Improved Bounds for Orthogonal Point Enclosure Query and Point Location in Orthogonal Subdivisions in $\mathbb{R}^3$ [*]

Saladi Rahul[†]

Abstract

In this paper, new results for two fundamental problems in the field of computational geometry are presented: orthogonal point enclosure query ($OPEQ$) in $\mathbb{R}^3$ and point location in orthogonal subdivisions in $\mathbb{R}^3$. All the results are in the pointer machine model of computation.

(1) In an orthogonal point enclosure query, a set S of n axes-parallel rectangles in $\mathbb{R}^3$ is to be preprocessed, so that given a query point $q \in \mathbb{R}^3$, one can efficiently report all the rectangles in S containing (or stabbed by) q. When rectangles are 3-sided (of the form $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$), there exists an optimal solution of Afshani (ESA'08) which uses $O(n)$ space and answers the query in $O(\log n + k)$ time, where k is the number of rectangles reported. Unfortunately, when the rectangles are 4-sided (of the form $[x_1, x_2] \times (-\infty, y] \times (-\infty, z]$), the best result one can achieve using existing techniques is $O(n)$ space and $O(\log^2 n + k)$ query time.

The key result of this work is an almost optimal solution for 4-sided rectangles. The first data structure uses $O(n \log^* n)$ space and answers the query in $O(\log n + k)$ time. Here $\log^* n$ is the iterated logarithm of n. The second data structure uses $O(n)$ space and answers the query in $O(\log n \cdot \log^{(i)} n + k)$ time, for any constant integer $i \geq 1$. Here $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ when $i > 1$.

To handle $OPEQ$ for general 6-sided rectangles (of the form $[x_1, x_2] \times [y_1, y_2] \times [z_1, z_2]$), existing structures in the literature occupy $\Omega(n \log n)$ space. This work presents the first known near-linear space data structure. It occupies $O(n \log^* n)$ space and answer the query in $O(\log^2 n \cdot \log\log n + k)$ time. This is almost optimal, since Afshani, Arge and Larsen (SoCG'10 and SoCG'12) proved that with $O(n)$ space, $OPEQ$ on 6-sided rectangles takes $\Omega(\log^2 n + k)$ time.

(2) In point location in orthogonal subdivisions, a set S of n non-overlapping axes-parallel rectangles in $\mathbb{R}^d (d \geq 3)$ is to be preprocessed, so that given a query point $q \in \mathbb{R}^d$, one can efficiently report the rectangle in S containing q. In a pointer machine model, the first known solution by Edelsbrunner, Haring and Hilbert in 1986 uses $O(n)$ space and answers the query in $O(\log^{d-1} n)$ query time. After a long gap, Afshani, Arge and Larsen (SoCG'10) improved the query time to $O(\log n (\log n / \log\log n)^{d-2})$. This work presents a data structure which occupies $O(n)$ space and answers the query in $O(\log^{d-3/2} n)$ time, improving the previously best known query time by roughly a $\sqrt{\log n}$ factor.

1 Introduction

Orthogonal point enclosure query (OPEQ) and *point location in orthogonal subdivisions* are two fundamental problems in the field of computational geometry. They have been well studied from the early days of computational geometry. In this work, we present improved results for both the problems in $\mathbb{R}^3$ in the pointer machine model. For details of this model, please see Appendix A.

In orthogonal point enclosure query, we preprocess a set S of n axes-parallel rectangles in $\mathbb{R}^d$, so that given a query point $q \in \mathbb{R}^d$, we can efficiently report all the rectangles in S containing (or stabbed by) q. There are a lot of practical applications of this query in various domains such as GIS, recommender systems, networking etc. For example, on a flight booking website such as *kayak.com*, all the users can specify their preference as a d-dimensional rectangle: "I am looking for flights with price in the range \$100 to \$300 and with departure date in the range 1st March to 5th March". Here price is the x-axis and departure date is the y-axis. Given a particular flight, all the users whose preference match this flight can be found out by posing an $OPEQ$ with q *(price, departure date)* $\in \mathbb{R}^2$ as the query point. In point location in orthogonal subdivisions, we preprocess a set S of n non-overlapping axes-parallel rectangles in $\mathbb{R}^d$, so that given a query point $q \in \mathbb{R}^d$, we can efficiently report the rectangle in S containing q.

1.1 Previous results

Orthogonal point enclosure query (OPEQ): There are several ways of obtaining an optimal solution of $O(n)$ space and $O(\log n + k)$ query

[*]This research was partly supported by a Doctoral Dissertation Fellowship (DDF) from the Graduate School of University of Minnesota.

[†]Department of Computer Science and Engineering, University of Minnesota, Twin Cities *sala0198@umn.edu*

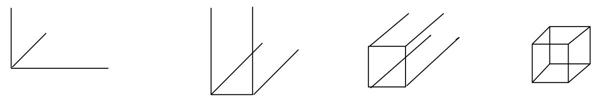

Figure 1: Different kinds of rectangles in three dimensional space.

time in $\mathbb{R}^1$, where k is the number of intervals reported [15, 18, 22]. In $\mathbb{R}^2$ an optimal solution of $O(n)$ space and $O(\log n + k)$ query time was obtained by Chazelle [12]. He introduced the hive-graph structure to answer the query. Later, another solution with optimal bounds was presented in [8] using a combination of persistence and interval tree.

By using segment trees [15, 22], we can generalize the optimal structure in $\mathbb{R}^2$ to higher dimensions. In $\mathbb{R}^d$ the space occupied will be $O(n \log^{d-2} n)$ and the query time will be $O(\log^{d-1} n + k)$. Afshani *et al.* [3] extend the above result for any parameter $h \geq 2$, to obtain an $O(nh \log^{d-2} n)$ space and $O(\log n \cdot (\log n / \log h)^{d-2} + k)$ query time solution. However, these structures occupy $\Omega(n \log n)$ space in $\mathbb{R}^3$. A natural question arises if an $O(n)$-space and $O(\log n + k)$-query time solution can be obtained in $\mathbb{R}^3$? The answer unfortunately is, no. Afshani *et al.* [2, 3] showed that with $O(n)$ space, the $OPEQ$ takes $\Omega(\log^2 n + k)$ time.

Point location in orthogonal subdivisions: There has been extensive work done on point location for general subdivisions. We refer the reader to the book chapter of Snoeyink [24] for a detailed survey on point location for general subdivisions. In this work, we only mention the previous results for orthogonal subdivisions. In a pointer machine model, the first known solution was by Edelsbrunner, Haring and Hilbert [19]. In $\mathbb{R}^d$ their solution uses $O(n)$ space and answers the query in $O(\log^{d-1} n)$ time. Later, Afshani, Arge and Larsen [2] improved the query time to $O(\log n (\log n / \log\log n)^{d-2})$.

De Berg, van Kreveld and Snoeyink [16] solved the problem in the word RAM model, while Nekrich [23] solved the problem in the I/O-model. Recently, these results have been improved in $\mathbb{R}^2$ by Chan [9]. There has also been recent work by Chan and Lee [11] on improving the constant factors hidden in the big-Oh bounds on the number of comparisons needed to perform point location in orthogonal subdivisions in $\mathbb{R}^3$.

1.2 Our results and techniques

Orthogonal point enclosure query in $\mathbb{R}^3$: We first introduce some notation to denote special kinds of rectangles in $\mathbb{R}^3$. A rectangle is called $(3+k)$-sided if the rectangle is bounded in k out of the 3 dimensions and unbounded (on one side) in the remaining $3-k$ dimensions. See Figure 1 for a 3-, 4-, 5- and 6-sided rectangle in $\mathbb{R}^3$. When rectangles are 3-sided, the $OPEQ$ can be answered in $O(\log n + k)$ query time and by using $O(n)$ space [1, 21]. However, when the rectangles are 4-sided, the best result one can achieve using existing techniques is $O(n)$ space and $O(\log^2 n + k)$ query time (see Theorem 2.1).

The key result of our work is an almost optimal solution for 4-sided rectangles. Our first data structure uses $O(n \log^* n)$ space and answers the query in $O(\log n + k)$ time. Our second data structure uses $O(n)$ space and answers the query in $O(\log n \cdot \log^{(i)} n + k)$ time, for any constant integer $i \geq 1$. At a high-level, the following are the key ideas:

(a) As will be shown later, an $OPEQ$ for 4-sided rectangles can be answered by asking $O(\log n)$ $OPEQs$ on 3-sided rectangles. By carefully applying the idea of shallow cuttings, we succeed in answering each of the $OPEQ$ on 3-sided rectangles in "effectively" $O(\log^* n)$ time (ignoring the output term). The trick is to identify the most "fruitful" shallow cutting to answer each of the $OPEQ$ on 3-sided rectangles; this is achieved by reducing $O(\log n)$ point location queries to the problem of $OPEQ$ in $\mathbb{R}^2$. We believe this is a novel idea. This leads to a solution which takes $O(n \log^* n)$ space and $O(\log n \cdot \log^* n + k)$ query time (see Theorem 3.1).

(b) To further reduce the query time to $O(\log n + k)$, our next idea is to increase the fanout of our base tree. This leads to breaking down each 4-sided rectangle into two *side rectangles* and one *middle rectangle*. As will become clear later, the decrease in height of the base tree implies that the side rectangles can now be reported in $O(\log n + k)$ time, instead of $O(\log n \cdot \log^* n + k)$ time. Handling the middle rectangles is the new challenge which arises. We build a structure so that the query on middle rectangles can be handled by asking $O(\log n)$ *2d-dominance reporting queries*. Another novel and new idea in this work is to build a structure which can efficiently identify the $O(\log n)$ data structures on which to pose these $2d$-dominance reporting queries. Also, we are able to answer each $2d$-dominance reporting query in $O(1)$ time (ignoring the output term). This finally leads to a data structure for answering $OPEQ$ on 4-sided rectangles in $O(\log n + k)$ query time and uses $O(n \log^* n)$ space (see Theorem 4.1).

The result obtained for 4-sided rectangles acts as a building block to answer the $OPEQ$ in $\mathbb{R}^3$ for 5-sided rectangles using $O(n \log^* n)$ space and $O(\log n \cdot \log\log n + k)$ query time. This allows us to finally answer $OPEQ$ in $\mathbb{R}^3$ for 6-sided rectangles. See Table 1 for a

Query	Space	Query Time	Notes
4-sided	$O(n)$	$O(\log^2 n + k)$	[1] + Interval tree
4-sided	$O(n \log^* n)$	$O(\log n + k)$	New
4-sided	$O(n)$	$O(\log n \cdot \log^{(i)} n + k)$	New
5-sided	$O(n)$	$O(\log^3 n + k)$	[1] + Interval tree
5-sided	$O(n \log^* n)$	$O(\log n \cdot \log\log n + k)$	New
6-sided	nh	$\Omega(\log^2 n/\log h + k)$	[2, 3]
6-sided	$O(n)$	$O(\log^4 n + k)$	[1] + Interval tree
6-sided	$O(n \log^* n)$	$O(\log^2 n \cdot \log\log n + k)$	New

Table 1: Summary of our results for orthogonal point enclosure in $\mathbb{R}^3$. $\log^* n$ is the iterated logarithm of n. $\log^{(1)} n = \log n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$ when $i > 1$ is a constant integer. Existing solutions in the literature for 6-sided rectangles require $\Omega(n \log n)$ space.

comparison of our results with the currently best known results. Note that we are only interested in structures which occupy linear or near-linear space.

Point location in orthogonal subdivisions in $\mathbb{R}^3$: Traditional techniques such as a separator-based approach [16] and a sampling-based approach [9] have been effective in handling point location in orthogonal subdivisions in $\mathbb{R}^2$. These techniques heavily rely on the fact that, given a set of k non-overlapping rectangles in $\mathbb{R}^2$, the "holes" in the plane (i.e. $\mathbb{R}^2$) can be filled using $O(k)$ rectangles. This is no longer true in $\mathbb{R}^3$, as k non-overlapping rectangles can create $\Omega(k^{3/2})$ holes. To overcome this issue, our key idea is to use an *interval tree* (discussed later) to project the rectangles onto a two-dimensional space and then apply sampling-based techniques on these two-dimensional rectangles. In the process we also reduce the problem to *OPEQ* on sub-linear number of 6-sided rectangles (an idea borrowed from Chan *et al.* [10] and Afshani *et al.* [3]). We finally obtain a solution which uses $O(n)$ space and answers the query in $O(\log^{d-3/2} n)$ time in $\mathbb{R}^d$. This improves the previously best known query time by roughly a $\sqrt{\log n}$ factor.

2 Simple structure for OPEQ using linear space

In this section, we present a simple but sub-optimal structure to answer *OPEQ* on 3, 4, 5, 6-sided rectangles. Handling *OPEQ* on 3-sided rectangles is easy. Map each 3-sided rectangle $(-\infty, x] \times (-\infty, y] \times (-\infty, z]$ into a three-dimensional point (x, y, z) and map the query point $q(q_x, q_y, q_z)$ into a 3-sided query rectangle $q' = [q_x, \infty) \times [q_y, \infty) \times [q_z, \infty)$. Therefore, the problem maps to the *three-dimensional dominance reporting query*: Report all the points lying inside the 3-sided query rectangle q'. Initially, Afshani [1] and recently, Makris and Tsakalidis [21] presented an optimal solution for three-dimensional dominance query ($O(n)$ space and $O(\log n + k)$ query time).

Now we present a solution to handle *OPEQ* on 4, 5, 6-sided rectangles. First, build an interval tree IT based on the x-projection of the rectangles of S. Please refer to Appendix B for the description of an interval tree. We make the following observation to build secondary structures at each node of the interval tree.

Observation 1. *Let S_v be the set of $(4+t)$-sided rectangles (where $t \in [0, 2]$) whose corresponding x-projection gets stored at node v. Consider a rectangle $r = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \in S_v$.*

1. *Suppose the query point $q(q_x, q_y, q_z)$ lies to the left of split(v), i.e., $q_x <= split(v)$. Then r contains q iff $q_x \in [x_1, \infty)$, $q_y \in [y_1, y_2]$ and $q_z \in [z_1, z_2]$.*
2. *Suppose the query point $q(q_x, q_y, q_z)$ lies to the right of split(v), i.e., $q_x > split(v)$. Then r contains q iff $q_x \in (-\infty, x_2]$, $q_y \in [y_1, y_2]$ and $q_z \in [z_1, z_2]$.*

Consider a node $v \in IT$. To handle the case where the query point q lies to the right of *split*(v), we build a structure IT_v^r: Each $(4+t)$-sided rectangle $r = [x_1, x_2] \times [y_1, y_2] \times [z_1, z_2] \in S_v$ is mapped into a $(4+t-1)$-sided rectangle $(-\infty, x_2] \times [y_1, y_2] \times [z_1, z_2]$. Using Observation 1, based on these newly mapped $(d+t-1)$-rectangles we build a structure to handle *OPEQ*. A similar structure IT_v^l is built to handle the case where the query point q lies to the left of *split*(v). Given a query point $q \in \mathbb{R}^3$, we visit a path from root to leaf node in IT containing q_x. At each node v in the path, depending on whether q is to the left or right of v we issue an *OPEQ* on IT_v^l or IT_v^r, respectively, to report the rectangles in $S_v \cap q$.

Theorem 2.1. *OPEQ on 4, 5, 6-sided rectangles can be answered using a structure of $O(n)$ size and in $O(\log^2 n + k)$, $O(\log^3 n + k)$ and $O(\log^4 n + k)$ query time, respectively.*

Proof. When $t = 0$ ($OPEQ$ on 4-sided rectangles), the secondary structures IT_v^l and IT_v^r will be the $OPEQ$ structure for 3-sided rectangles. Clearly, the space occupied by the entire structure will be $O(n)$. For a given query, at most $O(\log n)$ nodes in IT are visited and hence, the query time will be $O(\log^2 n + k)$. Now, it can be easily seen that when $t = 1$ and $t = 2$, the space remains $O(n)$ but the query time increases to $O(\log^3 n + k)$ and $O(\log^4 n + k)$, respectively.

3 OPEQ for 4-sided rectangles: almost optimal query time

In this section we present a proof for the following result.

Theorem 3.1. *Orthogonal point enclosure query on 4-sided rectangles can be answered using a structure of $O(n \log^* n)$ size and in $O(\log n \cdot \log^* n + k)$ query time.*

3.1 Shallow cuttings Given two points p and q in $\mathbb{R}^d$, we say p dominates q if p has a larger coordinate value than q in every dimension. Let P be a set of n three-dimensional points. A shallow cutting for the t-level of P gives a point set R with the following properties: (i) $|R| = O(n/t)$, (ii) Any 3-d point p that is dominated by at most t points of P dominates a point in R, (iii) Each point in R is dominated by $O(t)$ points of P. The existence of such shallow cuttings has been shown by Afshani [1]. Next we state a lemma which will help us use shallow cuttings efficiently in our data structure. This is a modification of a construction by Makris and Tsakalidis[20].

Lemma 3.1. *Let R be a set of points in $\mathbb{R}^3$. Choose a strip $\mathcal{R}$ in the plane (i.e., the first two dimensions of $\mathbb{R}^3$) such that the projection of all the points of R onto the plane lie inside it. One can construct a subdivision $\mathcal{A}$ of the strip $\mathcal{R}$ into $O(|R|)$ smaller orthogonal rectangles such that for any given query point $q(q_x, q_y, q_z)$ in $\mathbb{R}^3$, if we find the rectangle in $\mathcal{A}$ that contains the projection $q(q_x, q_y)$, then it is possible to find a point of R that is dominated by q or conclude that none of the points in R are dominated by q.*

3.2 Handling a special case We start the presentation of our solution by first handling a special case of a set S of n 4-sided rectangles all of which cross the hyperplane $x = x^*$. We shall establish the following lemma.

Lemma 3.2. *Given a set S of n 4-sided rectangles all of which cross the plane $x = x^*$, we wish to answer the orthogonal point enclosure query. The space occupied is $O(n \log^* n)$ and excluding the time taken to query the point location data structure, the query time is $O(\log^* n + k)$.*

We discuss the case where q is to the right of $x = x^*$. From Observation 1, a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z]$ is reported iff $x_2 \geq q_x$, $y \geq q_y$ and $z \geq q_z$, i.e., (x_2, y, z) dominates (q_x, q_y, q_z). Each rectangle in S is converted into a point (x_2, y, z). Call this new point set P. (The case where q is to the left of $x = x^*$ is handled symmetrically.)

The key idea here is to compute a shallow cutting for the $\log^{(i)} n$-level[1] of P to obtain a point set R_i, $\forall 0 \leq i \leq \log^* n$. For each point $p \in R_i$, based on the points of P which dominate it, build its *local structure* which is the optimal three-dimensional dominance reporting structure of Afshani [1]. Next, using Lemma 3.1 compute an arrangement $\mathcal{A}_i$ based on the point set R_i, $\forall 0 \leq i \leq \log^* n$. Finally, collect all the rectangles in the arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\log^* n}$ and construct the optimal structure of Chazelle [12] which can answer the orthogonal point enclosure query in $\mathbb{R}^2$. Call it a *global structure.*

For a given R_i, $|R_i| = O(n/\log^{(i)} n)$. *Local structure* of a point in R_i is built on $O(\log^{(i)} n)$ points of P and hence, occupies $O(\log^{(i)} n)$ space. Overall space occupied by the local structures of all the points in R_i will be $O(n)$. The total space occupied by all the local structures corresponding to $R_0, R_1, \ldots, R_{\log^* n}$ will be $O(n \log^* n)$. The number of rectangles in the arrangement $\mathcal{A}_i$ will be $O(n/\log^{(i)} n)$ and hence, the total number of rectangles in the arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots, \mathcal{A}_{\log^* n}$ will be $O(n)$. As the structure of Chazelle [12] uses space linear to the number of rectangles it is built on, the global structure will occupy only $O(n)$ space.

Given a query point q, if we succeed to find a point p from some point set R_i which is dominated by q in $\mathbb{R}^3$, then it is sufficient to query the local structure of p and be done. Also, it is desirable that the size of the local structure of p is as small as possible. Therefore, our objective is to find the largest i s.t. there is a point p in R_i which is dominated by q. For this, we query the global structure with (q_x, q_y), which will report exactly one rectangle from each $\mathcal{A}_i, \forall 1 \leq i \leq \log^* n$. Scan the reported rectangles to find that rectangle whose corresponding point satisfies our objective. Once such a point $p \in R_i$ has been found, we ask a three-dimensional dominance reporting query on the local structure of p and finish the algorithm.

The time taken to perform the query on the global structure is $O(\log n + \log^* n) = O(\log n)$, where $\log^* n$ is the number of rectangles reported. Scanning the reported rectangles to find an appropriate point $p \in R_i$ takes $O(\log^* n)$ time. If $i < \log^* n$, then $k =$

[1]$\log^{(0)} n = n$ and $\log^{(i)} n = \log(\log^{(i-1)} n)$. $\log^* n$ is the smallest value of i s.t. $\log^{(i)} n \leq 1$.

$\Omega(\log^{(i+1)} n)$, since there is no point in R_{i+1} which is dominated by q; then, querying the local structure of p takes $O(\log\log^{(i)} n+k) = O(\log^{(i+1)} n+k) = O(k)$ time. Else, if $i = \log^* n$, then querying the local structure of p takes $O(\log\log^{(\log^* n)} n + k) = O(1 + k) = O(1)$ time, since $k = O(1)$. Therefore, excluding the time taken to query the global structure, the query time is $O(\log^* n + k)$.

3.3 $O(\log n \cdot \log^* n + k)$**-query time solution** Making use of the solution for the special case above, we present a solution for orthogonal point enclosure on 4-sided rectangles which uses $O(n \log^* n)$ space and $O(\log n \log^* n + k)$ query time. As done before, we first build an interval tree IT based on the projections of the rectangles of S on the x-axis. We shall focus on the case of reporting rectangles at those nodes v where q is to the right of $split(v)$. The symmetrical case can be similarly handled. At each node v, based on rectangles S_v, construct the *local structure* as described in section 3.2. The crucial technical aspect to take care while constructing the local structure is the following: The arrangements $\mathcal{A}_{(\cdot)}$ are constructed using Lemma 3.1, which requires as input a strip $\mathcal{R}$. For a node v with $range(v) = [x_l, x_r]$, the strip $\mathcal{R}$ will be $[split(v), x_r] \times (-\infty, +\infty)$.

Finally, we collect all the rectangles in arrangements $\mathcal{A}_0, \mathcal{A}_1, \ldots$ from all the nodes in IT and construct the optimal structure of Chazelle [12], which can answer the orthogonal point enclosure query in $\mathbb{R}^2$. This is our actual *global structure*.

Space Analysis: From section 3.2, the space occupied by the local structure at node v will be $O(|S_v| \log^* |S_v|)$ and the total space occupied by all the local structures in IT will be $O(n \log^* n)$. The number of rectangles in the arrangements constructed at node v will be $O(|S_v|)$ and the total number of rectangles collected from all the nodes in IT will be $O(n)$. Therefore, the global structure will occupy $O(n)$ space. The total space occupied by our data structure will be $O(n \log^* n)$.

Given a query point q, let Π be the path from root to the leaf node containing q_x. We query the global structure to report all the rectangles containing (q_x, q_y). The crucial observation is that by our choice of strip $\mathcal{R}$ for each node in IT, if a node v doesn't lie on Π, then no rectangle corresponding to v will be reported. On the other hand, if a node v lies on Π, then exactly $\log^* |S_v|$ rectangles will be reported. To report the rectangles in $S_v \cap q$, we follow the query algorithm discussed in section 3.2. Repeating this procedure at every node on Π will report all the rectangles in $S \cap q$.

Query Analysis: Querying the global structure takes $O(\log n + \log n \cdot \log^* n)$ time, since $O(\log^* n)$ rectangles will be reported from each node on Π. From the analysis in section 3.2, the time taken to report rectangles in $S_v \cap q$, at each node v, will be $O(\log^* n + |S_v \cap q|)$. Therefore, the overall query time will be $O(\log n \cdot \log^* n + k)$. This finishes the proof of Theorem 3.1.

Remark 1: To answer $O(\log n)$ point location queries simultaneously, one would expect that an extra dimension on the rectangles is needed to capture their corresponding node in the interval tree. By suitably modifying the construction of Makris and Tsakalidis (with the introduction of the concept of "strip $\mathcal{R}$"), we are able to simultaneously solve multiple point location queries while staying with *OPEQ* in $\mathbb{R}^2$. We believe this is a novel idea.

Remark 2: To obtain Theorem 3.1, we computed a shallow cutting for the $\log^{(i)} n$-level of P to obtain a point set R_i, $\forall 0 \le i \le \log^* n$. Instead, suppose we compute a shallow cutting for the $\log^{(i)} n$-level of P to obtain a point set R_i, $\forall 0 \le i \le c$, for any integer constant $c \ge 1$. Then, it can verified that one can obtain a data structure with space $O(n)$ and query time $O(\log n \cdot \log^{(c+1)} n + k)$.

4 OPEQ for 4-sided rectangles: optimal query time

In the above mentioned solution, we obtain $O(\log n \cdot \log^* n + k)$ query time, since $|\Pi| = O(\log n)$ and the time taken to report $S_v \cap q$ at each node $v \in \Pi$ is $O(\log^* n + |S_v \cap q|)$. One way to obtain a query time of $O(\log n + k)$ is by restricting $|\Pi| = O(\log n/ \log^* n)$; indeed this can be achieved by decreasing the height of the interval tree to $O(\log n/ \log f) = O(\log n/ \log^* n)$ and increasing the fanout to $f = 2^{\log^* n}$. However, we now have to handle the "middle" structure for which we use some additional technical and novel ideas. We note that this idea of increasing the fanout of an interval tree has been used in the past [4, 7] for some aggregate problems involving rectangles; but our handling of the middle structure is completely different. The key observation to handle middle structure is that since the space is already $\log^* n$ factor away from linear, we can afford to "move" the $\log^* n$ factor from the query time to an additive term in the space complexity.

Skeleton structure: Construct an interval tree IT with fanout $f = 2^{\log^* n}$. The rectangles of S are projected onto the x-axis (each rectangle gets projected into an interval). Let E be the set of $2n$ endpoints of these projected intervals. Divide the x-axis into $2n$ vertical slabs such that each slab covers exactly 1 point of E. Create an f-ary tree IT on these slabs, each of which corresponds to a leaf node in IT. For each node $v \in IT$, we define its range on the x-axis, $range(v)$.

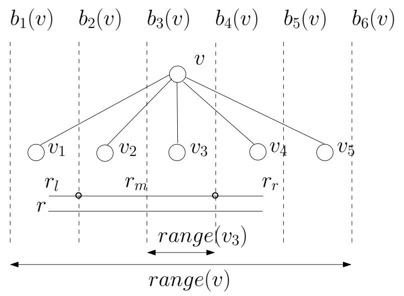

Figure 2: An internal node v in the interval tree IT.

If v is a leaf node, then $range(v)$ is the portion of the x-axis, occupied by the slab corresponding to the leaf; else if v is an internal node, then $range(v)$ is the union of the ranges of its children $v_1, v_2, \ldots, v_f$, i.e., $range(v) = \bigcup_{i=1}^{f} range(v_i) = [x_l, x_r]$. For each internal node $v \in IT$, we also define $f+1$ boundary slabs $b_1(v), b_2(v), \ldots, b_{f+1}(v)$: $b_1(v) = x_l, b_{f+1} = x_r$ and $\forall 1 < i < f+1$, b_i is the boundary separating $range(v_{i-1})$ and $range(v_i)$. Consider a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z] \in S$. Rectangle r is assigned to an internal node $v \in IT$, if the interval $[x_1, x_2]$ crosses one of the slab boundaries of v but doesn't cross any of the slab boundaries of parent of v. See Figure 3 for an example of an internal node v in the interval tree. Let S_v be the set of rectangles of S associated with node v.

Each rectangle in S is broken into three disjoint rectangles as follows: Consider a rectangle $r = [x_1, x_2] \times (-\infty, y] \times (-\infty, z] \in S_v$. Let x_1 lie in $range(v_i)$ and x_2 lie in $range(v_j)$. Then r is broken into a *left rectangle* $r_l = [x_1, b_{i+1}(v)) \times (-\infty, y] \times (-\infty, z]$, a *middle rectangle* $r_m = [b_{i+1}(v), b_j(v)] \times (-\infty, y] \times (-\infty, z]$ and a *right rectangle* $r_r = (b_j(v), x_2] \times (-\infty, y] \times (-\infty, z]$. Note that if $j = i+1$, then we will only have a left and a right rectangle. Define S_v^l, S_v^m and S_v^r to be the set of left, middle and right rectangles obtained by breaking S_v. Furthermore, let $S^l = \bigcup_{v \in IT} S_v^l$, $S^m = \bigcup_{v \in IT} S_v^m$ and $S^r = \bigcup_{v \in IT} S_v^r$ be the sets of all left, middle and right rectangles, respectively. Note that it suffices to build separate data structures to handle S^l, S^m and S^r.

The solution built in section 3 can easily be adapted to report $S^l \cap q$ and $S^r \cap q$ in $O(\log n + k)$ query time, since the height of the tree is now $O(\log n / \log^* n)$. The remaining part of this subsection will focus on building a suitable data structure(s) to report $S^m \cap q$ in $O(\log n + k)$ query time.

Local structure M_v: At each node $v \in IT$, we store a local structure M_v based on the rectangles S_v^m. We first describe construction of M_v and how to query it to report $S_v^m \cap q$. Then, we shall describe the global structure and the global query algorithm to report $S^m \cap q$. We shall utilize the fact that the endpoints of the x-projections of S_v^m comes from a fixed universe $[1 : f] = [1 : 2^{\log^* n}]$. The primary structure of M_v is a segment tree [15] built on the x-projections of S_v^m. For each node $u \in M_v$, define $S_v^m(u)$ to be the rectangles whose x-projection was associated with node u. Also, associate $range(u_v)$ with each node $u \in M_v$, where $range(u_v) \subseteq range(v)$ is the span of boundary slabs covered by the leaf nodes in the subtree of u (see Figure 3(a)). The height of M_v will be $O(\log 2^{\log^* n}) = O(\log^* n)$ and therefore, the space occupied by M_v will be $O(|S_v^m| \log^* n)$.

Given a query point $q(q_x, q_y, q_z)$, we trace a path Π_v in M_v from the root to the leaf node using q_x. At each node $u \in \Pi_v$, we need to report a rectangle $r \in S_v^m(u)$ iff yz-projection of r contains (q_y, q_z), i.e., $q_y \leq y$ and $q_z \leq z$ (a $2d$-dominance query). Unfortunately, using standard structures such as a priority search tree [15] will not help us to achieve our desired query time.

Instead, we shall build the following structure on the yz-projections of $S_v^m(u)$: Convert each rectangle $r \in S_v^m(u)$ into a new point (y, z) and the query q into a query rectangle $[q_y, \infty) \times [q_z, \infty)$. For the sake of convenience, we shall refer to the new point set as $S_v^m(u)$ itself. Sort the point set $S_v^m(u)$ in non-increasing order of their y-coordinate values. For simplicity, we will still refer to the sorted list as $S_v^m(u)$ itself. Add a dummy point at the end of the list with $y = -\infty$ and an arbitrary value of z. With the i^{th} element in $S_v^m(u)$, we store a list L_i which is the $1^{st}, 2^{nd}, \ldots, i^{th}$ element of $S_v^m(u)$ in non-increasing order of their z-coordinate values (see Figure 3 (b) & (c)). Then the total size of all the lists $L_i, \forall 1 \leq i \leq |S_v^m(u)|$ will be $O(|S_v^m(u)|^2)$. However, notice that given two consecutive elements i and $i+1$ in $S_v^m(u)$, L_{i+1} can be obtained from L_i by making $O(1)$ changes. Now treating y-coordinate as time, we store all the lists $L_i, \forall 1 \leq i \leq |S_v^m(u)|$, in a partially persistent structure [17]. The total number of memory modifications will be $O(|S_v^m(u)|)$ and hence, the total size of all the lists reduces from $O(|S_v^m(u)|^2)$ to $O(|S_v^m(u)|)$. To answer the query at node u, we locate the element i in $S_v^m(u)$ which is the predecessor of q_y. Then, we walk down the list L_i to report the corresponding rectangles till either the list gets exhausted or we reach a point whose z-coordinate is less than q_z. Ignoring the time taken to locate element i in $S_v^m(u)$, the time spent at node u will be $O(1 + |S_v^m(u) \cap q|)$. The performance of the local structure M_v is summarized next.

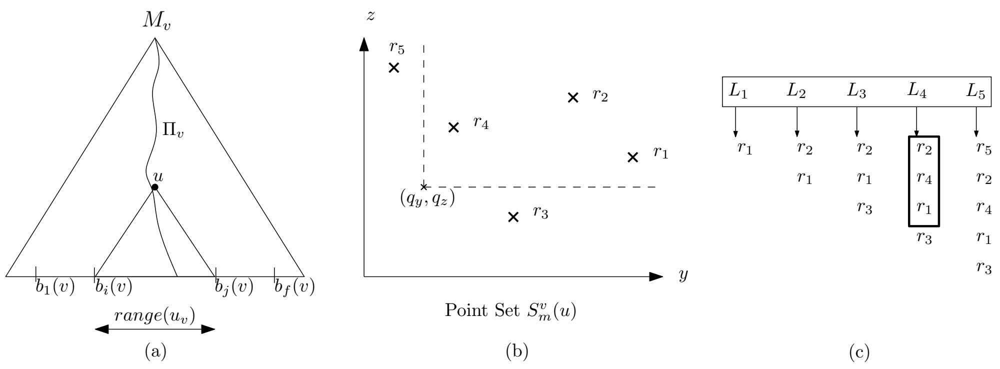

Figure 3: (a) M_v Structure. (b) Querying point set $S_v^m(u)$ with (q_y, q_z). (c) Lists L_i. For the example query in (b), we walk down the list L_4 to report r_2, r_4 and r_1.

LEMMA 4.1. *Given a set S_v^m of 4-sided rectangles, we wish to answer the orthogonal point enclosure query. The endpoints of the x-projections of S_v^m come from a fixed universe $[1 : f] = [1 : 2^{\log^* n}]$. Local structure M_v occupies $O(|S_v^m| \log^* n)$ space and excluding the time taken to locate element i in $S_v^m(u), \forall u \in \Pi_v$, the query time will be $O(\log^* n + |S_v^m \cap q|)$.*

Global structure: The only missing ingredient is an efficient technique to locate element i in $S_v^m(u)$'s which are visited during a query. Our technique will be based on the following simple yet powerful observation.

OBSERVATION 2. *For a given query $q(q_x, q_y, q_z)$, let the i^{th} element in point set $S_v^m(u)$ be the predecessor of q_y. Then we walk down the list L_i in $S_v^m(u)$ iff (i) $q_x \in range(u_v)$, and (ii) $q_y \in (y_{i+1}, y_i]$, where y_i and y_{i+1} are the y-coordinates of the i^{th} and $(i+1)^{th}$ entry in point set $S_v^m(u)$.*

Using the above observation, at every node $u \in M_v$, the i^{th} element in $S_v^m(u), \forall 1 \leq i \leq |S_v^m(u)|$, is mapped to a rectangle $range(u_v) \times (y_{i+1}, y_i]$ in $\mathbb{R}^2$. This process is repeated at every M_v structure in IT. Collect all the newly mapped rectangles and construct the optimal structure of Chazelle [12] which can answer $OPEQ$ in $\mathbb{R}^2$. This is our *global structure*.

Space Analysis: Since each local structure M_v occupies $O(|S_v^m| \log^* n)$ space, the overall space occupied by all the local structures in IT will be $O(n \log^* n)$. The number of rectangles mapped from all the nodes in M_v is $O(|S_v^m| \log^* n)$ and hence, the total number of rectangles collected to construct the global structure is $O(n \log^* n)$. Therefore, the global structure occupies $O(n \log^* n)$ space.

Given a query point q, we query the global structure with (q_x, q_y). From Observation 2 and our construction it is guaranteed that a rectangle $range(u_v) \times (y_{i+1}, y_i]$ is reported iff the i^{th} element in $S_v^m(u)$ is the predecessor of q_y. Then for each reported rectangle we go to its corresponding list L_i and report the rectangles in $S_v^m(u) \cap q$. This ensures that all the rectangles in $S^m \cap q$ get reported.

Query Analysis: Querying the global structure takes $O(\log n)$ time, since rectangles corresponding to $O(\log^* n)$ nodes from each of the $O(\log n / \log^* n)$ M_v structures are reported. Adding the time spent at each of the local M_v structures, we get $O(\sum_{v \in \Pi}(\log^* n + |S_v^m \cap q|)) = O(\log n + k)$. Overall query time to report $S^m \cap q$ will be $O(\log n + k)$. The final result is summarized below.

THEOREM 4.1. *Orthogonal point enclosure query on 4-sided rectangles can be answered using a structure of $O(n \log^* n)$ size and in $O(\log n + k)$ query time.*

5 OPEQ on 5- and 6-sided rectangles

In this section, we use the result obtained for $OPEQ$ on 4-sided rectangles to answer $OPEQ$ on 5- and 6-sided rectangles. First, we present a solution for 5-sided rectangles. Alstrup, Brodal and Rauhe introduced the grid-based technique [6] to index points for answering orthogonal range reporting queries. We also use their grid-based technique, but suitably adapt it for handling the indexing of 5-sided rectangles. At a high level, the query algorithm is based on the following approach: Theorem 2.1 handles $OPEQ$ on 5-sided rectangles in $O(\log^3 n + k)$ query time. When $k \geq \log^3 n$, Theorem 2.1 will have a query time $O(k)$, which is good. When $k < \log^3 n$, we can no longer use Theorem 2.1 but the low-output size allows us to pre-compute partial answers to each query. Appendix D provides the complete details of the data structure and the query algorithm.

THEOREM 5.1. *Orthogonal point enclosure query on 5-sided rectangles can be answered using a structure of*

$O(n \log^* n)$ size and in $O(\log n \cdot \log\log n + k)$ query time.

Now we look at *OPEQ* for 6-sided rectangles. In Theorem 2.1, *OPEQ* for 6-sided rectangles was handled by placing at each node of the interval tree a data structure which can handle *OPEQ* for 5-sided rectangles. Now placing Theorem 5.1 at each node of the interval tree leads to the following result.

THEOREM 5.2. *Orthogonal point enclosure query on* 6-*sided rectangles can be answered using a structure of $O(n \log^* n)$ size and in $O(\log^2 n \cdot \log\log n + k)$ query time.*

By using the idea of segment trees, the above result extends to higher dimensions as well.

THEOREM 5.3. *Orthogonal point enclosure query on $2d$-sided rectangles in $\mathbb{R}^d (d \geq 3)$ can be answered using a structure of $O(n \log^{d-3} n \cdot \log^* n)$ size and in $O(\log^{d-1} n \cdot \log\log n + k)$ query time.*

6 Point location in orthogonal subdivisions in $\mathbb{R}^3$

A set of k disjoint orthogonal rectangles in $\mathbb{R}^3$ can partition the three-dimensional space into $\Omega(k^{3/2})$ rectangles, i.e., $\Omega(k^{3/2})$ rectangles will be needed to fill the holes created (Ideally one would want to fill the holes with $O(k)$ rectangles). Therefore, "directly" using traditional techniques in the literature (such as separator-based approach [16], sampling-based approach [9]) will lead to a space-expensive data structure. The key idea we use here is to use an interval tree to project the rectangles onto a two-dimensional space and then apply sampling-based techniques on these two-dimensional rectangles (similar to the ideas used to answer *OPEQ* for 4-sided rectangles). Even though we use the term orthogonal subdivisions, our solution works for any set S of disjoint axes-parallel rectangles which need not cover the entire space.

6.1 Simple solution First, we present a simple solution for this problem. Based on the x-projection of the rectangles in set S, an interval tree IT is built. Let S_v be the set of rectangles assigned to a node $v \in IT$. Project the rectangles in S_v onto the yz-plane and build a $2d$ orthogonal point location data structure (for e.g., [15, 24]) based on the projected rectangles in S_v. The space occupied by the point location data structure will be $O(|S_v|)$ and it can answer a $2d$ point location query in the yz-plane in $O(\log |S_v|)$ time. The overall space occupied by the data structure will be $O(n)$.

Given a query point q, let Π be the path from root to the leaf node containing the x-coordinate of q. At each node $v \in \Pi$, we query the $2d$ point location structure built on S_v. Let $s \in S_v$ be the rectangle (if any) containing the yz-projection of q. Report s, if q lies inside the original rectangle (in $3d$) corresponding to s. This leads to a query time of $O(\log^2 n)$.

6.2 Improving the query time The objective is to reduce the query time spent at each node in Π from $O(\log n)$ to $O(f)$, for some parameter $f \in [\Omega(\log\log n), o(\log n)]$. The exact value of f will be determined later. To achieve this objective, we partition S_v into groups of size $\approx 2^f$, such that a $2d$ point location query on S_v is reduced to a $2d$ point location query on exactly one group.

Local Structure: At each node $v \in IT$, take a random sample $R_v \subseteq S_v$ of size $|S_v|/r$, where $r = 2^f$. Then, the rectangles in set R_v are projected onto the yz-plane. Compute a *trapezoidal decomposition* $T(R_v)$: a subdivision of the yz-plane into rectangles formed by the rectangles of R_v and the vertical upward and downward rays from each endpoint of R_v. It is easy to see that $T(R_v)$ will have $O(|S_v|/r)$ rectangles. For each rectangle $\square \in T(R_v)$, we define *conflict list*, $S_v^{\square}$, to be the set of rectangles of S_v whose projection onto yz-plane intersect with $\square$. By a standard analysis of Clarkson and Shor [14], the probability of

$$\sum_{\square \in T(R_v)} |S_v^{\square}| = O(|S_v|) \text{ and } \max_{\square \in T(R_v)} |S_v^{\square}| = O(r \cdot \lg |S_v|)$$

is greater than a positive constant. If the above mentioned properties are violated, then we discard our current random sample R_v and pick a new random sample. The expected number of trials to satisfy the above properties is $O(1)$. Finally, for each $\square \in T(R_v)$, we build a $2d$ orthogonal point location data structure based on the rectangles in the set $S_v^{\square}$. In this way, we have succeeded in partitioning S_v into $O(|S_v|/r)$ groups, with each group (i.e., a rectangle $\square \in T(R_v)$) containing $O(r \cdot \log |S_v|) = O(2^f \log |S_v|)$ rectangles of S_v. For a node $v \in \Pi$, if $\square \in T(R_v)$ is the rectangle containing yz-projection of q, then the rectangle in S_v containing yz-projection of q (if any) can be found in $O(\log(r \log |S_v|)) = O(f + \log\log n) = O(f)$ time. Now, we present a global structure to efficiently find these $\square$ rectangles at each node in Π.

Global Structure: We borrow an idea used in Chan *et al.* [10] and Afshani *et al.* [3]: At a node $v \in IT$ having a range, *range*(v), associated with it, each rectangle $\square \in T(R_v)$ is mapped to a $3d$-rectangle $\square \times$ *range*(v). Collect all the $3d$-rectangles created at all the nodes in IT and build a data structure which can answer *OPEQ* on 6-sided rectangles. Note that a query point q will lie inside a $3D$-rectangle, $\square \times range(v)$, iff (a) $v \in \Pi$, and (b) yz-projection of q lies inside $\square$.

Analysis: The space occupied by the interval tree and all the local structures is $O(n)$. Since $|\Pi| = O(\log n)$ and performing a $2d$ point location at each node in Π takes $O(f)$ time, the total query time spent at the local structures is $O(f \log n)$. To handle *OPEQ* on m 6-sided rectangles, we use the result of Afshani *et al.* [3] which uses $O(mh \log m)$ space and answers the query in $O(\log^2 m / \log h + k)$ time, for any $h \geq 2$. In our case, $m = \sum_{v \in IT} |T(R_v)| = O(n/2^f)$, since $|T(R_v)| = O(|S_v|/r) = O(|S_v|/2^f)$. To keep the space of the global structure $O(n)$, we need

$$h = O\left(\frac{2^f}{\log n}\right) \tag{6.1}$$

Querying the global structure takes $O(\frac{\log^2 n}{\log h} + \log n)$ time. The overall query time will be $O\left(f \log n + \frac{\log^2 n}{\log h} + \log n\right)$. In order to minimize this quantity while satisfying constraint 6.1, we set $f = \sqrt{\log n}$ and $h = \frac{2^{\sqrt{\log n}}}{\log n}$. Therefore, the overall query time is bounded by $O(\log^{3/2} n)$. The overall performance is summarized next.

THEOREM 6.1. *There exists an $O(n)$ space data structure to answer point location in orthogonal subdivisions in $\mathbb{R}^3$ in $O(\log^{3/2} n)$ query time.*

Higher Dimensions: The above solution can be easily extended to higher dimensions. In $\mathbb{R}^d$, build an interval tree IT based on the projection of the rectangles of S onto the last dimension. At each node $v \in IT$, based on the projection of the rectangles of S_v onto the first $d-1$ dimensions, build an orthogonal point location structure in $\mathbb{R}^{d-1}$. Given a query point $q \in \mathbb{R}^d$, at each node $v \in \Pi$, we query the point location structure built on S_v. The final result is summarized next.

THEOREM 6.2. *There exists an $O(n)$ space data structure to answer point location in orthogonal subdivisions in $\mathbb{R}^d (d \geq 3)$ in $O(\log^{d-3/2} n)$ query time.*

7 Open problems

We conclude with some open problems:

1. Is it possible to answer *OPEQ* for 4-sided rectangles in $\mathbb{R}^3$ in $O(\log n + k)$ query time using an $O(n)$ space structure? As of now, an optimal solution in $\mathbb{R}^3$ is known only for 3-sided rectangles.

2. Can point location in orthogonal subdivisions in $\mathbb{R}^3$ be done in $O(\log n)$ query time using an $O(n)$ space structure?

Acknowledgments: The author would like to thank his advisor Prof. Ravi Janardan for his constant support and advice. Specifically, the author is thankful to his advisor for teaching him the art of technical writing (errors are author's own). The author would like to thank Prof. Yufei Tao and Prof. Peyman Afshani for fruitful discussions on *OPEQ* and point location in orthogonal subdivisions, respectively. The author would like to thank Prof. Barna Saha for teaching a course on Algorithmic Techniques for Big Data Analysis which was extremely helpful in building foundations for conducting this research. Finally, the author would like to thank Prof. Paul Beame for teaching a course on Sublinear (and Streaming) Algorithms at University of Washington which triggered some of the ideas in this paper.

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] Pankaj K. Agarwal, Lars Arge, Haim Kaplan, Eyal Molad, Robert Endre Tarjan, and Ke Yi. An optimal dynamic data structure for stabbing-semigroup queries. *SIAM Journal of Computing*, 41(1):104–127, 2012.

[5] Pankaj K. Agarwal and Jeff Erickson. Geometric range searching and its relatives. *Advances in Discrete and Computational Geometry*, pages 1–56, 1999.

[6] 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.

[7] Lars Arge and Jeffrey Scott Vitter. Optimal external memory interval management. *SIAM Journal of Computing*, 32(6):1488–1508, 2003.

[8] A. Boroujerdi and Bernard M. E. Moret. Persistency in computational geometry. In *Proceedings of the Canadian Conference on Computational Geometry (CCCG)*, pages 241–246, 1995.

[9] Timothy M. Chan. Persistent predecessor search and orthogonal point location on the word ram. *ACM Transactions on Algorithms*, 9(3):22:1–22:22, 2013.

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

[11] Timothy M. Chan and Patrick Lee. On constant factors in comparison-based geometric algorithms and data structures. In *Proceedings of Symposium on Computational Geometry (SoCG)*, pages 40–49, 2014.

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

[13] Bernard Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal of Computing*, 17(3):427–462, 1988.

[14] Kenneth L. Clarkson and Peter W. Shor. Application of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[15] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, 3rd edition, 2008.

[16] 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.

[17] James R. Driscoll, Neil Sarnak, Daniel Dominic Sleator, and Robert Endre Tarjan. Making data structures persistent. *Journal of Computer and System Sciences (JCSS)*, 38(1):86–124, 1989.

[18] Herbert Edelsbrunner. A new approach to rectangle intersections, part I. *International Journal of Computer Mathematics*, 13:209–219, 1983.

[19] Herbert Edelsbrunner, Gunter Haring, and David Hilbert. Rectangular point location in d dimensions with applications. *The Computer Journal*, 29(1):76–82, 1986.

[20] Christos Makris and Athanasios K. Tsakalidis. Algorithms for three-dimensional dominance searching in linear space. *Information Processing Letters (IPL)*, 66(6):277–283, 1998.

[21] Christos Makris and Konstantinos Tsakalidis. An improved algorithm for static 3d dominance reporting in the pointer machine. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 568–577, 2012.

[22] Kurt Mehlhorn. *Data Structures and Algorithms 3*, volume 3 of *Monographs in Theoretical Computer Science. An EATCS Series.* Springer, 1984.

[23] Yakov Nekrich. I/O-efficient point location in a set of rectangles. In *Latin American Symposium on Theoretical Informatics (LATIN)*, pages 687–698, 2008.

[24] Jack Snoeyink. Handbook of discrete and computational geometry. In Jacob E. Goodman and Joseph O'Rourke, editors, *Point Location*, pages 559–574. CRC Press, Inc., Boca Raton, FL, USA, 1997.

Appendix A: Model of computation

All the data structures proposed in this paper are in a *pointer machine model*. This model has been used extensively for proving several interesting lower bounds and upper bounds for range searching and related problems. Loosely speaking, in this model the data structure is modeled as a graph and one is not allowed to do a random access. Formally, as defined by Chazelle [13], in this model a data structure can be regarded as a directed graph, where each node stores $O(1)$ real values and $O(1)$ pointers to other nodes. Random access to a node is not allowed and only pointers can be used to access a node. We begin answering a query using a pointer to a *root node* of the data structure. The *query time* of an algorithm is the total number of nodes visited, whereas the *size* of a structure is the number of its nodes and edges. For further details of this model, we refer the reader to the survey paper by Agarwal and Erickson [5].

Appendix B: Interval tree

We shall give a brief description of a classic structure called an *interval tree* [18] (see also [22]). It has traditionally been used to answer the orthogonal point enclosure query in $\mathbb{R}^1$.

Consider a set S of n intervals in $\mathbb{R}^1$ and let E be the set of endpoints of the intervals in S. Build a binary search tree IT in which the points of E are stored at the leaves from left to right in increasing order of their coordinate value. At each node $v \in IT$, we define $split(v)$ and $range(v)$. $split(v)$ is a value such that points of E in the left (resp. right) subtree of v have coordinate value less than or equal to (resp. greater than) $split(v)$. For the root node, $root$, $range(root) = (-\infty, +\infty)$. For a node v, if the $range(v) = [x_l, x_r]$ then the range of its left (resp. right) child will be $[x_l, split(v)]$ (resp. $(split(v), x_r]$). Each interval is assigned to exactly one node v in IT such that the interval is contained inside $range(v)$ but is not contained inside $range(\cdot)$ of its children.

Let S_v be the set of intervals assigned at node v. We maintain additional structures at node v: A list IT_v^l (resp. IT_v^r) which stores the left (resp. right) endpoints of S_v in non-decreasing (resp. non-increasing) order of their coordinate value. The space occupied by the interval tree is $O(n)$.

Given a query point q to answer orthogonal point enclosure in $\mathbb{R}^1$, we visit a path from root to the leaf node of IT, s.t., at every node v on the path, $q \in range(v)$. At each node v on the search path, if the query point q lies to the left of $split(v)$ then we traverse the list IT_v^l from left to right till the entries in the list get exhausted or we find an endpoint whose coordinate

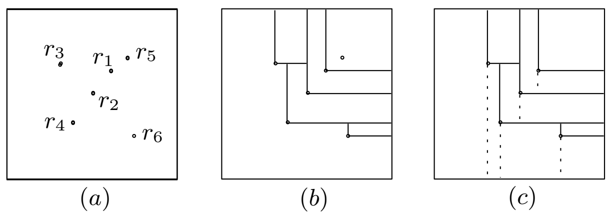

Figure 4: (a) Projection of points in R onto the xy-plane. (b) Region r'_i associated with each point. (c) Trapezoidal decomposition to obtain the subdivision $\mathcal{A}$.

value is greater than q. The case of q lying to the right of $split(v)$ is handled symmetrically. The time taken to answer the query is $O(\log n + k)$, where k is the number of intervals reported.

Appendix C: Proof of lemma 3.1

Let $r_1, r_2, \ldots, r_{|R|}$ be the sequence of points of R in non-decreasing order of their z-coordinate values. Each point $r_i(r_x, r_y, r_z)$ is projected onto the plane and a region r'_i is associated with it: $r'_i = ([r_x, \infty) \times [r_y, \infty)) \setminus \bigcup_{j=1}^{i-1} r'_j$. See Figure 4(a). Note that r'_i will be an empty set iff the point r_i dominates any other point in R. See Figure 4(b); r'_5 is an empty set. We shall discard all such points from the set R. Next we perform a trapezoidal decomposition of the strip $\mathcal{R}$ to obtain our subdivision $\mathcal{A}$, i.e., we shoot rays towards $y = -\infty$ from every remaining point in R till it hits an edge or the boundary of the strip. See Figure 4(c). It is easy to see that the number of rectangles in the subdivision will be $O(|R|)$.

Given a query point q, we perform a point location query on the subdivision $\mathcal{A}$. Two cases arise: (a) None of the r'_is contain q. It means none of the points in R are dominated by q. (b) Let r_i be the point associated with the rectangle which contains q. Note that among the points of R which are dominated by q in the plane, r_i has the smallest z-coordinate value. If the z-coordinate of r_i is smaller than q_z, then we have found a point in R that is dominated by q; else we can conclude that none of the points in R are dominated by q.

Appendix D: OPEQ for 5-sided rectangles

Structure: Define a parameter $t = \log^4 n$. Consider the projection of the rectangles of S on to the xy-plane and impose an orthogonal $(2\sqrt{\frac{n}{t}}) \times (2\sqrt{\frac{n}{t}})$ grid such that each horizontal and vertical slab contains the projections of $\sqrt{nt}$ sides. Let $S_{root} \subseteq S$ be the set of rectangles stored at the root. A rectangle of S belongs to S_{root} iff it intersects at least one horizontal or vertical boundary of the grid. A couple of data structures are built on S_{root} which will be discussed below. Call this

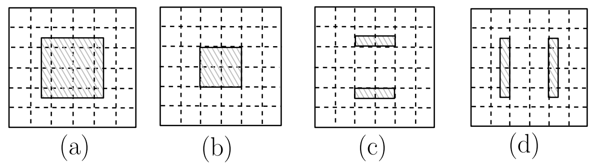

Figure 5: Breaking a rectangle in (a) into 2 horizontal side rectangles (shown in (c)) and 2 vertical side rectangles (shown in (d)).

the root of the recursion tree. Finally, we recurse on the rectangles which lie completely inside a slab. At each node of the recursion tree, if we have m rectangles in the subproblem then the value then the value of t changes to $\log^4 m$ and the grid size changes to $(2\sqrt{\frac{m}{t}}) \times (2\sqrt{\frac{m}{t}})$. We stop the recursion when a subproblem has less than c rectangles, for a suitably large constant c.

A *grid structure*, a *slow structure* and *side structure*'s are built on S_{root}. The *slow structure* is Theorem 2.1 built on 5-sided rectangles S_{root}. The slow structure is queried only when $|S_{root} \cap q| = \Omega(\log^3 n)$. A rectangle r' is *higher* than rectangle r'' if r' has a larger span than r'' along z-direction. In the *grid structure*, for each cell c of the grid, among the rectangles which completely cover c, store the highest $\log^3 n$ rectangles in a linked list L_c in decreasing order of their z-coordinates. As shown in Figure 5, each rectangle in S_{root} is broken into at most 4 *side rectangles.* Observe that the side rectangles are 4-sided rectangles. For each row and column slab, we have a *side structure* which is Theorem 3.1 built on the side rectangles lying inside it.

Space Analysis: The space occupied by the slow structure and the side structures is $O(|S_{root}|)$ and $O(|S_{root}| \log^* |S_{root}|)$, respectively. Note that a rectangle in S is stored at exactly one node in the recursion tree. Therefore, the overall space occupied by the slow structures and the side structures in the recursion tree is $O(n)$ and $O(n \log^* n)$, respectively. The space occupied by the grid structure will be $O(n/t \cdot \log^3 n) = O(n/\log n)$. Thus the space occupied, $S(n)$, by all the grid structures in the recursion tree is given by the recurrence

$$S(n) = \sum_{i=1}^{4\sqrt{n/t}} S(n_i) + O\left(\frac{n}{\log n}\right), \forall i, n_i \leq \sqrt{nt}.$$

This solves to $S(n) = O(n)$. Therefore, the overall space occupied by the data structure will be $O(n \log^* n)$.

Query: Given a query point q, at the root we locate the cell c on the grid containing q. Scan the list L_c

to report rectangles till we either (a) find a rectangle which doesn't contain q, or (b) the end of the list is reached. If case (b) happens, then we have reported $\log^3 n$ rectangles, so we query the slow structure to report $S_{root} \cap q$. If case (a) happens, then we also query the side structures of the horizontal and the vertical slab containing q. Next, we recursively query the horizontal and the vertical slab containing q.

Query Analysis: First we analyze the query time at the root of the recursion tree. Cell c on the grid can be located in $O(\log \sqrt{n/t}) = O(\log n)$ time. If case (a) happens, then the time spent is $O(\log n + |S_{root} \cap q|)$. Else, if case (b) happens, then the time spent is $O(\log^3 n + |S_{root} \cap q|) = O(|S_{root} \cap q|)$, since $|S_{root} \cap q| \geq \log^3 n$. Therefore, the query time at the root is $O(\log n + |S_{root} \cap q|)$. Let $Q(n)$ denote the overall query time (excluding the output portion). Then

$$Q(n) = 2Q(\sqrt{nt}) + O(\log n), t = \log^4 n.$$

This solves to $Q(n) = O(\log n \cdot \log \log n)$. Therefore, the overall query time will be $O(\log n \log \log n + k)$.