding to the row or column r lies in.

Stage-II. The xy-projection of r intersects at least one of the grid points. Then r is broken into at most five disjoint pieces. The first piece is a *grid rectangle*, which is the bounding box of all the grid points lying inside r, as shown in Figure 3(b). The remaining four pieces are two *column rectangles* and two *row rectangles* as shown in Figure 3(c) and (d), respectively. The grid rectangle is stored at v. Note that each column rectangle (resp., row rectangle) is now a 4-sided rectangle in $\mathbb{R}^3$ w.r.t. its column (resp., row), and is sent to its corresponding child node.

Stage-III. The xy-projection of a 4-sided piece of r intersects at least one of the grid points. Without loss of generality, assume that the 4-sided rectangle r is unbounded along the negative x-axis. Then the rectangle is broken into at most four disjoint pieces: a *grid rectangle*, two *row rectangles*, and a *column rectangle*, as shown in Figure 4(b), (c) and (d), respectively. The grid rectangle and the two row rectangles are stored at v, and the column rectangle is sent to its corresponding child node. Note that the two row rectangles are now 3-sided rectangles in $\mathbb{R}^3$ w.r.t. their corresponding rows (unbounded in one direction along x-, y- and z-axis).

Encoding structures. Let S_v be the set of rectangles stored at a node v in the tree. We apply a rank space reduction (replacing input coordinates by their ranks) so that the coordinates of all the endpoints are in $[2|S_v|]^3$. If v is a leaf node, then we build an instance of Lemma 6. Otherwise, the following three structures will be built using S_v:

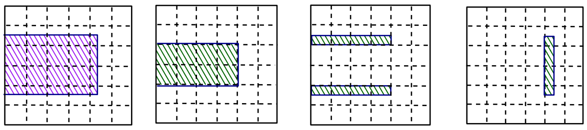


Figure 4

(A) Slow structure. An instance of Lemma 5 is built on S_v to answer the 3-d 5-sided rectangle stabbing query when the output size is "large".
(B) Grid structure. For each cell c of the grid, among the rectangles which completely cover c, pick the $\log^3 |S_v|$ rectangles with the largest span along the z-direction. Store them in a list $Top(c)$ in decreasing order of their span.
(C) 3-d dominance structure. For a given row or column in the grid, based on the 3-sided rectangles stored in it, a linear-space 3-d dominance reporting structure [1, 10] is built. This structure is built for each row and column slab.

Where are the original coordinates stored? Unlike the previous approaches for indexing points [5, 11], we use a somewhat unusual approach for storing the original coordinates of each rectangle. In the process of breaking each 5-sided rectangle described above, there will be four leaf nodes where portions of the rectangle will get stored. We will choose these leaf nodes to store the original coordinates of the rectangle (see Figure 2). The benefit is that each 3-sided rectangle (stored at a node v) has to maintain a decoding pointer of length merely $O(\log |S_v|)$ to point to its original coordinates stored in its subtree.

Query algorithm and analysis. Given a query point q, we start at the root node and perform the following steps: First, query the dominance structure corresponding to the horizontal and the vertical slab containing q. Next, for the grid structure, locate the cell c on the grid containing q. Scan the list $Top(c)$ to keep reporting till (a) all the rectangles have been exhausted, or (b) a rectangle not containing q is found. If case (a) happens and $|Top(c)| = \log^3 |S_v|$, then we discard the rectangles reported till now, and query the slow structure. The decoding pointers will be used to report the original coordinates of the rectangles. Finally, we recurse on the horizontal and the vertical slab containing q. If we visit a leaf node, then we query the leaf structure (Lemma 6).

First, we analyze the space. Let $s(|S_v|)$ be the *amortized* number of bits needed per input 5-sided rectangle in the subtree of a node v. The amortized number of bits needed per rectangle for the encoding structures and the pointers to the original coordinates is $O(\log |S_v|)$. This leads to the following recurrence:

$$s(n) = s(\sqrt{n \log^4 n}) + O(\log n)$$

which solves to $s(n) = O(\log n)$ bits. Therefore, the overall space is bounded by $O(n)$ words.

Next, we analyze the query time. To simplify the analysis, we will exclude the output size term while mentioning the query time. At the root, the time taken to query the grid and the dominance structure is $O(\log \log_w n)$. This leads to the following recurrence:

$$Q(n) = 2Q(\sqrt{n \log^4 n}) + O(\log \log_w n)$$

with a base case of $Q(w^{1/4}) = O(1)$. This solves to $Q(n) = O(\log_w n - \log \log_w n) = O(\log_w n)$. For each reported rectangle it takes constant time to recover its original coordinates. The time taken to query the slow structure is dominated by the output size. Therefore, the overall query time is $O(\log_w n + k)$.

▶ **Theorem 7.** *There is a data structure of size $O(n)$ words which can answer any 3-d 5-sided rectangle stabbing query in $O(\log_w n + k)$ time. This is optimal in the word RAM model.*

Our solution for 2-d top-k rectangle stabbing can be found in the full version.

3.3 3-d 6-sided rectangle stabbing

The complete discussion on 6-sided rectangle stabbing can be found in the full version. Here we will only highlight the key result.

▶ **Lemma 8.** *There exists an optimal linear-space data structure that answers z-restricted 3-d 4-sided rectangle stabbing queries in $O(\log\log_w n + k)$ time. A z-restricted 4-sided rectangle is of the form $(-\infty, x] \times (-\infty, y] \times [i, j]$, where integers $i, j \in [w^\varepsilon]$ and $\varepsilon = 0.1$.*

Proof. We can safely assume that $n > w^{2\varepsilon} \cdot \log w \log\log n$, because the case of $n < w^{2\varepsilon} \cdot \log w \log\log n = O(w^{1/4})$ can be handled in $O(1 + k)$ time by using the structure of Lemma 6. To keep the discussion short, we will assume that $k < \log w \cdot \log\log n$ (handling small values of k is typically more challenging).

Shallow cuttings. A point p_1 is said to *dominate* point p_2 if it has a larger x-coordinate and a larger y-coordinate value. Our main tool to handle this case are *shallow cuttings* which have the following three properties: (a) A t-shallow cutting for a set P of 2-d points is a union of $O(n/t)$ cells where every cell is of the form $[a, +\infty) \times [b, +\infty)$, (b) every point that is dominated by at most t points from P will lie within some cell(s), and (c) each cell contains at most $O(t)$ points of P. A cell $[a, +\infty) \times [b, +\infty)$ can be identified by its corner (a, b). We denote by $Dom(c)$ the set of points that dominate the corner c.

Data structure. We classify rectangles according to their z-projections. The set S_{ij} contains all rectangles of the form $r = (-\infty, x_f] \times (-\infty, y_f] \times [i, j]$. Since $1 \le i \le j \le w^\varepsilon$, there are $O(w^{2\varepsilon})$ sets S_{ij}. Every rectangle r in S_{ij} is associated with a point $p(r) = (x_f, y_f)$. We construct a t-shallow cutting L_{ij} with $t = \log w \cdot \log\log n$ for the set of points $p(r)$, such that $r \in S_{ij}$. A rectangle $r = (-\infty, x_f] \times (-\infty, y_f] \times [i, j]$ is stabbed by a query point $q = (q_x, q_y, q_z)$ if and only if $p(r) \in S_{ij}$ and the point $p(r)$ dominates the 2-d point (q_x, q_y). We can find points of a set S_{ij} that dominate q using the shallow cutting L_{ij}. However, to answer the stabbing query we must simultaneously answer a dominance query on $O(w^{2\varepsilon})$ different sets of points.

We address this problem by *grouping* corners of different shallow cuttings into one structure. Let $\mathcal{C}_{ij}$ denote the set of corners in a shallow cutting L_{ij} and let $\mathcal{C} = \bigcup_{\forall i,j \in [w^\varepsilon]} \mathcal{C}_{ij}$. The set $\mathcal{C}$ is divided into disjoint groups, so that every group G_α consists of $w^{2\varepsilon}$ consecutive corners (with respect to their x-coordinates): for any $c \in G_\alpha$ and $c' \in G_{\alpha+1}$, $c.x < c'.x$. We say that a corner $c \in \mathcal{C}_{ij}$ is immediately to the left of G_α if it is the rightmost corner in $\mathcal{C}_{ij}$ such that $c_x \le c'_x$ for any corner $c' = (c'_x, c'_y)$ in G_α. The set of corners $\overline{G}_\alpha$ contains (1) all corners from G_α, and (2) for every pair i, j such that $1 \le i \le j \le w^\varepsilon$, the corner $c \in \mathcal{C}_{ij}$ immediately to the left of G_α. The set R_α contains all rectangles r such that $p(r) \in Dom(c)$ for each corner $c \in \overline{G}_\alpha$. Since R_α contains $O(w^{2\varepsilon} \log w \cdot \log\log n) = O(w^{1/6})$ rectangles, we can perform a rank-space reduction and answer queries on R_α in $O(k + 1)$ time by using Lemma 6.

Next, we will show that the space occupied by this structure is $O(n)$. The crucial observation is that the number of corners in G_α is $w^{2\varepsilon}$ and the number of "immediately left" corners added to each $\overline{G}_\alpha$ is also bounded by $w^{2\varepsilon}$. The number of corners in set $\mathcal{C}$ is bounded by $\sum_{\forall L_{i,j}} O\left(1 + \frac{|S_{ij}|}{t}\right) = O(n/t)$, since $n/t > w^{2\varepsilon}$. Therefore, the number of groups will be $O\left(\frac{n}{tw^{2\varepsilon}}\right)$. Each set R_α contains $O(w^{2\varepsilon}t)$ rectangles. Therefore, the total space occupied by this structure is $\sum_{\forall\alpha} |R_\alpha| = O(\frac{n}{tw^{2\varepsilon}} \cdot w^{2\varepsilon}t) = O(n)$.

Query algorithm. Given a query point $q = (q_x, q_y, q_z)$, we find the set G_α that "contains" q_x. Then we report all the rectangles in R_α that are stabbed by q by using Lemma 6. We need $O(\log\log_w n)$ time to find the group G_α [22] and, then $O(1+k)$ time to report $R_\alpha \cap q$. ◀

References

1 Peyman Afshani. On dominance reporting in 3D. In *Proceedings of European Symposium on Algorithms (ESA)*, pages 41–51, 2008.

2 Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 240–246, 2010.

3 Peyman Afshani, Lars Arge, and Kasper Green Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 323–332, 2012.

4 Peyman Afshani, Gerth Stølting Brodal, and Norbert Zeh. Ordered and unordered top-k range reporting in large data sets. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 390–400, 2011.

5 Stephen Alstrup, Gerth Stølting Brodal, and Theis Rauhe. New data structures for orthogonal range searching. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 198–207, 2000.

6 Lars Arge, Andrew Danner, and Sha-Mayn Teh. I/O-efficient point location using persistent B-trees. *ACM Journal of Experimental Algorithmics*, 8, 2003.

7 J.L. Bentley. Solutions to Klee's rectangle problems. *Technical Report, Carnegie-Mellon University, Pittsburgh, PA*, 1977.

8 Gerth Stølting Brodal. External memory three-sided range reporting and top-k queries with sublogarithmic updates. In *Proceedings of Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 47, pages 23:1–23:14. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

9 Gerth Stølting Brodal, Rolf Fagerberg, Mark Greve, and Alejandro Lopez-Ortiz. Online sorted range reporting. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 173–182, 2009.

10 Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Transactions on Algorithms*, 9(3):22, 2013.

11 Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 1–10, 2011.

12 Bernard Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal of Computing*, 15(3):703–724, 1986.

13 Mark de Berg, Marc J. van Kreveld, and Jack Snoeyink. Two- and three-dimensional point location in rectangular subdivisions. *Journal of Algorithms*, 18(2):256–277, 1995.

14 Herbert Edelsbrunner, Leonidas J. Guibas, and Jorge Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal of Computing*, 15(2):317–340, 1986.

15 Herbert Edelsbrunner, G. Haring, and D. Hilbert. Rectangular point location in d dimensions with applications. *Comput. J.*, 29(1):76–82, 1986.
16 Michael T. Goodrich, Jyh-Jong Tsay, Darren Erik Vengroff, and Jeffrey Scott Vitter. External-memory computational geometry. In *Proceedings of Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 1993.
17 John Iacono and Stefan Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, 2000.
18 David G. Kirkpatrick. Optimal search in planar subdivisions. *SIAM Journal of Computing*, 12(1):28–35, 1983.
19 Richard J. Lipton and Robert Endre Tarjan. Applications of a planar separator theorem. *SIAM Journal of Computing*, 9(3):615–627, 1980.
20 Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.
21 Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal of Computing*, 40(3):827–847, 2011.
22 Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of ACM Symposium on Theory of Computing (STOC)*, pages 232–240, 2006.
23 Saladi Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 200–211, 2015.
24 Saladi Rahul and Ravi Janardan. A general technique for top-k geometric intersection query problems. *IEEE Transactions on Knowledge and Data Engineering (TKDE)*, 26(12):2859–2871, 2014.
25 Saladi Rahul and Yufei Tao. On top-k range reporting in 2d space. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 265–275, 2015.
26 Saladi Rahul and Yufei Tao. Efficient top-k indexing via general reductions. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2016.
27 Neil Sarnak and Robert Endre Tarjan. Planar point location using persistent search trees. *Communications of the ACM (CACM)*, 29(7):669–679, 1986.
28 Cheng Sheng and Yufei Tao. Dynamic top-k range reporting in external memory. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, 2012.
29 Qingmin Shi and Joseph JáJá. Novel transformation techniques using q-heaps with applications to computational geometry. *SIAM Journal of Computing*, 34(6):1474–1492, 2005.
30 Jack Snoeyink. Point location. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, pages 767–787. CRC Press, 2nd edition, 2004.
31 Yufei Tao. A dynamic I/O-efficient structure for one-dimensional top-k range reporting. In *Proceedings of ACM Symposium on Principles of Database Systems (PODS)*, pages 256–265, 2014.