Space-Efficient Multi-Dimensional Range Reporting

Marek Karpinski⋆ and Yakov Nekrich⋆⋆

Dept. of Computer Science, University of Bonn.

Abstract. We present a data structure that supports three-dimensional range reporting queries in $O(\log\log U + (\log\log n)^3 + k)$ time and uses $O(n\log^{\varepsilon} n)$ space, where U is the size of the universe, k is the number of points in the answer, and ε is an arbitrary constant. This result improves over the data structure of Alstrup, Brodal, and Rauhe (FOCS 2000) that uses $O(n\log^{1+\varepsilon} n)$ space and supports queries in $O(\log n + k)$ time and the data structure of Nekrich (SoCG'07) that uses $O(n\log^4 n)$ space and supports queries in $O(\log\log U + (\log\log n)^2 + k)$ time. Our result allows us to significantly reduce the space usage of the fastest previously known static and incremental d-dimensional data structures, $d \geq 3$, at a cost of increasing the query time by a negligible $O(\log\log n)$ factor.

1 Introduction

The range reporting problem is to store a set of d-dimensional points P in the data structure, so that for a query rectangle Q all points in $Q \cap P$ can be reported. In this paper we significantly improve the space usage and pre-processing time of the fastest previously known static and semi-dynamic data structures for orthogonal range reporting with only a negligible increase in the query time.

The range reporting is extensively studied at least since 1970s; the history of this problem is rich with different trade-offs between query time and space usage. Static range reporting queries can be answered in $O(\log^d n + k)$ time and $O(n\log^{d-1} n)$ space using range trees [4] known since 1980; here and further k denotes the number of points from P in the query rectangle. The query time can be reduced to $O(\log^{d-1} n + k)$ time by applying the fractional cascading technique of Chazelle and Guibas [8] designed in 1985. The space usage was further improved by Chazelle [6]. In 90s, Subramanian and Ramaswamy [12] and Bozanis, Kitsios, Makris, and Tsakalidis [5] showed that d-dimensional queries can be answered in $\widetilde{O}(\log^{d-2} n + k)$ time[1] at a cost of higher space usage:

⋆ Email marek@cs.uni-bonn.de.

⋆⋆ Email yasha@cs.uni-bonn.de.

[1] We define $\widetilde{O}(f(n)) = O(f(n)\log^c(f(n)))$ for a constant c.

Source	Query Time	Space
[4]	$O(\log^d n + k)$	$O(n \log^{d-1} n)$
[8]	$O(\log^{d-1} n + k)$	$O(n \log^{d-1} n)$
[6]	$O(\log^{d-1} n + k)$	$O(n \log^{d-2+\varepsilon} n)$
[12]	$O(\log^{d-2} n \log^{**} n + k)$	$O(n \log^{d-1} n)$
[5]	$O(\log^{d-2} n + k)$	$O(n \log^{d} n)$
[2]	$O(\log^{d-2} n/(\log\log n)^{d-3} + k)$	$O(n \log^{d-2+\varepsilon} n)$
[11]	$O(\log^{d-3} n/(\log\log n)^{d-5} + k)$	$O(n \log^{d+1+\varepsilon} n)$
[1]†	$O(\log^{d-3} n/(\log\log n)^{d-5} + k)$	$O(n \log^{d+\varepsilon} n)$
This paper	$O(\log^{d-3} n/(\log\log n)^{d-6} + k)$	$O(n \log^{d-2+\varepsilon} n)$

Table 1. Data structures in $d > 3$ dimensions; † indicates that a data structure is randomized. We define $\log^*(n) = \min\{\, t \mid \log^{(t)} n \le 1 \,\}$ and $\log^{**} n = \min\{\, t \mid \log^{*(t)} n \le 1 \,\}$ where $\log^{*(t)} n$ denotes computing $\log^*$ t times.

their data structures use $O(n \log^{d-1} n)$ and $O(n \log^d n)$ space respectively. Alstrup, Brodal, and Rauhe [2] designed a data structure that answers queries in $\widetilde{O}(\log^{d-2} n + k)$ time and uses $O(n \log^{d-2+\varepsilon} n)$ space for an arbitrary constant $\varepsilon > 0$. Recently, Nekrich [11] reduced the query time by $\widetilde{O}(\log n)$ factor and presented a data structure that answers queries in $O(\log^{d-3} n/(\log\log n)^{d-4} + k)$ time for $d > 3$. Unfortunately, the data structure of [11] uses $O(n \log^{d+1+\varepsilon} n)$ space. Very recently, Afshani [1] reduced the space usage to $O(n \log^{d+\varepsilon} n)$; however his data structure uses randomization. In this paper we present a data structure that matches the space efficiency of [2] at a cost of increasing the query time by a negligible $O(\log\log n)$ factor: our data structure supports queries in $O(\log^{d-3} n/(\log\log n)^{d-5} + k)$ time and uses $O(n \log^{d-2+\varepsilon} n)$ space for $d > 3$. See Table 1 for a more precise comparison of different results.

Our result for d-dimensional range reporting is obtained as a corollary of the three-dimensional data structure that supports queries in $O(\log\log U + (\log\log n)^3 + k)$ time and uses $O(n \log^{1+\varepsilon} n)$ space. Our three-dimensional data structure is to be compared with the data structure of [2] that also uses $O(n \log^{1+\varepsilon} n)$ space but answers queries in $O(\log n + k)$ time and the data structure of [11] that answers queries in $O(\log\log U + (\log\log n)^3 + k)$ time but needs $O(n \log^4 n)$ space. See Table 2 for a more extensive comparison with previous results. A corollary of our result is an efficient semi-dynamic data structure that supports three-dimensional queries in $\widetilde{O}(\log n + k)$ time and insertions in $O(\log^5 n)$ time. Thus we improve the update time of the data structure from [11] that supports insertions in $O(\log^8 n)$ time.

If we are ready to pay penalties for each point in the answer, the space usage can be further reduced: we describe a data structure that uses $O(n\log^{d-2} n(\log\log n)^3)$ space and answers queries in $O(\log^{d-3} n(\log\log n)^3 + k\log\log n)$ time. We can also use this data structure to answer emptiness queries (to determine whether query rectangle Q contains points from P) and one-reporting queries (i.e. to report an *arbitrary* point from $P\cap Q$ if $P\cap Q\neq\emptyset$). This is an $\widetilde{O}(\log n)$ factor improvement in query time over the data structure from [2]. Other similar data structures are either slower or require higher penalties for each point in the answer.

Source	Query Time	Space
[6]	$O(\log^2 n + k)$	$O(n\log^{1+\varepsilon} n)$
[12]	$O(\log n\log^{**} n + k)$	$O(n\log^2 n)$
[5]	$O(\log n + k)$	$O(n\log^3 n)$
[2]	$O(\log n + k)$	$O(n\log^{1+\varepsilon} n)$
[11]	$O(\log\log U + (\log\log n)^2 + k)$	$O(n\log^{4+\varepsilon} n)$
[1]†	$O(\log\log U + (\log\log n)^2 + k)$	$O(n\log^3 n)$
This paper	$O(\log\log U + (\log\log n)^3 + k)$	$O(n\log^{1+\varepsilon} n)$

Table 2. Three-dimensional data structures; † indicates that a data structure is randomized.

Throughout this paper, ε denotes an arbitrarily small constant, k denotes the number of points in the answer, and U denotes the size of the universe. If each point in the answer can be output in constant time, we will sometimes say that the query time is $O(f(n))$ (instead of $O(f(n) + k)$). In section 3 we describe a space efficient data structure for three-dimensional range reporting on a grid. In section 4 we describe a variant of our data structure that uses less space but needs $O(\log\log n)$ time to output each point in the answer.

2 Preliminaries

We use the same notation as in [13] to denote the special cases of three-dimensional range reporting queries: a product of three half-open intervals will be called a (1,1,1)-sided query; a product of a closed interval and two half-open intervals will be called a (2,1,1)-sided query; a product of two closed intervals and one half-open interval (resp. three closed intervals) will be called a (2,2,1)-sided (resp. (2,2,2)-sided) query. Clearly

(1,1,1)-sided queries are equivalent to dominance reporting queries, and (2,2,2)-sided query is the general three-dimensional query. The following transformation is described in e.g. [13] and [12].

Lemma 1. *Let $1 \leq a_i \leq b_i \leq 2$ for $i = 1, 2, 3$. A data structure that answers (a_1, a_2, a_3) queries in $O(q(n))$ time, uses $O(s(n))$ space, and can be constructed in $O(c(n))$ time can be transformed into a data structure that answers (b_1, b_2, b_3) queries in $O(q(n))$ time, uses $O(s(n) \log^t n)$ space and can be constructed in $O(c(n) \log^t n)$ time for $t = (b_1 - a_1) + (b_2 - a_2) + (b_3 - a_3)$.*

We will also need the following folklore result:

Lemma 2. *There exists a $O(n^{1+\varepsilon})$ space data structure that supports range reporting queries on a d-dimensional grid of size n for any constant d in $O(k)$ time.*

Proof: One dimensional range reporting queries on a grid of size n can be answered in $O(k)$ time using a trie with node degree n^ε. Using range trees [4] with node degree ρ we can transform a d-dimensional $O(s(n))$ space data structure into a $(d+1)$-dimensional data structure that uses $O(s(n)h(n) \cdot \rho)$ space and answers range reporting queries in $O(q(n)h(n))$ time, where $h(n) = \log n / \log \rho$ is the height of the range tree. Since $\rho = n^\varepsilon$, $h(n) = O(1)$. Hence, the query time does not depend on dimension and the space usage increases by a factor $O(n^\varepsilon)$ with each dimension. □

We use Lemma 2 to obtain a data structure that supports queries that are a product of a $(d-1)$-dimensional query on a universe of size $n^{1-\varepsilon}$ and a half-open interval. We will show in the next Lemma that such queries can be answered in $O(n)$ space and $O(1)$ time.

Lemma 3. *There exists a $O(n)$ space data structure that supports range reporting queries of the form $Q' \times [-\infty, x)$ where Q' is a $(d-1)$-dimensional query on $U_1 \times U_2 \times \ldots \times U_{d_1}$ and $|U_1| \cdot |U_2| \cdot \ldots \cdot |U_{d-1}| = O(n^{1-\varepsilon})$.*

Proof: There are $O(n^{1-\varepsilon})$ possible projections of points onto the first $d-1$ coordinates. Let $\min(p_1, \ldots, p_{d-1})$ denote the point with minimal d-th coordinate among all points whose first $d-1$ coordinates equal to $p_1, p_2, \ldots, p_{d-1}$. We store points $\min(p_1, \ldots, p_{d-1})$ for all $p_1 \in U_1$,$p_2 \in U_2$,…,$p_{d-1} \in U_{d-1}$ in a data structure M. Since M contains $O(n^{1-\varepsilon})$ points, we can use Lemma 2 and implement M in $O(n)$ space. For all possible $p_1 \in U_1$,$p_2 \in U_2$,…,$p_{d-1} \in U_{d-1}$ we also store a list $L(p_1, \ldots, p_{d-1})$ of points whose first $d-1$ coordinates are $p_1, \ldots, p_{d-1}$; points in $L(p_1, \ldots, p_{d-1})$ are sorted by their d-th coordinates. Given a query $Q = Q' \times [-\infty, x)$, we first answer Q using the data structure M. Then, for every point $p = (p_1, \ldots, p_{d-1}, p_d)$ found with help of M, we traverse the corresponding list $L(p_1, \ldots, p_{d-1})$ and report all points in this list whose last coordinate does not exceed x. $\square$

In several places of our proofs we will use the *reduction to rank space* technique [9, 6]. This technique allows us to replace coordinates of a point by its rank. Let P_x, P_y, and P_z be the sets of x, y-, and z-coordinates of points from P. For a point $p = (p_x, p_y, p_z)$, let $p' = (\text{rank}(p_x, P_x), \text{rank}(p_y, P_y), \text{rank}(p_z, P_z))$, where $\text{rank}(e, S)$ is defined as the number of elements in S that are smaller than or equal to e. A point p belongs to an interval $[a, b] \times [c, d] \times [e, f]$ if and only if a point p' belongs to an interval $[a', b'] \times [c', d'] \times [e', f']$ where $a' = \text{succ}(a, P_x)$, $b' = \text{pred}(b, P_x)$, $c' = \text{succ}(c, P_y)$, $d' = \text{pred}(d, P_y)$, $e' = \text{succ}(e, P_z)$, $f' = \text{pred}(f, P_z)$, and $\text{succ}(e, S)$ $(\text{pred}(e, S))$ denotes the smallest (largest) element in S that is greater (smaller) than or equal to e. Reduction to rank space can be used to improve the query time. Following [2], we can also use this technique to reduce the space usage: if a data structure contains m elements, reduction to rank space allows us to store each element in $O(\log m)$ bits.

3 Space-Efficient Three-Dimensional Data Structure

In this section we describe a data structure that supports three-dimensional range reporting queries in $O((\log\log n)^3 + \log\log U + k)$ time where U is the universe size and uses $O(n \log^{1+\varepsilon} n)$ space. Our data structure combines the recursive divide-and-conquer approach introduced in [2], the result of Lemma 3, and the transformation of (a_1, a_2, a_3)-queries into (b_1, b_2, b_3)-queries described in Lemma 1. We start with a description of a space-efficient modification of the data structure for (1,1,1)-sided queries. Then, we obtain data structures for $(2, 1, 1)$-sided and $(2, 2, 1)$-sided queries using the recursive divide-and-conquer and Lemma 3. Finally, we obtain the data structure that supports arbitrary queries using Lemma 1.

Lemma 4. *[11] Given a set of three-dimensional points P and a parameter t, we can construct in $O(n \log^3 n)$ time a $O(n)$ space data structure T that supports the following queries on a grid of size n:*
(i) for a given query point q, T determines in $O((\log\log n)^2)$ time whether q is dominated by at most t points of P
(ii) if q is dominated by at most t points from P, T outputs in $O(t + (\log\log n)^2)$ time a list L of $O(t)$ points such that L contains all points of P that dominate q.

As described in [11], Lemma 4 allows us to answer (1,1,1)-sided queries in $O((\log\log n)^2)$ time and $O(n\log n)$ space. We can reduce the space usage to $O(n\log\log n)$ using an idea that is also used in [1]. In the proofs of Lemmas 5, 6, and 7, as well as Theorem 1, we assume that all point coordinates belong to a universe of size n. Reduction to rank space technique described in section 2 allows us to transform a data structure on a grid of size n into a data structure on a grid of size U, so that the query time increases by an additive term $O(\log\log U)$ and the space usage is not increased.

Lemma 5. *There exists a data structure that answers (1,1,1)-sided queries in $O((\log\log U + \log\log n)^2 + k)$ time, uses $O(n\log\log n)$ space, and can be constructed in $O(n\log^3 n\log\log n)$ time.*

Proof: For each parameter $t = 2^{2i}$, $i = 1, 2, \ldots, \log\log n/2$, we construct a data structure T_i of Lemma 4. Given a query point q, we examine data structures $T_1, T_2, \ldots, T_{\log\log n/2}$ and check whether q is dominated by more than 2^{2i} points for $i = 1, 2, \ldots, \log\log n/2$. Thus we identify i, such that q is dominated by more than 2^{2i} and less than 2^{2i+2} points or determine that q is dominated by at least $\log n$ points. In the former case, we generate in $O((\log\log n)^2 + 2^{2i+2})$ time a list L that contains all points that dominate q. Then, we examine all points in L and output all points that dominate q in $O(2^{2i+2})$ time. The total query time is $O(i \cdot (\log\log n)^2 + 2^{2i+2}) = O((\log\log n)^2 + k)$, because $k \geq 2^{2i}$. If q is dominated by at least $\log n$ points of P, we can use a linear space data structure with $O(\log n)$ query time, e.g. the data structure of Chazelle and Edelsbrunner [7], to answer the query in $O(\log n + k) = O(k)$ time.

Since each data structure T_i uses linear space, the space usage of the described data structure is $O(n\log\log n)$. □

Lemma 6. *There exists a data structure that answers (2,1,1)-sided queries in $O(\log\log U + (\log\log n)^3 + k)$ time, uses $O(n\log^\varepsilon n)$ space, and can be constructed in $O(n\log^3 n\log\log n)$ time.*

Proof: We divide the grid into x-slices $X_i = [x_{i-1}, x_i] \times n \times n$ and y-slices $Y_j = n \times [y_{j-1}, y_j] \times n$, so that each x-slice contains $n^{1/2+\gamma}$ points and each y-slice contains $n^{1/2+\gamma}$ points; the value of a constant γ will be specified below. The cell C_{ij} is the intersection of the i-th x-slice and the j-th y-slice, $C_{ij} = X_i \cap Y_j$. The data structure D_t contains a point (i, j, z) for each point $(x, y, z) \in P \cap C_{ij}$. Since the first two coordinates of points in D_t are bounded by $n^{1/2-\gamma}$, D_t uses $O(n)$ space and supports (2,1,1)-sided queries in constant time by Lemma 3. For each x-slice X_i there are two data structures that support two types of (1,1,1)-sided queries, open in $+x$ and in $-x$ directions. For each y-slice Y_j, there is a data structure that supports $(1,1,1)$-sided queries open in $+y$ direction. For each y-slice Y_j and for each x-slice X_i there are recursively defined data structures. Recursive subdivision stops when the number of elements in a data structure is smaller than a predefined constant. Hence, the number of recursion levels is $v \log\log n$ for $v = \log_{\frac{2}{1+2\gamma}} 2$.

Given a query $Q = [a,b] \times (-\infty, c] \times (-\infty, d]$ we identify the indices i_1, i_2, and j such that projections of all cells C_{ij}, $i_1 < i < i_2$, $j < j_1$, are entirely contained in $[a,b] \times (-\infty, c]$. Let $a_0 = x_{i_1-1}$, $b_0 = x_{i_2-1}$, and $c_0 = y_{j_1-1}$. The query Q can be represented as $Q = Q_1 \cup Q_2 \cup Q_3 \cup Q_4$, where $Q_1 = [a_0, b_0] \times (-\infty, c_0] \times (-\infty, d]$, $Q_2 = [a, a_0) \times (-\infty, c] \times (-\infty, d]$, $Q_3 = (b_0, b] \times (-\infty, c] \times (-\infty, d]$, and $Q_4 = [a_0, b_0] \times (c_0, c] \times (-\infty, d]$. Query Q_1 can be answered using A_t. Queries Q_2 and Q_3 can be represented as $Q_2 = ([-\infty, a_0) \times (-\infty, c] \times (-\infty, d]) \cap X_{i_1}$ and $Q_3 = ((-\infty, b] \times (-\infty, c] \times (-\infty, d]) \cap X_{i_2}$; hence, Q_2 and Q_3 are equivalent to $(1,1,1)$-sided queries on x-slices X_{i_1} and X_{i_2}. The query Q_4 can be answered by a recursively defined data structure for the y-slice Y_{j_1} because $Q_4 = ([a_0, b_0] \times (-\infty, c] \times (-\infty, d]) \cap Y_{j_1}$. If $i_1 = i_2$ and the query Q is contained in one x-slice, then Q is processed by a recursively defined data structure for the corresponding x-slice. Thus a query is reduced to one special case that can be processed in constant time, two $(1,1,1)$-sided queries, and one (2,1,1)-sided query answered by a data structure that contains $n^{1/2+\gamma}$ elements.

Queries Q_2 and Q_3 can be answered in $O((\log\log n)^2)$ time, the query Q_1 can be answered in constant time. The query Q_4 is answered by a recursively defined data structure that contains $O(n^{1/2+\gamma})$ elements. If $i_1 = i_2$ or $j_1 = 1$, i.e. if Q is entirely contained in one x-slice or one y-slice, then the query is answered by a data structure for the corresponding slice that contains $O(n^{1/2+\gamma})$ elements. Hence, the query time $q(n) = O((\log\log n)^2) + q(n^{1/2+\gamma})$ and $q(n) = O((\log\log n)^3)$.

The data structure consists of $O(\log\log n)$ recursion levels. The total number of points in all data structures on the i-th recursion level is $2^i n$. Hence all data structures on the i-th recursion level require $O(2^i n \log n)$ bits of space. The space usage can be reduced by applying the reduction to rank space technique [9, 6]. As explained in section 2, reduction to rank space allows us to replace point coordinates by their ranks. Hence, if we use this technique with a data structure that contains m elements, each point can be specified with $O(\log m)$ bits. Thus, we can reduce the space

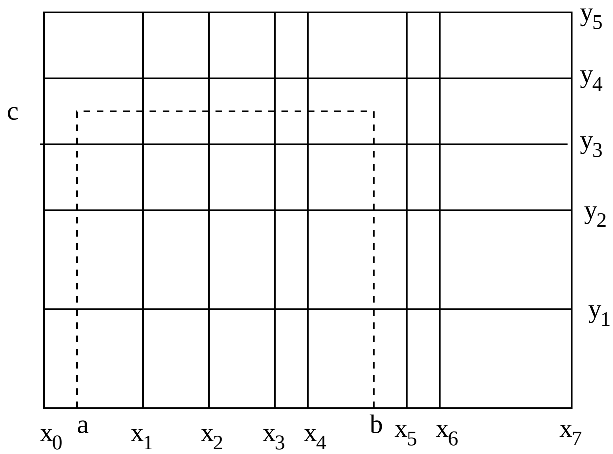

Fig. 1. Example of a $(2, 1, 1)$-sided query projected onto the xy-plane. $i_1 = 1$, $i_2 = 4$, $j = 3$.

usage by replacing point coordinates by their ranks on certain recursion levels.

We apply reduction to rank space on every $\delta \log\log n$-th recursion level for $\delta = \varepsilon/3$. Let V be an arbitrary data structure on recursion level $r = s\delta \log\log n - 1$ for $1 \le s \le (1/\delta)\log_{\frac{2}{1+2\gamma}} 2 \cdot \log\log n$. Let W be the set of points that belong to an x-slice or a y-slice of V. We store a dictionary that enables us to find for each point $p = (p_x, p_y, p_z)$ from W a point $p' = (p'_x, p'_y, p'_z)$ where $p'_x = \text{rank}(p_x, W_x)$, $p'_y = \text{rank}(p_y, W_y)$, $p'_z = \text{rank}(p_z, W_z)$, and W_x,W_y, and W_z are the sets of x-, y-, and z-coordinates of all points in W. Let W' be the set of all points p'. Conversely there is also a dictionary that enables us to find for a point $p' \in W'$ the corresponding $p \in W$. The data structure that answers queries on W stores points in the rank space of W. In general, all data structures on recursion levels $r, r+1, \ldots, r + \delta \log\log n - 1$ obtained by subdivision of W store points in rank space of W. That is, point coordinates in all those data structures are integers bounded by $|W|$. If such a data structure R is used to answer a query Q, then for each point $p_R \in R \cap Q$, we must find the corresponding point $p \in P$. Since range reduction was applied $O(1)$ time, we can find for any $p_R \in R$ the corresponding $p \in P$ in $O(1)$ time.

Each data structure on level $r = s\delta \log\log n$ for $0 \le s \le (1/\delta)v \cdot \log\log n$ and $v = \log_{\frac{2}{1+2\gamma}} 2$ contains $O(n^l)$ elements for $l = (1/2 + \gamma)^r$. Hence an arbitrary element of a data structure on level r can be specified with $l \cdot \log n$ bits. The total number of elements in all data structures on the r-th level is $n2^r$. Hence all elements in all data structures on the r-th recursion level need $O(n2^r((\frac{1+2\gamma}{2})^r)\log n \log\log n)$ bits.

We choose γ so that $1 + 2\gamma \leq 2^{\varepsilon/(3v)}$. Since $r \leq v \log\log n$, $(1 + 2\gamma)^r \leq 2^{(\varepsilon/3)\log\log n} \leq \log^{\varepsilon/3} n$. Therefore all data structures on level r use $\log^{\varepsilon/3} n \cdot O(n \log n \log\log n) = O(n \log^{1+2\varepsilon/3} n)$ bits of space or $O(n \log^{2\varepsilon/3} n)$ words of $\log n$ bits. The number of elements in all data structures on levels $r+1, r+2, \ldots$ increases by a factor two in each level. Hence, the total space needed for all data structures on all levels q, $r \leq q < r + \delta \log\log n$, is $(\sum_{f=1}^{\delta \log\log n - 1} 2^f) O(n \log^{2\varepsilon/3} n) = O(2^{\delta \log\log n} n \log^{\varepsilon} n)$. Since $\delta \leq \varepsilon/3$, $2^{\delta \log\log n} \leq \log^{\varepsilon/3} n$. Thus all data structures in a group of $\delta \log\log n$ consecutive recursion levels use $O(n \log^{\varepsilon} n)$ words of space. Since there are $O(1/\delta) = O(1)$ such groups of levels, the total space usage is $O(n \log^{\varepsilon} n)$.

The data structure on level 0 (the topmost recursion level) can be constructed in $O(n \log^3 n \log\log n)$ time. The total number of elements in all data structures on level s is $2^s n \log\log n$. But each data structure on the r-th recursion level contains at most $n_r = n^l$ elements and can be constructed in $O(l^3 \cdot n_r \log^3 n \log\log n)$ time where $l = (1 + 2\gamma)^r / 2^r$. Hence, all data structure on the r-th recursion level can be constructed in $O((2^r l^3) n \log n \log\log n) = O(((1 + 2\gamma)^{3r}/2^{2r}) n \log^3 n \log\log n)$ time. We can choose γ so that $(1 + 2\gamma)^3 \leq 2$ and $(1 + 2\gamma)^{3r}/2^{2r} \leq 1/2^r$. Then, all data structure on the r-th recursion level can be constructed in $O((1/2^r) n \log^3 n \log\log n)$ time. Summing up by all r, we see that all recursive data structures can be constructed in $O(n \log^3 n \log\log n)$ time.

□

Lemma 7. *There exists a data structure that answers (2,2,1)-sided queries in* $O(\log\log U + (\log\log n)^3 + k)$ *time, uses* $O(n \log^{\varepsilon} n)$ *space, and can be constructed in* $O(n \log^3 n \log\log n)$ *time.*

Proof: The proof technique is the same as in Lemma 6. The grid is divided into x-slices $X_i = [x_{i-1}, x_i] \times n \times n$ and y-slices $Y_j = n \times [y_{j-1}, y_j] \times n$ in the same way as in the proof of Lemma 6. Each x-slice X_i supports $(2, 1, 1)$-sided queries open in $+y$ and $-y$ direction; each y-slice Y_j supports $(2, 1, 1)$-sided queries open in $+x$ and $-x$ direction. All points are also stored in a data structure A_t that contains a point (i, j, z) for each point $(x, y, z) \in P \cap C_{ij}$. For every x-slice and y-slice there is a recursively defined data structure. The reduction to rank space technique is applied on every $\delta \log\log n$-th level in the same way as in the Lemma 6.

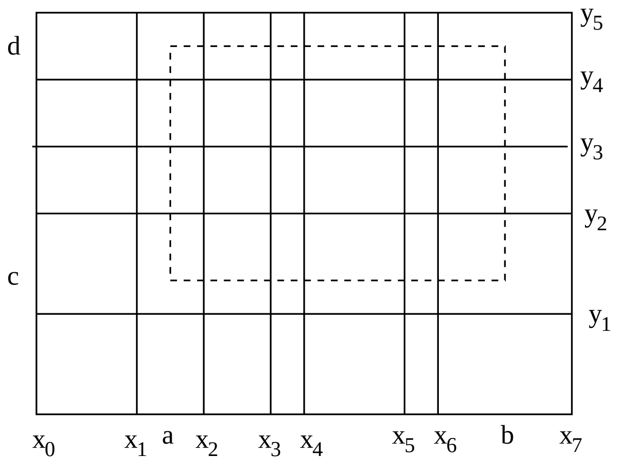

Fig. 2. Example of a (2,2,1)-sided query projected onto the xy-plane. $i_1 = 2$, $i_2 = 7$, $j_1 = 2$, and $j_2 = 5$.

Given a query $Q = [a, b] \times [c, d] \times (-\infty, e]$ we identify indices i_1, i_2, j_1, j_2 such that all cells C_{ij}, $i_1 < i < i_2$ and $j_1 < j < j_2$ are entirely contained in Q. Then Q can be represented as a union of a query $Q_1 = [a_0, b_0] \times [c_0, d_0] \times (-\infty, e]$ and four $(2, 1, 1)$-sided queries $Q_2 = [a, a_0) \times [c, d] \times (-\infty, e]$, $Q_3 = (b_0, b] \times [c, d] \times (-\infty, e]$, $Q_4 = [a_0, b_0] \times [c, c_0) \times (-\infty, e]$, and $Q_5 = [a_0, b_0] \times (d_0, d] \times (-\infty, e]$, where $a_0 = x_{i_1}$, $b_0 = x_{i_2-1}$, $c_0 = y_{j_1}$, and $d_0 = y_{j_2-1}$. The query Q_1 can be answered in constant time, and queries Q_i, $1 < i \leq 5$, can be answered using the corresponding x- and y-slices. Since queries Q_i, $1 < i \leq 5$, are equivalent to (2,1,1)-sided queries each of those queries can be answered in $O((\log \log n)^3 + k)$ time.

If the query Q is entirely contained in one x-slice or one y-slice, then Q is processed by a data structure for the corresponding x-slice resp. y-slice. Since the data structure consists of at most $v \log \log n$ recursion levels, the query can be transferred to a data structure for an x- or y-slice at most $v \log \log n$ times for $v = \log_{\frac{2}{1+2\gamma}} 2$. Hence, the total query time is $O(\log \log n + (\log \log n)^3 + k) = O((\log \log n)^3 + k)$.

The space usage and construction time are estimated in exactly the same way as in Lemma 6 $\square$

Theorem 1. *There exists a data structure that answers three-dimensional orthogonal range reporting queries in $O(\log \log U + (\log \log n)^3 + k)$ time, uses $O(n \log^{1+\varepsilon} n)$ space, and can be constructed in $O(n \log^4 n \log \log n)$ time.*

Proof: This result directly follows from Lemma 7 and Lemma 1. □

Furthermore, we also obtain the result for d-dimensional range reporting, $d \geq 3$.

Corollary 1. *There exists a data structure that answers d-dimensional orthogonal range reporting queries in $O(\log^{d-3} n/(\log\log n)^{d-6} + k)$ time, uses $O(n \log^{d-2+\varepsilon} n)$ space, and can be constructed in $O(n \log^{d+1+\varepsilon} n)$ time.*

Proof: We can obtain a d-dimensional data structure from a $(d-1)$-dimensional data structure using range trees with node degree $\log^{\varepsilon} n$. See e.g. [2], [11] for details. □

Using Theorem 1 we can reduce the space usage and update time of the semi-dynamic data structure for three-dimensional range reporting queries.

Corollary 2. *There exists a data structure that uses $O(n \log^{1+\varepsilon} n)$ space, and supports three-dimensional orthogonal range reporting queries in $O(\log n(\log\log n)^2 + k)$ time and insertions in $O(\log^{5+\varepsilon} n)$ time.*

Proof: We can obtain the semi-dynamic data structure from the static data structure using a variant of the logarithmic method [3]. A detailed description can be found in [11]. The space usage remains the same, the query time increases by a $O(\log n/\log\log n)$ factor, and the amortized insertion time is $O(\frac{c(n)}{n} \log^{1+\varepsilon} n)$, where $c(n)$ is the construction time of the static data structure. □

The result of Corollary 2 can be also extended to $d > 3$ dimensions using range trees.

4 Three-Dimensional Emptiness Queries

We can further reduce the space usage of the three-dimensional data structure if we allow $O(\log\log n)$ penalties for each point in the answer. Such a data structure can also be used to answer emptiness and one-reporting queries. As in the previous section, we design space-efficient data structures for $(2,1,1)$-sided and $(2,2,1)$-sided queries. The proof is quite similar to the data structure of section 3 but some parameters must be chosen in a slightly different way.

Theorem 2. *There exists a data structure that answers three-dimensional orthogonal range reporting queries in $O(\log\log U + (\log\log n)^3 + k \log\log n)$ time, uses $O(n \log n(\log\log n)^3)$ space, and can be constructed in time $O(n \log^4 n \log\log n)$.*

For completeness, we provide the proof of Theorem 2 in the Appendix. Using the standard range trees and reduction to rank space techniques we can obtain a d-dimensional data structure for $d > 3$

Corollary 3. *There exists a data structure that answers d-dimensional orthogonal range reporting queries for* $d > 3$ *in* $O(\log^{d-3} n (\log\log n)^3 + k \log\log n)$ *time, uses* $O(n \log^{d-2} n (\log\log n)^3)$ *space, and can be constructed in* $O(n \log^{d+1} n \log\log n)$ *time.*

References

1. P. Afshani *On Dominance Reporting in 3D*, to be published in ESA 2008.
2. S. Alstrup, G. S. Brodal, T. Rauhe, *New Data Structures for Orthogonal Range Searching*, Proc. FOCS 2000, 198-207.
3. J. L. Bentley, *Decomposable Searching Problems*, Information Processing Letters 8(5), 244-251 (1979).
4. J. L. Bentley, *Multidimensional Divide-and-Conquer*, Commun. ACM 23, 214-229, 1980.
5. P. Bozanis, N. Kitsios, C. Makris and A. Tsakalidis, *New Results on Intersection Query Problems*, The Computer Journal 40(1), 22-29, 1997.
6. B. Chazelle, *A Functional Approach to Data Structures and its Use in Multidimensional Searching*, SIAM J. on Computing 17, 427-462 (1988). See also FOCS'85.
7. B. Chazelle, H. Edelsbrunner, *Linear Space Data Structures for Two Types of Range Search*, Discrete & Computational Geometry 2, 113-126, 1987.
8. B. Chazelle, L.J. Guibas *Fractional Cascading: I. A Data Structuring Technique*, Algorithmica 1 , 133-162 (1986). See also ICALP'85.
9. H. Gabow, J. L. Bentley, R. E. Tarjan, *Scaling and related techniques for geometry problems* Proc. STOC 1984, 135-143.
10. Y. Nekrich, *Space Efficient Dynamic Orthogonal Range Reporting*, Algorithmica 49(2), 94-108 (2007)
11. Y. Nekrich, *A Data Structure for Multi-Dimensional Range Reporting*, Proc. SoCG 2007, 344-353.
12. S. Subramanian, S. Ramaswamy, *The P-range Tree: A New Data Structure for Range Searching in Secondary Memory*, Proc. SODA 1995, 378-387.
13. D. E. Vengroff, J. S. Vitter, *Efficient 3-D Range Searching in External Memory*, Proc. STOC 1996, 192-201.

Appendix. Proof of Theorem 2

Lemma 8. *There exists a data structure that answers (2,1,1)-sided queries in* $O(\log\log U + (\log\log n)^3 + k \log\log n)$ *time, uses* $O(n(\log\log n)^2)$ *space, and can be constructed in* $O(n \log^3 n \log\log n)$ *time.*

Proof: The data structure consists of the same components as the data structure of Lemma 6. But the size of x-slices and y-slices is reduced, so that each x-slice and each y-slice contains $n^{1/2} \log^p n$ points for a constant

$p \geq 2$. The data structure D_t contains a point $(i, j, z_{\min})$ for each cell $C_{ij} = X_i \cap Y_j$, $C_{ij} \cap P \neq \emptyset$, such that $z_{\min}$ is the minimal z-coordinate of a point in $C_{ij} \cap P$. The data structure D_t can contain up to $n/\log^{2p} n$ elements. Combining the results of Lemma 1 and Lemma 6, we can implement A_t in $O((n/\log^{2p} n) \log n \log\log n) = O(n)$ space, so that queries are supported in $O((\log\log n)^2 + k)$ time. A list L_{ij} contains all points in C_{ij} sorted by their z-coordinates. For each x-slice X_i, there are two data structures that support $(1, 1, 1)$-sided queries open in $+x$ and $-x$ direction. For each y-slice Y_j there is a data structure for $(1, 1, 1)$-sided queries open in $+y$ direction. For each x-slice and y-slice, there is a recursively defined data structure. As shown in Proposition 1 of [10], the total number of elements in a data structure on the r-th recursion level can be estimated as $s^r(n) = O(n^{1/2^r} \log^p n \sqrt{\log\log n})$. The recursive sub-division stops when a data structure contains no more than $\log n$ elements. In this case, the data structure is implemented using e.g. the data structure of [2], so that queries are answered in $O(\log\log n)$ time and $O(\log n (\log\log n)^{1+\varepsilon})$ space.

In the same way as in Lemma 6, the query Q can be represented as a union of (at most) one (2,1,1)-sided query on A_t, two (1,1,1)-sided queries on x-slices, and one (2,1,1)-sided query on a recursively defined data structure for a y-slice. Hence, the query time is $O((\log\log n)^3)$ if we ignore the time we need to output points in the answer.

Unlike the data structure of Lemma 6, we apply range reduction on every recursion level. Since the number of elements in a data structure on level r is $= O(n^{1/2^r} \log^p n \sqrt{\log\log n})$, every element in a data structure on level r can be represented with $\log(s^r(n)) = O((1/2^r) \log n + \log\log n)$ bits. The total number of elements in all data structures on level r is $O(n2^r)$. Hence, all level r data structures need $O(n \log n + n2^r \log\log n)$ bits. Summing up by all recursion levels, the total space usage is $O(n \log n \log\log n) + \sum_{r=1}^{r_{\max}-1} n2^r \log\log n$. The maximum recursion level $r_{\max} = \log\log n + c_r$ for a constant c_r. Hence, the second term can be estimated as $\sum_{r=1}^{r_{\max}} n2^r \log\log n = O(n \log n \log\log n)$. If a data structure on the recursion level $r_{\max}$ contains m elements, then it uses $O(m(\log\log n)^{1+\varepsilon})$ words of space because $m \leq \log n$. All data structures on level $r_{\max}$ use $O(n \log n \log\log n)$ words of space. Thus the data structure uses $O(n \log\log n)$ words of $\log n$ bits.

The drawback of applying reduction to rank space on each recursion level is that we must pay a (higher than a constant) penalty for each point in the answer. Consider a data structure D_r on the r-th level of recursion, and let P_r be the set of points stored in D_r. Coordinates of any point stored in D_r belong to the rank space of P_r. To obtain the point $p \in P$

that corresponds to a point $p_r \in P_r$ we need $O(r) = O(\log\log n)$ time. Hence, our data structure answers queries in $O((\log\log n)^3 + k \log\log n)$ time.

The construction time can be estimated with the formula

$$c(n) = O(n \log^3 n \log\log n) + 2(n^{1/2}/\log^p n)c(n^{1/2} \log^p n)$$

Therefore, $c(n) = O(n \log^3 n \log\log n)$. □

Lemma 9. *There exists a data structure that answers (2,2,1)-sided queries in* $O(\log\log U + (\log\log n)^3 + k \log\log n)$ *time, uses* $O(n(\log\log n)^3)$ *space, and can be constructed in* $O(n \log^3 n \log\log n)$ *time.*

Proof: The data structure is the same as in Lemma 8 but in each x-slice there are two data structures for $(2, 1, 1)$-sided queries open in $+x$ and $-x$ directions, and in each y-slice there are two data structures for $(2, 1, 1)$-sided queries open in $+y$ and $-y$ direction.

The query is processed in the same way as in Lemma 7. The space usage can be analyzed in the same way as in Lemma 8. Construction time can be estimated with the formula $c(n) = O(n \log^3 n \log\log n) + 2(n^{1/2}/\log^p n)c(n^{1/2} \log^p n)$ and $c(n) = O(n \log^3 n \log\log n)$. □

Finally, we can apply Lemma 1 and obtain the data structure for three-dimensional orthogonal range reporting queries.