Orthogonal Range Reporting and Rectangle Stabbing for Fat Rectangles

Timothy M. Chan[1], Yakov Nekrich[2], and Michiel Smid[3]

[1] Department of Computer Science, University of Illinois at Urbana-Champaign
tmc@illinois.edu
[2] Cheriton School of Computer Science, University of Waterloo.
yakov.nekrich@googlemail.com
[3] School of Computer Science, Carleton University michiel@scs.carleton.ca

Abstract. In this paper we study two geometric data structure problems in the special case when input objects or queries are fat rectangles. We show that in this case a significant improvement compared to the general case can be achieved.
We describe data structures that answer two- and three-dimensional orthogonal range reporting queries in the case when the query range is a *fat* rectangle. Our two-dimensional data structure uses $O(n)$ words and supports queries in $O(\log\log U + k)$ time, where n is the number of points in the data structure, U is the size of the universe and k is the number of points in the query range. Our three-dimensional data structure needs $O(n \log^{\varepsilon} U)$ words of space and answers queries in $O(\log\log U + k)$ time. We also consider the rectangle stabbing problem on a set of three-dimensional fat rectangles. Our data structure uses $O(n)$ space and answers stabbing queries in $O(\log U \log\log U + k)$ time.

1 Introduction

Orthogonal range reporting and rectangle stabbing are two fundamental problems in computational geometry. In the orthogonal range reporting problem we keep a set of points in a data structure; for any axis-parallel query rectangle Q we must report all points in Q. Rectangle stabbing is, in a sense, a dual problem. We keep a set of axis-parallel rectangles in a data structure. For a query point q we must report all rectangles that are stabbed by q, i.e., all rectangles that contain q. A rectangle is fat if its aspect ratio (the ratio of its longest and shortest edges) is bounded by a constant. In this paper we consider the range reporting problem in scenario when query rectangles are fat. We show that significant improvements can be achieved for this special case. We also describe a data structure that supports three-dimensional stabbing queries on a set of fat three-dimensional rectangles.

The range reporting problem and its variants have been studied extensively over the last four decades; see for example, [13, 8, 9, 25, 3, 2, 22, 21, 1, 15, 6, 7]. We refer to [17, 23] for extensive surveys of previous results. The best known data structure for two-dimensional point reporting uses $O(n \log^{\varepsilon} n)$ words of space and supports queries in $O(\log \log U + k)$ time [7]. Henceforth n is the total number of geometric objects (points or rectangles) in the data structure, k is the number of reported objects, and ε is an arbitrarily small positive constant; we assume that all point coordinates are positive integers bounded by a parameter U. The space usage can be reduced to linear or almost-linear at the cost of paying a non-constant penalty for every reported point. Thus there is an $O(n)$-word data structure that supports queries in $O(\log \log U + (k+1) \log^{\varepsilon} n)$ time and $O(n \log \log n)$-word data structure that answers queries in $O(\log \log U + k \log \log n)$ time. If we want to use linear space and spend constant time for every reported point, then the overall query cost is increased to polynomial: the fastest linear-space data structure requires $O(n^{\varepsilon} + k)$ time to answer a query [5]. Better results are known only in the special case when the query range is bounded on three sides [19, 2]; there is a linear-space data structure that answers three-sided queries in $O(\log \log U + k)$ time (or even in $O(1 + k)$ time if $U = O(n)$) [2]. In this paper we show that two-dimensional orthogonal range reporting queries can be answered in $O(\log \log U + k)$ time using an $O(n)$-space data structure under assumption that query rectangles are fat. We also demonstrate that the fatness assumption is profitable for three-dimensional orthogonal range reporting. We show in this paper how to report all points in a three-dimensional axis-parallel fat rectangle in $O(\log \log U + k)$ time using a $O(n \log^{\varepsilon} U)$-word data structure. This is to be compared with the result of Chan et al. [7] that achieves the same query time for arbitrary rectangle queries but uses $O(n \log^{1+\varepsilon} n)$ words of space. The third problem considered in this paper is the three-dimensional stabbing problem on a set of fat rectangles. For a query point q we must report all rectangles that are stabbed by q. We describe a data structure that uses $O(n)$ words of space and supports queries in $O(\log U \log \log U + k)$ time. For comparison, the best known data structures for general rectangles use $O(n \log^{*} n)$ words of space and support queries in $O(\log^2 n)$ time [24].

Our data structure for two-dimensional range reporting, described in Sections 2 and 3, is based on quadtrees. Using a marking scheme on nodes of a quadtree, we divide the plane into $O(n/d)$ canonical rectangles, so that each rectangle contains $O(d)$ points for $d = \log n$. For any fat query rectangle Q, we can quickly find all canonical rectangles R satisfying $Q \cap$

Reference	Space	Time	Query Type
[7]	$O(n)$	$O(\log\log U + (k+1)\log^{\varepsilon} n)$	General
[7]	$O(n \log\log n)$	$O(\log\log U + k \log\log n)$	General
[7]	$O(n \log^{\varepsilon} n)$	$O(\log\log U + k)$	General
[5]	$O(n)$	$O(n^{\varepsilon} + k)$	General
[19]	$O(n$	$O(\log n + k)$	Three-sided
[2]	$O(n$	$O(\log\log U + k)$	Three-sided
[10]	$O(n)$	$O(\log n + k)$	Fat
New	$O(n)$	$O(\log\log U + k)$	Fat

Table 1. Space-time trade-offs for two-dimensional range reporting. Result in line 7 is a corollary from [10], but it is not stated there.

$R \neq \emptyset$ and report all points in $Q \cap R$ for all such R. In Section 4 we describe a data structure that supports three-dimensional range reporting for fat query ranges. It is based on recursive decomposition of the grid similar to [2, 7, 15], but in our case the grid is divided into uniform cells. We describe a data structure for stabbing queries on a set of three-dimensional fat rectangles in Section B. This result is based on reducing a stabbing query to $O(\log U)$ three-dimensional dominance queries. The results of this paper are valid in the word RAM model of computation.

Related Work. A result about range reporting in two-dimensional fat rectangles is implicitly contained in the paper of Chazelle and Edelsbrunner [10]. In [10] the authors describe a linear-space data structure for triangular range reporting. Their data structure can report all points in an arbitrary query triangle, provided that the sides of the triangle parallel to three fixed directions; queries are supported in $O(\log n + k)$ time where k is the number of reported points. We can represent a square as a union of two such triangles and we can represent an arbitrary fat rectangle as a union of $O(1)$ squares. Hence we can answer two-dimensional range reporting queries for fat rectangles in $O(\log n + k)$ time and $O(n)$ space.

Data structures for fat convex objects are studied in e.g., [16, 12, 11, 4]. Iacono and Langerman [14] describe a data structure that supports point location queries in a set of axis-parallel fat d-dimensional rectangles. This data structure answers queries in $O(\log\log U)$ time and uses $O(n \log\log U)$ space for any fixed dimension d.

2 Quadtree-Based Rectangular Subdivision

In this section we describe a planar rectangular subdivision that is used by our two-dimensional data structure. To make the description self-contained, we start with the definition of a compressed quadtree.

A quadtree T_Q is a hierarchical data structure that divides the plane into regions. Let U denote the maximum of x- and y-coordinates of all points. We associate a square (also called a cell) $\texttt{square}(v)$ to every quadtree node v. The root of a quadtree is associated to the square $[0, U] \times [0, U]$. W. l. o. g. we assume that U is a power of 2. If a square $\texttt{square}(v)$ of a node v contains more than one point, then the node v has four children. We divide $\texttt{square}(v)$ into four squares of equal size and associate them to four child nodes of v. A compressed quadtree T is a subtree of T_Q obtained by keeping only those internal nodes of T_Q that have more than one non-empty child.

Marking Nodes in a Quadtree. Let T denote a compressed quadtree on a set of n points. Let $d = \log n$. We mark selected nodes in T by employing the following marking scheme: (i) every d-th leaf is marked and (ii) if an internal node u has at least two children with marked descendants, then u is marked. We can mark nodes of a given quadtree T in linear time using the following method. We will say that a node u is a special node if exactly one child of u has marked descendants. First we traverse the leaves of T in the left-to-right order and mark every d-th leaf, starting with the leftmost one. Then we visit all internal nodes of T in post-order. If a visited node u has exactly one child u_i such that u_i is either marked or special, then we declare that the node u is special. If u has two or more children that are either special or marked, then the node u is marked. Marked nodes induce a subtree T' of T. T' has n/d leaves. Since every internal node of T' has at least two children, T' has at most $n/d - 1$ internal nodes. Hence the total number of marked nodes is $O(n/d)$. Similar methods for selecting nodes were previously used in other tree-based data structures, see e.g., [20, 18].

Rectangular Subdivision. When nodes are marked, we traverse T from the top to the bottom and divide it into $O(n/d)$ rectangles so that each rectangle contains $O(d)$ points. The subdivision is produced as follows. A direct marked descendant of a node u is a descendant u' of u such that u' is marked and there are no marked nodes between u and u'. Suppose that a node u is a marked node and let $u_1, \ldots, u_f$ denote its direct marked descendants. A marked node has at most 4 direct marked descendants, therefore $f \leq 4$. Let $\texttt{square}(u)$ denote the cell of

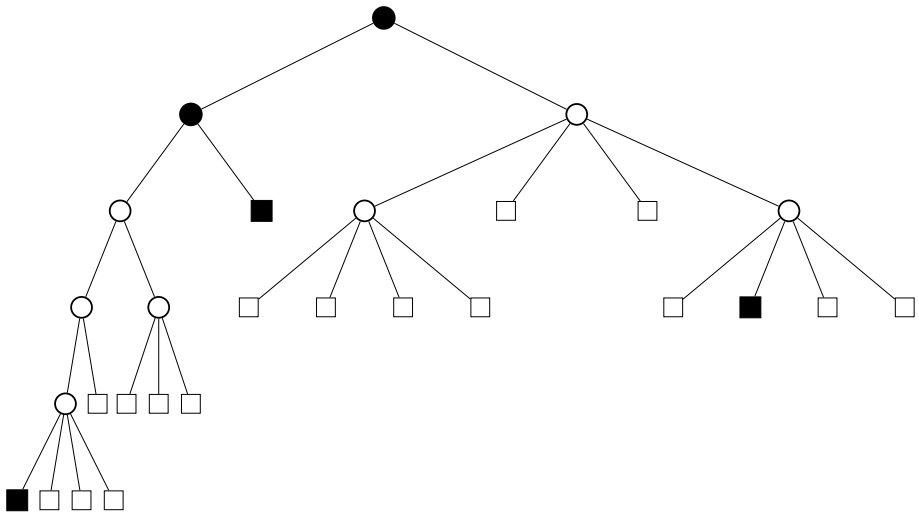

Fig. 1. Marking nodes in a compressed quadtree for $d = 8$. Leaves are shown with squares and internal nodes are shown with circles. Marked leaves and internal nodes are depicted with filled circles and filled squares respectively. Only a part of the quadtree is shown.

a node u. We can represent $\texttt{square}(u) \setminus (\cup_{i=1}^{f} \texttt{square}(u_i))$ as a union of a constant number of rectangles $R_j(u)$. We will say that $R_j(u)$ are rectangles associated to the node u. See Fig. 2. There are $O(n/d)$ marked nodes in the quadtree. Our subdivision consists of rectangles $R_i(u)$ for all marked internal nodes of T and cells $\texttt{square}(v)$ for all marked leaves v of T. By dividing every marked node with marked descendants into rectangles as described above, we obtain a sub-division of the plane into $O(n/d)$ rectangles. Rectangles of this subdivision will be further called *canonical rectangles*.

Lemma 1. *Every canonical rectangle contains $O(d)$ points.*

Proof: Consider a rectangle $R(u)$ associated to a node u. Let $u_1, \ldots, u_f$ denote the direct marked descendants of u. We can show that the set $P_0 = \texttt{square}(u) \setminus (\cup_{i=1}^{f} \texttt{square}(u_i))$ contains $O(d)$ points. Let L_0 denote the set of leaves in which points from P_0 are stored. There are at most d leaf nodes from L_0 between u_i and u_{i+1} for $1 \leq i < f$; there are at most d leaf descendants of u to the left of u_1 and at most d leaf descendants of u to the right of u_f. Hence the total number of leaves in L_0 does not exceed $(f + 2)d$. Since $R(u) \subseteq L_0$ and $f \leq 4$, $R(u)$ contains $O(d)$ points. □

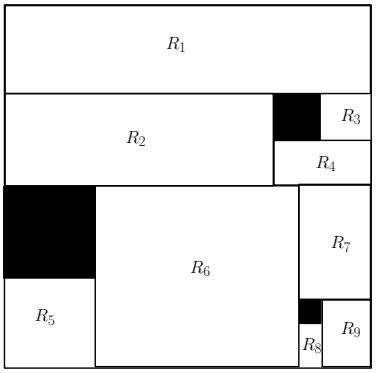

Fig. 2. Subdivision of a marked cell into rectangles. Cells corresponding to direct marked descendants are shown in black.

3 Orthogonal Range Reporting for Fat Boxes in 2-D

Data Structure. We divide the plane into canonical rectangles as described in Section 2. For every rectangle R in this subdivision we keep the list $L_x(R)$ of points in R sorted by their x-coordinates and the list $L_y(R)$ of points in R sorted by their y-coordinates. We also keep a data structure $D(R)$ that supports two-dimensional range reporting queries on points of R. Since R contains $O(\log n)$ points, we can implement $D(R)$ in $O(\log n)$ space so that queries are supported in $O(k)$ time. The data structure $D(R)$ will be described in Section A. We will denote by P the set of points stored in our data structure.

Orthogonal Range Queries. For simplicity we will consider the case when the query range is a square. Any fat rectangle can be represented as a union of $O(1)$ squares. Consider a query $Q = [a, b] \times [c, d]$. All canonical rectangles that intersect Q can be divided into three categories: (i) corner rectangles that contain a corner of Q (ii) rectangles that cut one side of Q or are completely contained in Q; such rectangles will be called internal rectangles (iii) rectangles that cross two opposite sides of Q, but do not contain corners of Q; we say that such rectangles are spanning rectangles or that type (iii) rectangles span Q. See Fig. 3.

Lemma 2. *If a rectangle $Q = [a, b] \times [c, d]$ is a square, then Q is spanned by $O(1)$ canonical rectangles.*

Proof: Suppose that a canonical rectangle $R(u)$, associated to a node u, spans Q. Then either (i) $\texttt{square}(u)$ contains two corners of Q and Q is not contained in $\texttt{square}(u')$ for any descendant u' of u, or (ii) $\texttt{square}(u)$

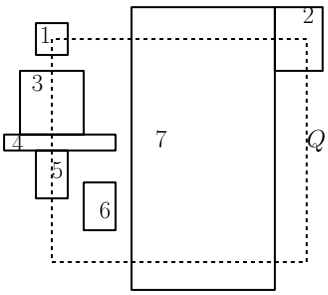

Fig. 3. Examples of different rectangles with respect to a query Q. Rectangles 1 and 2 are corner rectangles, rectangles 3, 4, 5, 6 are internal rectangles, and rectangle 7 spans Q.

contains Q but Q is not contained in $\texttt{square}(u')$ for any descendant u' of u.

If Q is contained in $\texttt{square}(u)$ and at least one rectangle $R(u)$ spans Q, then Q is not contained in $\texttt{square}(u')$ for any descendant u' of u. Hence there is at most one cell that satisfies condition (i).

Suppose that $\texttt{square}(u)$ contains two corners of Q and some rectangle $R(u)$ spans Q. Let us assume w.l.o.g. that Q crosses the left side ℓ of $\texttt{square}(u)$. Let u' be some descendant of u. If $\texttt{square}(u')$ for a descendant u' of u does not touch the left side of $\texttt{square}(u)$, then the distance from ℓ to $\texttt{square}(u')$ is greater than or equal to the size of $\texttt{square}(u')$. Hence $\texttt{square}(u')$ does not contain two corners of Q. If $\texttt{square}(u')$ touches ℓ and contains two corners of Q, then there is no canonical rectangle $R(u)$ that spans Q. Thus there is at most one cell that satisfies condition (ii).

Since there is only one cell satisfying condition (i) and only one cell satisfying condition (ii), the total number of canonical rectangles that span Q is bounded by a constant.

□

A query range can overlap with a large number of internal rectangles. But we can find all internal rectangles R, such that $R \cap Q \cap P \neq \emptyset$ by answering a range reporting query on a set P' (defined below) that contains $O(n/d)$ representative points for $d = \log n$. It was shown in Lemma 2 that a square is intersected by $O(1)$ spanning rectangles. There are at most four corner rectangles for any query range Q. Since there is a constant number of corner rectangles and spanning rectangles, we can process all of them in constant time. A more detailed description follows.

We can identify all internal rectangles (type (ii) rectangles) that contain at least one point from $P \cap Q$ as follows. For every canonical rectangle,

we keep its topmost point, its lowermost point, its leftmost point, and its rightmost point in the set P'. P' contains $O(n/d)$ points. We keep P' in the data structure D' that supports orthogonal range reporting queries in $O(\log\log U + k)$ time [7]. D' uses space $O(n' \log^{\varepsilon} n')$, where n' is the number of points in P'. Since $n' = O(n/d)$, D' uses space $O(n)$. If $rect(u)$ is an internal rectangle and $rect(u) \cap Q \cap P \neq \emptyset$, then at least one of its extreme points is in Q. We can find all such rectangles by answering the same query Q on the set P'. For every reported point p, we examine the canonical rectangle R_p that contains p.

There are at most four corner rectangles. We can find corner rectangles by keeping all canonical rectangles in the point location data structure of Chan [6]. For each corner point q of Q, we identify the rectangle R_q that contains q in $O(\log\log U)$ time.

Rectangles that span Q are the most difficult to deal with. All points of a spanning rectangle can be outside of Q. It is not clear how we can find spanning rectangles R such that $R \cap Q \cap P \neq \emptyset$. Existence of these rectangles is the reason why our method cannot be extended to the general case of the orthogonal range reporting. However, by Lemma 2, a square query range Q is spanned by $O(1)$ canonical rectangles from our subdivision. All rectangles that span Q can be found as follows. For a rectangle R we denote by $\texttt{left}(R)$, $\texttt{right}(R)$, $\texttt{bot}(R)$, and $\texttt{top}(R)$ the lower and upper bounds of its horizontal and vertical projections; that is, $R = [\texttt{left}(R), \texttt{right}(R)] \times [\texttt{bot}(R), \texttt{top}(R)]$. If a rectangle R spans Q, then at least one side of R spans Q. That is, R satisfies one of the following conditions: (i) $\texttt{left}(R) \leq a$, $\texttt{right}(R) \geq b$, and $c \leq \texttt{top}(R) \leq d$; (ii) $\texttt{left}(R) \leq a$, $\texttt{right}(R) \geq b$, and $c \leq \texttt{bot}(R) \leq d$; (iii) $a \leq \texttt{left}(R) \leq b$, $\texttt{bot}(R) \leq c$, and $\texttt{top}(R) \geq d$; (iv) $a \leq \texttt{right}(R) \leq b$, $\texttt{bot}(R) \leq c$, and $\texttt{top}(R) \geq d$. We keep information about every rectangle in four three-dimensional data structures. The data structure $\mathcal{R}_1$ contains a tuple $(\texttt{left}(R), \texttt{right}(R), \texttt{top}(R))$ for every canonical rectangle R. $\mathcal{R}_1$ can find all R that satisfy $\texttt{left}(R) \leq a$, $\texttt{right}(R) \geq b$, and $c \leq \texttt{top}(R) \leq d$. The data structure $\mathcal{R}_2$ contains a tuple $(\texttt{left}(R), \texttt{right}(R), \texttt{bot}(R))$ for every canonical rectangle R. $\mathcal{R}_2$ can find all R that satisfy $\texttt{left}(R) \leq a$, $\texttt{right}(R) \geq b$, and $c \leq \texttt{bot}(R) \leq d$. Data structures $\mathcal{R}_3$ and $\mathcal{R}_4$ contain tuples $(\texttt{left}(R), \texttt{bot}(R), \texttt{top}(R))$ and $(\texttt{right}(R), \texttt{bot}(R), \texttt{top}(R))$ respectively for every canonical rectangle R. $\mathcal{R}_3$ supports queries $a \leq \texttt{left}(R) \leq b$, $\texttt{bot}(R) \leq c$, and $\texttt{top}(R) \geq d$; $\mathcal{R}_4$ supports queries $a \leq \texttt{right}(R) \leq b$, $\texttt{bot}(R) \leq c$, and $\texttt{top}(R) \geq d$. Queries supported by data structures $\mathcal{R}_i$ are a special case of three-dimensional orthogonal range reporting queries, called 4-sided queries (the query range is bounded on

four sides). Using the result of Chan et al. [7], we can answer such queries in $O(\log\log U + k)$ time using $O(n' \log^{\varepsilon} n)$ space where $n' = O(n/d)$ is the number of tuples in $\mathcal{R}_i$. If a rectangle R is returned by a query to $\mathcal{R}_i$, then R spans Q or R contains two corners of Q. If Q is a square, then we can answer all queries on $\mathcal{R}_i$ described above and identify all canonical rectangles that span Q in $O(\log\log U + f) = O(\log\log U)$ time, where $f = O(1)$ is the number of canonical rectangles that span Q.

For every corner or spanning rectangle R, we find all points in $R \cap Q$ using the data structure $D(R)$. Since the total number of corner and spanning rectangles is bounded by $O(1)$, we can find all relevant points in $O(k)$ time. Using data structure D' we can find all internal rectangles in $O(\log\log U + n_I)$ time where n_I is the number of internal rectangles. For every internal rectangle R_I we traverse the list of points in $L_x(R_I)$ or $L_y(R_I)$ and report all points in $R_I \cap Q$ in time $O(k_I)$ where $k_I = |R_I \cap Q|$. The result of this section can be summed up as follows.

Theorem 1. *There is a linear-space data structure that answers orthogonal range reporting queries in $O(\log\log U + k)$ time provided the query range $Q = [a, b] \times [c, d]$ is a fat rectangle.*

4 Orthogonal Range Reporting for Fat Boxes in 3-D

In this section, we describe a data structure for 3-d orthogonal range reporting for fat query boxes, by adopting a recursive grid approach. Nonuniform grids have been used in previous range searching data structures by Alstrup, Brodal, and Rauhe [2] and Chan, Larsen, and Patrascu [7], but we use uniform grids instead. Also, the way we use recursion is a little different, and more closely resembles the recursion from van Emde Boas trees. Each node in our recursive structure is augmented with a general 5-sided range reporting structure; thus, our solution can be viewed as a reduction from fat 6-sided range searching to 5-sided range searching.

The data structure. Let P be a given set of n points in $[U]^3$, where $[U]$ denotes $\{0, 1, \ldots, U-1\}$. Let r be a parameter (a function of U) to be chosen later. Divide $[U]^3$ into r^3 *grid cells*, each a cube of the form $\{(x, y, z) : (U/r)i \le x < (U/r)(i+1),\ (U/r)j \le y < (U/r)(j+1),\ (U/r)k \le z < (U/r)(k+1)\}$ for some $i, j, k \in [r]$. We call (i, j, k) the label of such a grid cell. A *grid slab* refers to a region of the form $\{(x, y, z) : (U/r)i \le x < (U/r)(i+1)\}$, $\{(x, y, z) : (U/r)j \le y < (U/r)(j+1)\}$, or $\{(x, y, z) : (U/r)k \le z < (U/r)(k+1)\}$. A *grid-aligned box* refers to a box

whose x-, y-, and z-coordinates are all multiples of U/r. We construct our data structure as follows:

A. For each nonempty grid cell γ, recursively build a data structure for $P \cap \gamma$; also store $P \cap \gamma$ as a linked list.
B. Let Γ be the set of all nonempty grid cells. Recursively build a data structure for the labels of Γ.
C. For each grid slab σ, build Chan, Larsen, and Patrascu's data structure [7] for $P \cap \sigma$ for 3-d 5-sided queries, which requires $O(n \log^{\varepsilon} n)$ words of space and $O(\log \log U)$ query time[4].

Analysis of space. Since we use a uniform grid, we will represent the space usage and query time as functions of the universe size U. Let $s(U)$ be the *amortized* space complexity of our data structure in bits, i.e., the total space complexity in bits divided by the number of points n. Item A of the data structure requires at most $s(U/r)$ bits per point, since after translation, each grid cell becomes $[U/r]^3$. This ignores the space for the linked lists, which require a total of $O(n \log U)$ bits. Item B requires at most $s(r)$ bits per point, since the labels lie in $[r]^3$. Item C requires a total of $O(n \log^{\varepsilon} n \log U) \leq O(n \log^{1+\varepsilon} U)$ bits (since $n \leq U^3$). Thus,

$$s(U) \;\leq\; s(U/r) + s(r) + O(\log^{1+\varepsilon} U).$$

Query algorithm. We consider the case when the query range is an (axis-parallel) cube; any fat query box can be expressed as a union of $O(1)$ cubes. Given a query cube Q, we report all points of P in Q as follows:

1. If Q is completely contained in a grid cell γ, then recursively report all points of $P \cap \gamma$ in Q. Otherwise:
2. Decompose Q into (at most) one grid-aligned cube Q' and (at most) six other boxes $Q_1, \ldots, Q_6$, where each Q_i is a 5-sided box in a grid slab σ_i. (See Figure 4 for an analogous 2-d depiction.)
3. Recursively report all grid cells of Γ in Q'. For each reported grid cell $\gamma \in \Gamma$, report all points in the linked list $P \cap \gamma$.
4. For each $i \in \{1, \ldots, 6\}$, report all points of $P \cap \sigma_i$ in Q_i.

Analysis of query time. Let $t(U)$ denote the running time of the query algorithm, excluding the outputting cost (which is $O(k)$ for k output points). Step 1 takes $t(U/r)$ time. The recursive call in step 3 takes $t(r)$ time. Step 2 takes $O(\log \log U)$ time. Thus,

$$t(U) \;\leq\; \max\left\{t(U/r) + O(1),\; t(r) + O(\log \log U)\right\}.$$

[4] For simplicity, we ignore the time needed to output points in this section.

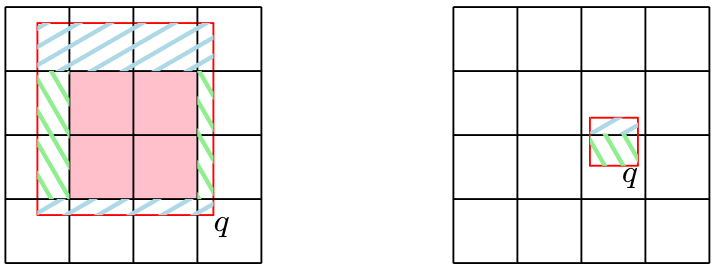

Fig. 4. In 2-d, if a query square Q is not completely contained in any grid cell, it can be decomposed into one grid-aligned square and four 3-sided rectangles in four grid slabs (shown in the left), or just two 3-sided rectangles (shown in the right). Similarly, in 3-d, if a query cube Q is not completely contained in any grid cell, it can be decomposed into one grid-aligned cube and six 5-sided rectangles in six grid slabs, or just two 5-sided rectangles.

We can eliminate the $O(1)$ in the first term of the max by the following idea: Consider the tree formed by expanding the recursion due to item A (treating the recursive structures from item B as secondary structures at the nodes of the main tree). Then we can jump to the first node of the tree at which Q is not completely contained in a grid cell, in $O(1)$ time by an LCA operation in the tree (actually, because the tree is perfectly balanced, for our choice of r (see below), the LCA operation can be simulated by standard arithmetic and bitwise-logical operations on the coordinates of Q).

Conclusion. Setting $r = U^{1/b}$ for a fixed parameter b gives

$$\begin{aligned} s(U) &\le s(U^{1-1/b}) + s(U^{1/b}) + O(\log^{1+\varepsilon} U) \\ t(U) &\le \max\left\{t(U^{1-1/b}),\ t(U^{1/b}) + O(\log\log U)\right\}, \end{aligned}$$

which solves to $s(U) = O(b\log^{1+\varepsilon} U)$ and $t(U) = O(\log_b \log U \cdot \log\log U)$. The outputting cost goes up to $O(k \log_b \log U)$, since each point may be reported $O(\log_b \log U)$ times.

Setting $b = \log^{\varepsilon} U$ gives $O(\log^{1+2\varepsilon} U)$ amortized space in bits and $O(\log\log U + k)$ query time. Readjusting ε by a half, we conclude:

Theorem 2. *We can store n points in $[U]^3$ in a data structure with $O(n\log^{\varepsilon} U)$ words of space so that we can report all k points in any query fat box in $O(\log\log U + k)$ time.*

Remark. The above theorem can't be improved with current state of the art, because 3-d 4-sided range reporting reduces to our problem and

the current best data structure for the former requires $O(n \log^{\varepsilon} n)$ space and $O(\log\log U + k)$ query time. To see the reduction, note that a 4-sided box $[x_1, x_2) \times (-\infty, y) \times (-\infty, z)$ contains the same points as the cube $[x_1, x_2) \times [y - (x_2 - x_1), y) \times [z - (x_2 - x_1), z)$, assuming that $x_2 - x_1 > U$. The assumption can be guaranteed after stretching the x-axis by a factor of U (so that the universe is now $[U^2] \times [U] \times [U]$).

References

1. P. Afshani. On dominance reporting in 3d. In *Proc. 16th Annual European Symposium on Algorithms(ESA)*, pages 41–51, 2008.
2. S. Alstrup, G. S. Brodal, and T. Rauhe. New data structures for orthogonal range searching. In *Proc. 41st Annual Symposium on Foundations of Computer Science, (FOCS 2000)*, pages 198–207, 2000.
3. L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. 18th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS)*, pages 346–357, 1999.
4. B. Aronov, M. de Berg, and C. Gray. Ray shooting and intersection searching amidst fat convex polyhedra in 3-space. *Comput. Geom.*, 41(1-2):68–76, 2008.
5. J. L. Bentley and H. A. Maurer. Efficient worst-case data structures for range searching. *Acta Inf.*, 13:155–168, 1980.
6. T. M. Chan. Persistent predecessor search and orthogonal point location on the word RAM. *ACM Trans. Algorithms*, 9(3):22, 2013.
7. T. M. Chan, K. G. Larsen, and M. Patrascu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry, (SoCG 2011)*, pages 1–10, 2011.
8. B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.
9. B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, 1988.
10. B. Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, 2:113–126, 1987.
11. M. de Berg and C. Gray. Vertical ray shooting and computing depth orders for fat objects. *SIAM J. Comput.*, 38(1):257–275, 2008.
12. A. Efrat, M. J. Katz, F. Nielsen, and M. Sharir. Dynamic data structures for fat objects and their applications. *Computational Geometry*, 15(4):215 – 227, 2000.
13. H. N. Gabow, J. L. Bentley, and R. E. Tarjan. Scaling and related techniques for geometry problems. In *Proc. 16th Annual ACM Symposium on Theory of Computing (STOC 1984)*, pages 135–143, 1984.
14. J. Iacono and S. Langerman. Dynamic point location in fat hyperrectangles with integer coordinates. In *Proc. 12th Canadian Conference on Computational Geometry*, 2000.
15. M. Karpinski and Y. Nekrich. Space efficient multi-dimensional range reporting. In *Proc. 15th Annual International Conference on Computing and Combinatorics (COCOON 2009)*, pages 215–224, 2009.
16. M. J. Katz. 3-d vertical ray shooting and 2-d point enclosure, range searching, and arc shooting amidst convex fat objects. *Computational Geometry*, 8(6):299 – 316, 1997.

17. M. v. Kreveld and M. Löffler. *Range Searching*, pages 1–7. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
18. M. Lewenstein, Y. Nekrich, and J. S. Vitter. Space-efficient string indexing for wildcard pattern matching. In *Proc. 31st International Symposium on Theoretical Aspects of Computer Science (STACS 2014)*, pages 506–517, 2014.
19. E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.
20. G. Navarro and Y. Nekrich. Top-k document retrieval in optimal time and linear space. In *Proc. 23rd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012*, pages 1066–1077, 2012.
21. Y. Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry, (SoCG)*, pages 344–353, 2007.
22. Y. Nekrich. Space efficient dynamic orthogonal range reporting. *Algorithmica*, 49(2):94–108, 2007.
23. Y. Nekrich. *Orthogonal Range Searching on Discrete Grids*, pages 1–6. Springer US, Boston, MA, 2008.
24. S. Rahul. Improved bounds for orthogonal point enclosure query and point location in orthogonal subdivisions in $\mathbb{R}^3$. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2015)*, pages 200–211, 2015.
25. D. E. Vengroff and J. S. Vitter. Efficient 3-d range searching in external memory. In *Proc. 28th Annual ACM Symposium on the Theory of Computing (STOC 1996)*, pages 192–201, 1996.

A Orthogonal Range Reporting on a Small Set of Points

In this section we show how two-dimensional orthogonal range reporting on a set of $d = O(\log n)$ points can be supported in $O(k)$ time. Our data structure uses space $O(d)$, but needs an additional universal look-up table of size $o(n)$. That is, we can keep many instances of our data structure for different point sets and all instances can use the same look-up table.

Lemma 3. *If a set P contains $d = O(\log n)$ points, then we can keep P in a linear-space data structure $D(P)$ that answers two-dimensional range reporting queries in $O(k)$ time. This data structure relies on a universal look-up table of size $o(n)$.*

Proof: First we observe that we can answer a query on a set P' that contains at most $d' = (1/4)\log n/\log\log n$ points using a look-up table of size $o(n)$. Suppose that all points in P' have positive integer coordinates bounded by d'. There are $2^{d'\log d'}$ combinatorially different sets P'. For every instance of P', we can ask $(d')^4$ different queries and the answer to each query consists of $O(d')$ points. Hence the total space needed to keep answers to all possible queries on all instances of P' is $O(2^{(\log d')d'}(d')^5) = o(n)$ points. The general case (when point coordinates are arbitrary integers) can be reduced to the case when point coordinates are bounded by d' using reduction to rank space [13, 2].

A query on P can be reduced to $O(1)$ queries on sets that contain $O(d')$ points using the grid approach [7, 2]. The set of points P is divided into $4 \log d$ columns C_i and $4 \log d$ rows R_j so that every row and every column contains $(1/4)d/\log d$ points. Hence we can support range reporting queries on points in a row/column using the look-up table approach described above. The top set P_t contains a meta-point (i, j) iff the intersection of the i-th column and the j-th row is not empty, $R_j \cap C_i \neq \emptyset$. Since P_t contains $O(\log^2 d) = o(d')$ points, we can also support queries on P_t in $O(k)$ time. For each meta-point (i, j) in P_t we store the list of points L_{ij} contained in the intersection of the i-th column and the j-th row, $L_{ij} = C_i \cap R_j \cap P$.

Consider a query $Q = [a, b] \times [c, d]$. If Q is contained in one column or one row, we answer the query using the data structure for that column/row. Otherwise we identify the rows R_l and R_t that contain c and d respectively (i.e., the line $y = c$ is contained in R_b and the line $y = d$ is contained in R_t). We also identify the columns C_f and C_r containing a and b. We report all points in $Q \cap C_l$, $Q \cap C_r$, $Q \cap R_b$ and $Q \cap R_t$. We find all meta-points (i, j) in P_t such that $f < i < r$ and $l < j < t$; for every found (i, j) we report all points in L_{ij}. □

B Rectangle Stabbing in Three Dimensions

We consider the scenario when a set of fat three-dimensional rectangles is stored in a data structure. Corners of rectangles lie on an integer grid of size U. Given a query point q with integer coordinates, we must report all rectangles that contain q.

Our construction is based on an uncompressed octree T. The set $P(u)$ for an octree node u contains all rectangles R such that: (i) R contains at least one corner of $\texttt{cell}(u)$, but (ii) R does not contain any corners of $\texttt{cell}(w)$ for any ancestor w of u. We will say that a node u is *relevant* for a rectangle R if $R \in P(u)$.

Lemma 4. *Every fat rectangle R is stored in $O(1)$ sets $P(u)$.*

Proof: Any fat three-dimensional rectangle R can be divided into $O(1)$ cubes $Q_1, Q_2, \ldots, Q_f$. If R satisfies conditions (i) and (ii) with respect to some node u, then there is at least one cube Q_i that satisfies conditions (i) and (ii) for some descendant v of u. Therefore it is sufficient to show that any cube Q_i is relevant for $O(1)$ nodes of T.

Let s denote the size of a cube Q_i. There is exactly one cell size ℓ, such that $s < \ell \leq 2s$. Suppose that a cube Q_i is entirely contained in

a size-ℓ cell of some node u. Since $s \geq \ell/2$, Q_i contains the common corner ν of all u's children. Q_i does not contain corners of u or any of its ancestors. Hence Q_i is relevant for at most eight children u_i of u. Now suppose that Q_i contains a corner ν of some size-ℓ cell, $\texttt{cell}(u'_1)$. Since $s < \ell$, Q_i intersects at most eight cells of size ℓ that share the common corner ν. Every cell $\texttt{cell}(u'_i)$ with corner ν that intersects Q_i satisfies condition (i). Among all ancestors of u'_i there is exactly one node that satisfies both conditions (i) and (ii). Hence every Q_i is relevant for at most eight nodes. □

For each node u we keep rectangles $R \in P(u)$ in data structures that answer three-dimensional dominance queries. There is one data structure for every corner ν of $\texttt{cell}(u)$. If a rectangle R is relevant for a node u, then $\texttt{cell}(u)$ contains one corner μ_R of R[5]. If a rectangle $R \in P(u)$ contains a corner ν of $\texttt{cell}(u)$, then we store R in the data structure $D_\nu(u)$ that supports stabbing queries for points $q \in \texttt{cell}(u)$. Since rectangles $R \in D_\nu$ contain the corner ν of $\texttt{cell}(u)$, the rectangle stabbing query is equivalent to a three-dimensional dominance query in this case. Suppose, for example, that ν is the corner with the smallest x-, y-, and z-coordinates in $\texttt{cell}(u)$. Then reporting rectangles $R \in D_\nu$ containing the point q is equivalent to reporting corners μ_R such that $x(\mu_r) \geq x(q)$, $y(\mu_r) \geq y(q)$, and $z(\mu_r) \geq z(q)$. See Figure 5 for an example in the 2-d case. The linear-space data structure of Chan [6] answers three-dimensional point reporting queries in $O(\log\log U + k)$ time. Hence D_ν supports stabbing queries for points $q \in \texttt{cell}(u)$ in $O(\log\log U + k)$ time.

Given a query point q, we visit all nodes u on the path from the root to a leaf node that contains q. In every visited node u we answer eight dominance queries and thus report all rectangles $R \in P(u)$ that contain q. The total time to answer a query is $O(\log U \log\log U + k)$. Every rectangle stabbed by q is relevant for some visited node u. Therefore our procedure correctly reports all rectangles stabbed by q. The data structure uses space $O(n)$ because each rectangle is stored a constant number of times.

Theorem 3. *There is an $O(n)$-space data structure that answers three-dimensional rectangle stabbing queries for a set of fat rectangles on a $[U]^3$ grid. Queries are supported in $O(\log U \log\log U + k)$ time.*

[5] To avoid tedious details we assume that every rectangle contains exactly one corner of $\texttt{cell}(u)$. The case when a rectangle $R \in P(u)$ contains two or four corners of $\texttt{cell}(u)$ can be handled in a similar way.

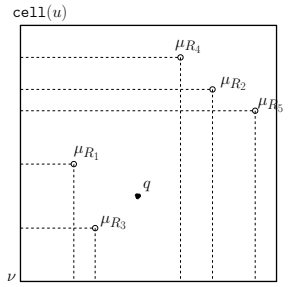

Fig. 5. Relevant rectangles for a quadtree in two dimensions. Relevant rectangles containing the corner ν are shown with dashed lines, cell boundaries are shown with solid lines. Point q is stabbed by R_2, R_4, and R_5 because $x(\mu_{R_j}) \geq x(q)$ and $y(\mu_{R_j}) \geq y(q)$ for $j = 2, 4, 5$.