Optimal Deterministic Shallow Cuttings for 3D Dominance Ranges

Peyman Afshani * Konstantinos Tsakalidis †

Abstract

Shallow cuttings are one of the most fundamental tools in range searching as many problems in the field admit efficient static data structures due to their application. We present the first efficient deterministic algorithms that given a set of n three-dimensional points, they construct optimal size (single and multiple) shallow cuttings for orthogonal dominance ranges. In particular, we show how to construct a single shallow cutting in $O(n \log n)$ worst case time, using $O(n)$ space. We also show how to construct in the same complexity, a logarithmic number of shallow cuttings of the input simultaneously. Our algorithms are optimal in the comparison and the algebraic comparison models, and they are an important step forward, since only polynomial guarantees were previously achieved for the deterministic construction of shallow cuttings, even in three dimensions. In fact, our methods yield the first worst case efficient preprocessing algorithms for a series of important orthogonal range searching problems in the pointer machine and the word-RAM models, where such shallow cuttings are utilised to support the queries efficiently.

1 Introduction

We consider the problem of deterministically constructing shallow cuttings for orthogonal dominance ranges in three dimensions. Shallow cuttings were originally conceived for the halfspace range reporting problem [14] and they led to the best known data structures for that problem [8, 5]. In the orthogonal setting, similar (but less-efficient) concepts were used (e.g., [19, 16]) until 2008, when Afshani [1] observed that they could also be used in orthogonal problems and since then they have found numerous applications in orthogonal problems such as orthogonal range reporting [1, 2, 10, 3], top-k reporting [4], adaptive and approximate orthogonal range counting [17, 6, 11]. Despite their power and versatility, they are notoriously difficult to construct in deterministic time. In fact, only deterministic polynomial bounds are known for their construction [7], even in orthogonal problems [1], except for three-sided orthogonal ranges in the plane [4].

Notations and Definitions. Let P be a set of n points in $\mathbb{R}^d$. A point p dominates a point q iff each coordinate of p is greater than or equal to that of q. For a given point $q \in \mathbb{R}^d$ (not necessarily in set P), the *level* of q (with respect to P) is the number of points of P dominated by q. A $(1/k)$-cutting for the $(\leq k)$-level of P, or a *k-shallow cutting* for short, is a set of points S such that every point in S has level $O(k)$, and furthermore any point in $\mathbb{R}^d$ with level at most k is dominated by some point in S. The *size* of a k-shallow cutting is $|S|$, i.e. the number of points in S. The optimal size of a shallow cutting is $O(n/k)$. Every point $p \in \mathbb{R}^d$ uniquely defines the region of space that it dominates (*d-dimensional dominance range*). We call this region the corresponding *cell* of the *apex* point p, and we use p to denote both of them, as well as the d-dimensional orthogonal surface that bounds the cell. We also define the *conflict list* of a cell $p \in S$ to be the subset of P that lies inside p. Finally, we say that a surface S *lies between level a and b*, if every point on S has level between a and b.

*MADALGO (Center for Massive Data Algorithmics, a Center of the Danish National Research Foundation), Department of Computer Science, Aarhus University, Denmark. peyman@madalgo.au.dk

†Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. tsakalid@cse.ust.hk

Query Range	Query Time	Space	Preprocessing Time
3D dominance [1]	$\log n + k$	n	$n \log n$ †
concurrent 3D dominance [2]	$C \log\log n + \log n + k$	n	$n \log n$ †
3D rectangle [2]	$\log n + k$	$n \left(\frac{\log n}{\log\log n}\right)^2$	$n \log n$ †
dD hyper-rectangle [2]	$\log n \left(\frac{\log n}{\log\log n}\right)^{d-3} + k$	$n \left(\frac{\log n}{\log\log n}\right)^{d-1}$	$n \log n \left(\frac{\log n}{\log\log n}\right)^{d-3}$
3D dominance [1, 9]	$\log\log U + k$	n	$n \log n$

Table 1: Asymptotic bounds for variants of the orthogonal range reporting problem with input size n, query output size k and C different colors, in the pointer machine and word-RAM ($U \times U \times U$ grid) models, stated respectively above and below the horizontal line. All preprocessing times are new, and optimal when marked with †.

Related Work. As discussed, shallow cuttings were first conceived for the halfspace ranges [14]. However, soon it became clear that even bivariate algebraic functions of constant degree admit shallow cutting, although their sizes depend on combinatorial properties of the functions [7]. For dominance ranges, it is known that they admit k-shallow cuttings of size $O(n/k)$ [1]. One can of course use the techniques developed for the algebraic functions to construct shallow cuttings for the dominance ranges, but that would take polynomial deterministic time. $O(n \log n)$ expected time can be achieved for such 3D dominance ranges, by adapting the approach of Ramos [18] for constructing shallow cuttings for 3D halfspace ranges. In fact, the latter is the only 3D problem for which $O(n \log n)$ preprocessing time is achieved [18, 8, 5], however by using randomization. Deterministic construction algorithms take polynomial time in the worst case [15, Theorem 2.1]. Because of lack of efficient methods for their construction, some researchers have turned to weaker alternatives to shallow cuttings. The deterministic algorithm of Nekrich [16] constructs k-shallow cuttings of suboptimal size $O(n)$ (a.k.a., *k-approximate boundaries* [19]) in $O(n \log^3 n)$ time, using $O(n)$ space. A construction similar to shallow cuttings, but with weaker properties [10], can be obtained from a random sample of the input pointset in $O(n \log n)$ time, using $O(n)$ space. All construction algorithms are implementable in the pointer machine.

Shallow cuttings of optimal size $O(n/k)$ are especially useful in building space-efficient data structures for supporting various kinds of orthogonal range searching queries [1, 17, 2, 6][1]. Range reporting queries [1, 2] require a single shallow cutting of the input pointset. In particular, Afshani [1] combines a single shallow cutting with a point location data structure to support *3D dominance reporting queries*, i.e. report the points that dominate a given query point, in optimal $O(\log n + k)$ time, where k is the output size, using linear space in the pointer machine. In the word-RAM model and in combination with the latest results on the point location problem [9], the query time is improved to $O(\log\log U + k)$, when the points lie on a $U \times U \times U$ grid [1]. These data structures are used as building blocks to answer the more general *3D orthogonal range reporting queries*, i.e. report the input points that are contained in a given 3D axis-aligned query rectangle: In the word-RAM model, Chan, Larsen, and Pătraşcu [10] have obtained the best data structure that uses $O(n \log^{1+\varepsilon} n)$ space (in which $\varepsilon > 0$ is an arbitrary constant) and can answer queries in $O(\log\log U + k)$ time. In the pointer machine model, Afshani, Arge and Larsen [2] have obtained an optimal data structure that uses $O(n(\frac{\log n}{\log\log n})^2)$ space and can answer queries in $O(\log n + k)$ time. The word-RAM result extends effortlessly to higher dimensions, where each extra dimension suffers a penalty of $O(\log^{1+\varepsilon} n)$ in space and of $O(\log n / \log\log n)$ in query time. To extend the pointer machine result to higher dimensions, Afshani et al. [2] consider a generalized form of the problem (called *concurrent 3D dominance reporting queries*), which they solve using a variety of techniques that include 3D shallow cuttings. In doing so, the penalty suffered for each extra dimension is $O(\log n / \log\log n)$ in both space and query time, which gives the currently best result for orthogonal range reporting in the pointer machine.

On the other hand, approximate range counting queries [17, 6] require multiple shallow cuttings of the input pointset. In particular, Afshani, Hamilton and Zeh [6] show how to support *approximate 3D dominance*

[1]This is not true for 4D dominance shallow cuttings as their size is $O(n^2/k)$.

counting queries, i.e. given a query point q compute a number k' such that $k \leq k' \leq (1+\epsilon)k$, where k is the actual number of points that dominate q. They achieve $O(\log \frac{n}{k})$ query time using linear space in the pointer machine. In the word-RAM model and with input in rankspace, Nekrich [17] shows how to support such queries with additive error k^ρ in $O((\log\log n)^3)$ time and linear space, when the error parameter $\rho \in (0,1)$ is given in advance. When the parameter is given at query time, $O((\log\log n)^3 + 3^{\log \frac{1}{\rho}} \log\log n)$ time and $O(n \log^4 n)$ space is achieved [17]. Using standard techniques, the corresponding variants of the more general *approximate 3D range counting queries*, where the query range is a 3D orthogonal rectangle, are supported in the same query time and $O(n \log^4 n)$ space and $O(n \log^7 n)$ space, respectively for both cases [17]. Finally, note that the above papers do *not* discuss the worst case preprocessing time, since so far the construction of shallow cuttings is efficient only in expectation.

Our Contributions. In Section 3 we present a deterministic algorithm for the pointer machine model that given a set of n points in $\mathbb{R}^3$ and a non-negative integer parameter k, it constructs a k-shallow cutting for dominance ranges of optimal size $O(n/k)$. The algorithm takes $O(n \log n)$ time and uses $O(n)$ space. This improves significantly the deterministic construction time of 3D shallow cuttings from polynomial to $O(n \log n)$, as well as for the k-approximate boundaries [16]. The latter is also improved in terms of the output's space-efficiency. Previously, space-efficient constructions could only be obtained in the same expected complexity, and moreover they had weaker properties that demanded more complicated data structures to account for them [10].

It should be obvious from the previous discussion that by constructing shallow cuttings deterministically, we can obtain deterministic preprocessing times for a number of important problems. Thus, we obtain optimal $O(n \log n)$ worst case preprocessing time for supporting (simple and concurrent) 3D dominance reporting queries [1, 2] in the pointer machine. Thus, for simple 3D dominance reporting, we match all complexities of the structure of Makris and Tsakalidis [13] that avoids shallow cuttings. Moreover, we obtain the currently most efficient worst case preprocessing time for this problem in the word-RAM [1]. The improvement in concurrent 3D dominance reporting allows for preprocessing the pointer machine structures for 3D and d-dimensional orthogonal range reporting in optimal and $O(n \log n (\frac{\log n}{\log\log n})^{d-3})$ worst case time, respectively. Finally, we couple the structures of [10] for 3D and d-dimensional range reporting in the word-RAM with deterministic preprocessing algorithms. Refer to Table 1 for the list of improvements in orthogonal range reporting.

Our Techniques. Our result is obtained by a combination of the plane-sweep technique with the incremental maintenance of shallow cuttings for planar dominance ranges. Similar approaches for constructing approximate boundaries [19, 16] have failed to guarantee optimal $O(n/k)$ output size. To tackle this challenge, we use a notion of *shallow vertices* and upper-bound their total number by a novel amortization argument that uses random sampling and exploits the geometry of the plane. This also guarantees that our simple algorithm runs in optimal time and moreover that it is deterministic, since randomization is only used in the analysis.

Multiple Shallow Cuttings. In Section 4 we show how to construct a logarithmic number of k_i-shallow cuttings of the input pointset *simultaneously* in $O(n \log n)$ time, using $O(n)$ space, for geometrically increasing parameters k_i. Approximate range counting queries require the computation of such shallow cuttings and thus, by using our algorithm, we obtain the first worst case efficient algorithms for preprocessing the structures for approximate 3D dominance and range counting [6, 17]. The result is obtained by non-trivial modifications to the algorithm of Section 3 that mainly avoid the space blow-up caused by merely running multiple instances independently.

2 Preliminaries

A k-shallow cutting for dominance ranges on a planar pointset is an orthogonal surface that has the shape of a *staircase* orthogonal curve $c_1 d_1 c_2 d_2 \ldots d_{t-1} c_t$ defined by alternating *outer corner* points c_i and *inner corner*

points d_i, such that $c_i(x) = d_i(x)$ and $d_i(y) = c_{i+1}(y)$. In other words, it is a set of $t-1$ vertical line segments $c_i d_i = [c_i(x)] \times [c_i(y), d_i(y)]$ alternating with a set of t horizontal line segment $d_i c_{i+1} = [d_i(x), c_{i+1}(x)] \times [d_i(y)]$.

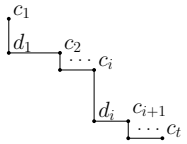

Figure 1: 2D shallow cutting (staircase) of an (omitted) pointset. Outer corners c_i dominate $10k$ points and inner corners d_i dominate $9k$ points.

Lemma 1. *For a set of n presorted planar points and for any integer k, a k-shallow cutting of size $O(n/k)$ can be computed in $O(n)$ time and space. The staircase lies between the $9k$- and $10k$-level.*

Proof. Let $x_1, \ldots, x_n$ be the x-coordinates of the points in sorted order. The shallow cutting is obtained from a staircase curve that begins from the point $c_1 = (x_{10k}, +\infty)$ and follows T vertical and horizontal line segments alternatively. A vertical line segment starts from a corner c_i that dominates $10k$ points and it ends at an endpoint d_i that dominates only $9k$ points. The segment $c_i d_i$ can be found in $O(k)$ time by scanning all the points dominated by c_i. A horizontal line segment starts from a corner d_i that dominates $9k$ points and ends at an endpoint c_{i+1} that dominates $10k$ points. If there are n_i points above the line segment $d_i c_{i+1}$, and since the input points are sorted by x-coordinate, we can find c_{i+1} in $O(n_i + k)$ time (see Figure 1). Since there are k points to the left of (resp. below) every constructed vertical (resp. horizontal) segment (except maybe for the last vertical segment where it is at most k), we construct in total $T = O(n/k)$ segments. Thus computing all vertical segments takes $O(Tk) = O(n)$ time, and all horizontal segments $O(\Sigma_{i=1}^{T} n_i + Tk) = O(n)$ time. □ □

3 Shallow Cuttings for 3D Dominance Ranges

In this section we prove the following theorem for the pointer machine model:

Theorem 1. *Given a set P of n points in $\mathbb{R}^3$ and any integer k, a k-shallow cutting S for 3-dimensional dominance ranges can be constructed deterministically in optimal $O(n \log n)$ worst case time and $O(n)$ space, such that $|S| = O(n/k)$.*

The algorithm is based on the plane sweep approach. In particular, after presorting the points, we sweep a plane parallel to the xy-plane starting from $z = +\infty$ and progressing towards $z = -\infty$. During the sweep we maintain a planar shallow cutting of the xy-projection of the points of P that lie *below* the sweep plane. That is, initially we store the xy-projections of all points of P, and the sweep is simulated by deleting the projection of the point in P with the next largest z-coordinate. As the sweep progresses, the planar shallow cutting is modified for the current z-coordinate by replacing consecutive parts of it with new staircase "patches" that dominate the removed parts (in the 2D sense). This update operation involves removing existing planar cells and creating new ones instead. In particular, each planar cell with apex point (x, y) that is created due to the deletion of a point with z-coordinate z, corresponds to the 3D cell with apex point (x, y, z) in the output 3D shallow cutting S. To show correctness, notice that at any point of time z, the existing 3D cells occur as 2D cells on the currently maintained planar shallow cutting. A 3D point $p := (x', y', z')$ that dominates k input points, must dominate these points also in the plane at the time when the sweep plane was at z'. Thus there exists a corner c in the planar shallow cutting that dominates the projection of p. Since the apex points of the current 3D cells have z-coordinate larger than z', the corresponding 3D apex point for c dominates p also in the 3D sense.

Thus, to obtain a 3D k-shallow cutting within the desired bounds, it suffices to solve the following 2D semi-dynamic problem:

Problem 3.1. *Given a set P of n points in the plane, maintain a planar k-shallow cutting under deletions of points from P, such that:*
(i) the amortized cost of each deletion is $O(\log n)$ and
(ii) the total number of created shallow cutting cells is $O(n/k)$.

3.1 Algorithm for Problem 3.1

We maintain a 2D shallow cutting in the form of a staircase $c_1 d_1 c_2 d_2 \dots d_{t-1} c_t$ under the invariant that the level of each inner corner is at least k and that the level of each outer corner is at most $10k$. To complete our algorithm, we need to show how a point is deleted and then argue that the total number of cells created is $O(n/k)$. In our algorithm, we will use the following auxiliary structure.

Lemma 2. *Given a set P of n points in the plane, there exists a data structure that supports the following queries: given a y-coordinate value y (resp. x-coordinate value x) and an integer k, return a value x (resp. y) such that the point (x, y) dominates k points of P if such an x (resp. y) exists, and report these dominated points. The data structure supports deletions of points from P in $O(\log n)$ time, answers a query in $O(\log n + k)$ time, takes $O(n \log n)$ preprocessing time, and uses $O(n)$ space.*

Proof. The data structure can be easily obtained by combining the priority search tree with a heap selection algorithm that can select k points from a heap. Refer to [4] for the details. □ □

Data Structures. We maintain two symmetric copies of the data structure of Lemma 2 over the input pointset. We also maintain a planar shallow cutting in the form of a staircase $S = c_1, d_1, \dots, c_t$ stored in a doubly linked list. For every point $p \in P$ that is dominated by some corners of S, we store a pointer to some (inner or outer) corner of S that dominates p. Since the corners of S that dominate p occur consecutively in S, we can traverse all of them efficiently by use of this pointer. To facilitate updating, we also store a corresponding backward pointer from the corner to the point. Finally for each corner, we store a counter equal to the number of input points it dominates (level). After computing S, the initial pointers and the counters can be computed by scanning simultaneously the sets P and S by increasing x-coordinate.

Deletion Algorithm. Let p be the point to be deleted next. First, we delete p from the auxiliary data structures. Next, we find the corners of S that dominate p by traversing its pointer. If the pointer is not defined and thus no such corner exists, we are done. Otherwise, let m corners dominate p. We access them in $O(m)$ time and we decrement their counters by one. This may cause the invariant to be violated for some inner corners that occur consecutively in S.

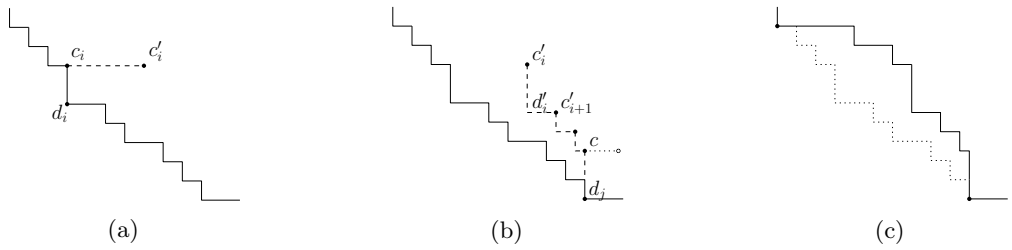

Figure 2: "Patching" the staircase to cover the violating inner corner d_i: (a) We find the initial corner c'_i. (b) The new staircase is extended (in both directions) until it covers an inner corner d_j of the old staircase with level at least $7k$. (c) The dominated part of the old staircase is deleted.

It remains to show how the invariant can be fixed. The idea is to *patch* the staircase S by a new staircase S' that covers at least one inner corner of S which violates the invariant. S' is such that its inner corners satisfy the invariant and that its points 2D-dominate a consecutive part of S. We replace the dominated part of S with S', and we continue to the next (to the right) inner corner of S that violates the invariant and has not

been replaced yet. It is clear that the algorithm may create several patches that are nevertheless disjoint with each other.

Now we present the details of how every patch is processed. Let d_i be the leftmost inner corner of S whose invariant is violated, and let c_i be the preceding outer corner in S. First, we find an **initial** outer corner of S' by querying the auxiliary data structure with y-coordinate $c_i(y)$ and value $10k$. This returns a new outer corner c'_i, such that $c_i(x) < c'_i(x)$ and $c_i(y) = c'_i(y)$. See Figure 2a. Then, we find the corners **succeeding** c'_i in S'. In particular, to determine the next inner corner $d'_i = (c'_i(x), y')$, we scan the points dominated by c'_i (reported by the auxiliary structure) to find the point with the $9k$-th largest y-coordinate y'. We add the vertical segment $c'_i d'_i = [c'_i(x)] \times [c'_i(y), d'_i(y)]$ to S', and find the next outer corner c'_{i+1} of S' by querying the auxiliary structure as above, but with the y-coordinate $d'_i(y)$ instead. Let d_j be the leftmost inner corner of S dominated by c'_{i+1}. If d_j has level less than $7k$, we add the horizontal segment $d'_i c'_{i+1} = [d'_i(x), c'_{i+1}(x)] \times [d'_i(y)]$ to S', and continue for the next inner corner as above. Otherwise, we **terminate** the patching by adding the horizontal segment $d'_i c = [d'_i(x), d_j(x)] \times [d'_i(y)]$ and the vertical segment $c d_j = [d_j(x)] \times [d'_i(y), d_j(y)]$ to S'. In this case we call d_j a *joined corner* and we consider it newly created. See Figure 2b. Moreover, we repeat the above procedure symmetrically to find the corners of S' that **precede** c'_i. Finally, we **incorporate** S' to S. That is, for every horizontal (resp. vertical) segment of S', we remove the old corners of S dominated by the upper (resp. right) endpoint of the segment, by updating the linked list appropriately. For every removed old corner of S, we reassign the pointer of the input points it dominates to this endpoint, by use of the corresponding backward pointer. See Figure 2c.

So in summary, when the level of an inner corner has fallen below k, we patch the staircase by using the auxiliary data structures. This process creates two joined corners, removes a consecutive part of the old staircase and adds a number of new staircase corners. The correctness is obvious, so it remains to analyse the complexity.

3.2 Analysis

We proceed with the time and space analysis of the algorithm. Presorting the points and building the data structures of Lemma 2 takes $O(n \log n)$ time. Constructing the staircase takes $O(n)$ time by Lemma 1, and computing the initial pointers and counters takes $O(n)$ time. Thus, the total preprocessing time is $O(n \log n)$. The total space is $O(n)$, since the auxiliary structure takes $O(n)$ space (Lemma 2), we store $O(n)$ pointers, and the staircase with the counters takes $O(n/k)$ space.

Total Running Time. Throughout our algorithm we will have n deletions. Let T be the total number of cells created by the algorithm. By Lemma 2, the deletions take $O(n \log n)$ time. For each deletion of a point, we access and decrement the counters of the staircase corners that contain it in their conflict lists; this takes time proportional to the number of staircase corners that dominate the deleted point. However, since each staircase corner dominates $O(k)$ points, this will take $O(Tk)$ time in total. We also reassign the pointers to the corners of the part removed from the staircase. It is clear that this takes $O(k)$ time for each removed corner. Since a corner is created and removed at most once, updating all pointers will take $O(Tk)$ time in total. Finally, we need to analyse the cost of creating new cells (i.e., corners). Each new cell is created by querying the auxiliary data structure that has $O(\log n + k)$ query time. This results in $O(T \log n + Tk)$ total time for creating all the T cells. Adding these up, we obtain $O(n \log n + T \log n + Tk)$ total running time. Thus, if we can show that $T = O(n/k)$, then the total running time is $O(n \log n)$ and then we are done.

3.2.1 Bounding the Size

Here we prove the most involved part of the analysis, namely that our algorithm creates at most $O(n/k)$ cells. The overall approach is a charging argument. However, further definitions are needed before we can present our charging scheme.

For a point p in the plane, we say that a point q is *to the southeast of* p, if q has larger x-coordinate and smaller y-coordinate than p. We say that the points $p_1, \ldots, p_t$ are the *first t points to the southeast of* p, if they lie to the southeast of p and they have the t smallest x-coordinates among all the points to the southeast of p.

For any two input points $p = (p_x, p_y)$ and $q = (q_x, q_y)$, we call the point $u := (\max\{p_x, q_x\}, \max\{p_y, q_y\})$ a *vertex of the arrangement* (or a vertex for short). Furthermore, if u dominates $O(k)$ input points, we call it a *shallow vertex*. Note that as we delete the points from P, some new vertices of the arrangement may become shallow, while other vertices might simply disappear; which happens when one of their two defining points gets deleted. We will give an amortization argument that uses shallow vertices to pay for either the removal or the creation of the inner corners. In particular, we will charge the removal of an inner corner to enough and different shallow vertices. Although we will allow some of these shallow vertices not to exist at the time of the removal, we will show that each shallow vertex can only be charged once. Nevertheless, charging shallow vertices is not sufficient to account for the removal of joined inner corners, for which we will use another charging scheme.

To complete this argument, first we need a bound on the total number of shallow vertices created throughout the sequence of deletions. The bound is provided by Lemma 4 that relies on Lemma 3, which in turn demands the following definition: An input point is *minimal* when it does not dominate any other input point. The diagram of minimal points has also a staircase shape with inner and outer corners, where the input points lie on the inner corners and each outer corner is adjacent to two inner corners.

Lemma 3. *Any sequence of deletions from a set P of n planar points creates $O(n)$ minimal vertices.*

Proof. A static pointset P contains n minimal points in the worst case. This is true also under the deletion sequence, since P contains n points altogether and any point in P may become minimal at most once before it gets deleted. Thus, the sequence of deletions creates at most n inner corners. Since each inner corner is adjacent to two outer corners, it follows that the number of outer corners created by the deletion sequence is also $O(n)$. □ □

Lemma 4. *Any sequence of n deletions from a set of n planar points creates $O(nk)$ shallow vertices.*

Proof. We prove the lemma using the random sampling technique of Clarkson and Shor [12]. Consider one particular deletion sequence, $S := p_1, \ldots, p_n$. Fix a subsequence $S_r = r_1, \ldots, r_t$ of S by including each p_i independently and with probability $1/k$. Assume deleting $p_1, \ldots, p_i$ leads to the creation of a shallow vertex u defined by points p_j and p_s, $j, s > i$. Since we have sampled each element of S independently in S_r, with probability $1/k^2$ both p_j and p_s will be in S_r and with probability $(1 - 1/k)^{O(k)} = \Omega(1)$ none of the points dominated by u will be sampled in S_r. This event E_u occurs with probability $\Omega(1/k^2)$.

Now consider the pointset $P_r = \{r_1, \ldots, r_t\}$ under the deletion sequence S_r. Event E_u occurs when u appears as a *minimal* vertex of P_r under the deletion sequence S_r. In other words, if we delete the points of P_r one by one according to sequence S_r, at some point we will have the situation that both p_j and p_s (the defining points of u) are not deleted but all the points of P_r that are dominated by u have been deleted from P_r. Thus, if there are M shallow vertices in total, by linearity of expectation, we expect to see $\Omega(M/k^2)$ minimal vertices in the pointset P_r under the deletion sequence S_r. However, the expected size of P_r is n/k and by Lemma 3 any sequence of deletions can only create $O(|P_r|)$ minimal vertices. This implies that the expected number of minimal vertices m created under the sequence S_r is $O(n/k)$. Thus, since $m = \Omega(M/k^2)$ and $m = O(n/k)$, we have that $M = O(nk)$. □ □

Remember that to fix the invariant, we patch the violating part of the staircase with a new staircase. The operation removes a section of the old staircase. As discussed, we need to either pay for the removal or the creation of inner corners. The payment scheme is as follows:

(1) Charging $k^2/4$ shallow vertices can pay for the removal or the creation of $O(1)$ inner corners.

(2) Removing 5 joined corners can pay for the creation of 2 (joined or non-joined) inner corners.

The intuition behind the above scheme is the following: First, we show that we can use (1) to pay for the removal of most inner corners. By Lemma 4 we have $O(nk)$ shallow vertices and so, if we can show that each shallow vertex is spent only once, then it follows that (1) can only be used $O(n/k)$ times. Unfortunately, this scheme doesn't work for joined corners. Nevertheless, we can show that each time we patch a staircase, at

least five (joined or non-joined) inner corners are removed. If (1) was used to pay for their removal, then it can also pay for the creation of the two new joined corners. If not, then at least five joined corners were removed by patching, and we can use (2) to pay for the creation of the two new joined corners. Since patching creates only two joined corners, this charging scheme results in a cascading effect that implies that the number of times (2) is used cannot be asymptotically larger than the number of times (1) is used. Now, we present the details.

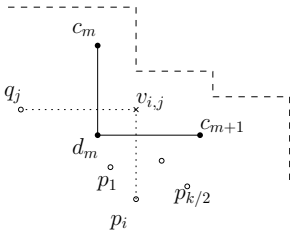

Figure 3: The non-joined corner d_m of the old staircase (solid line) is covered by the new staircase (dashed line) and thus dominates less than $7k$ points. Points $p_1, \ldots, p_{k/2}$ are the first $k/2$ points to the southeast of d_m and they lie below the horizontal segment $d_m c_{m+1}$. Thus, the vertex v_{ij} is shallow and lies below the new staircase.

Lemma 5. *Each non-joined inner corner d_m can be associated with $k^2/4$ shallow vertices at the time of its removal, such that no shallow vertex is associated with two different corners. The associated shallow vertices may not exist at the removal time, but they must exist at the creation time of d_m.*

Proof. We consider the status of the pointset at two crucial times. First, the time $\mathtt{T}$ when d_m is created and, second, the time when d_m is deleted; we refer to the latter as present time. Let $p_1, \ldots, p_{k/2}$ be the first $k/2$ points to the southeast of d_m at time $\mathtt{T}$. See Figure 3. Since d_m is not a joined corner, the x-coordinate of all these points must be smaller than the x-coordinate of the succeeding outer corner c_{m+1}. For every $p_i = (x_i, y_i), i \in [1, k/2]$, consider the points to the northwest of d_m that existed at time $\mathtt{T}$, and among them consider the point q_j with the j-th smallest y-coordinate. Let $v_{i,j}$ be the vertex determined by p_i and q_j. At time $\mathtt{T}$, $v_{i,j}$ was dominating exactly $i + j \leq k$ more input points than d_m, meaning that its level was at most $10k$. So at time $\mathtt{T}$, $v_{i,j}$ existed as a shallow vertex. At present, the level of d_m is less than $7k$, and thus the level of $v_{i,j}$ is less than $8k$, which means that $v_{i,j}$ lies below the new staircase. We can now associate the removal of d_m to any such vertex $v_{i,j}$ and it is clear that there are at least $k^2/4$ of them. The fact that the x-coordinates of all these $\Omega(k^2)$ vertices are smaller than $c_{m+1}(x)$ implies that we will not associate the same vertex to different inner corners of the old staircase. The fact that after the removal from the old staircase, these vertices lie below the new staircase implies that they will not be associated again with any other corner created in the future. □ □

By Lemma 5, we can use (1) to pay for the removal of every non-joined corner. Since the number of shallow vertices is $O(nk)$, we cannot use (1) more than $O(n/k)$ times, meaning that the number of created non-joined corners is $O(n/k)$. It remains to account for the joined corners with the aid of two following lemmata.

Lemma 6. *The levels of two adjacent corners of the staircase differ by at most $3k$.*

Proof. If the inner corner is not a joined corner, then it is clear that the difference in level from the preceding and succeeding outer corners is at most k. This holds similarly for the outer corners. I.e. two adjacent corners are created with exactly k difference in their levels and this difference can only decrease. Otherwise when a joined inner corner is created, it has level at least $7k$, while the preceding and succeeding outer corners have level at most or exactly $10k$. □ □

Lemma 7. *After patching, at least five inner corners are dominated by the new staircase.*

Proof. Let d_i be the inner corner that triggers the patching. The level of d_i is less than k and, by Lemma 6, its adjacent outer corners c_i and c_{i+1} have level less than $4k$, and thus the inner corners d_{i-1} and d_{i+1} also have level less than $4k$. Similarly, the inner corners d_{i-2} and d_{i+2} have level less than $7k$. Hence, all these inner corners are dominated by the new staircase and deleted from the old staircase. □ □

Consider the part of the old staircase that is patched by a new staircase. By Lemma 7, it contains at least five inner corners. If at least one of these is not a joined corner, we can use (1) with the $k^2/4$ shallow vertices associated with it to also pay for the two joined corners created by the patching. This leaves the case when the part contains at least five joined corners. In this case, we will use (2) to pay for the creation of the two joined corners. Assume that (1) has been used $X = O(n/k)$ times, and thus that $O(X)$ joined inner corners have been paid by it. Since each time (2) is used we have a net loss of three joined inner corners, (2) cannot be used more than $O(X) = O(n/k)$ times. This concludes the analysis of our algorithm and proves Theorem 1.

4 Constructing Multiple Shallow Cuttings

In this section we show how to construct multiple shallow cuttings of the same input pointset simultaneously. In particular, we modify the algorithm of Section 3 to prove the following theorem for the pointer machine:

Theorem 2. *Given a set P of n points in $\mathbb{R}^3$, all k_i-shallow cuttings, $k_i = 10^i$ for $i \in [1, \log_{10} n]$, for 3-dimensional dominance ranges can be constructed deterministically and simultaneously in optimal $O(n \log n)$ worst case time and $O(n)$ space, such that $|S_i| = O(n/k_i)$.*

We follow the same sweep-plane approach, starting with the whole pointset and progressing the sweep in decreasing z-coordinate. The main difference here is that we maintain multiple planar k_i-shallow cuttings of the current pointset's xy-projection, which is equivalent to solving simultaneously $O(\log n)$ instances of Problem 3.1. Thus we demand every staircase to satisfy independently the invariant of Subsection 3.1.

Data Structures. We observe that the auxiliary structures of Lemma 2 do not depend on the parameter k so we can use the same structures for all the different values of k_i. Moreover, we maintain $O(\log n)$ staircases S_i as described in Subsection 3.1. However to avoid blowing up the space to $O(n \log n)$, we redefine the pointers stored at the input points. In particular, we store a pointer from every corner c of staircase S_i to some corner of S_{i+1} that dominates c. The corners that do not contain such pointers belong to the "highest" staircase S_t, i.e. the one with largest index. To facilitate updating, every such pointer is coupled by a corresponding backward pointer. Finally, for every input point p dominated by corners of several staircases, we only store one pointer to some such dominating corner that belongs to the staircase with the smallest index, i.e. the "lowest" staircase "above" p, i.e. the staircase with smallest index among the staircases that intersect the vertical ray starting from p and extending to $y = +\infty$. See Figure 4. In this way, the total space consumption remains $O(n)$.

Deletion Algorithm. Let p be the point to be deleted next. We delete p from the auxiliary structures, and follow its pointer to a corner of the lowest staircase S_i above p. If the pointer is not defined, p lies above the highest staircase S_t and thus we are done. Otherwise, we traverse the pointer and decrement by one the counters of the corners of S_i that dominate p. When we are done, we repeat this process for the next staircase S_{i+1}, by traversing the pointer of any already processed corner of S_i. The iteration terminates after processing S_t. Observe that we are crucially using the properties of the shallow cuttings: any corner of a k_i-shallow cutting is dominated by at least one corner of k_{i+1}-shallow cutting since $k_{i+1} = 10k_i$. Nonetheless, the removal of p may cause the invariant to be violated for several inner corners of the staircases $S_j, j \in [i, t]$ above p. We consider these staircases in increasing index j and restore the invariant as described in Section 3, with the difference that we update the pointers as following. When patching S_j, we update the pointers of

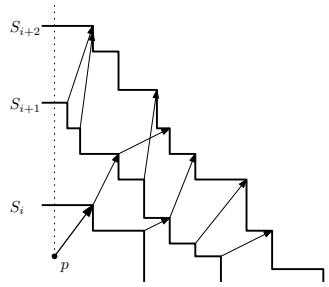

Figure 4: Multiple shallow cuttings. Pointers between the outer corners are shown. An input point p points to the lowest staircase above it.

the removed corners as described for pointers of input points in Section 3. Also, every input point dominated by a new corner of the patch either lies below the removed part of S_j or above it. In the latter case it lies below $S_j + 1$, since the staircases are disjoint (again this follows from our invariant and that $k_{i+1} = 10k_i$), and we set it to reference the new corner. We also do that in the former case, if the point actually refers to a removed corner of S_j. Otherwise the point lies below any staircase $S_{j'}$ with $j' < j$, and thus its pointer has been already updated.

Time Analysis. The n deletions from the auxiliary structures cost $O(n \log n)$ time. Without loss of generality, we assume that all possible $t = \log_{10} n$ values k_i have been set. Let T_i be the number of new cells for staircase S_i. Since we process every staircase independently, Lemma 4 implies that $T_i = O(n/k_i)$. It takes $O(T_i(\log n + k_i))$ to query the auxiliary structures and to update the pointers of the input points. Any staircase S_i contains $O(n/k_i)$ old or new corners in total, and thus we need as much time to update their pointers. Thus in total we need $O(n \log n + \sum_{i\in[1,t]} T_i(\log n + k_i) + \sum_{i\in[1,t]} n/k_i) = O(n \log n)$ total time, since $\sum_{i\in[1,t]} k_i = O(n)$ and $\sum_{i\in[1,t]} n/k_i = O(n)$.

5 Applications

In this section we present the applications of the algorithms described in Sections 3 and 4.

Orthogonal Range Reporting. The internal memory data structures for single and concurrent dominance reporting [1, 2] use a single shallow cutting. Moreover, the bottleneck in their preprocessing time stems from the computation of the cutting, as all other necessary machinery can be preprocessed in $O(n \log n)$ deterministic time. Thus, by employing the algorithm of Theorem 1 to construct the cutting, we need optimal $O(n \log n)$ total time and linear space to preprocess the structures for 3D dominance reporting in the pointer machine and the word-RAM models [1]. We also obtain the same optimal preprocessing complexity for concurrent 3D dominance reporting [2] and thus also for the currently most efficient 3D range reporting data structure in the pointer machine [2]. By standard techiques, the latter implies $O(n \frac{\log^{d-2} n}{(\log \log n)^{d-3}})$ preprocessing time and $O(n(\frac{\log n}{\log \log n})^{d-1})$ space for the currently most efficient d-dimensional range reporting structure in the pointer machine [2]. Finally, the word-RAM structures for 3D and d-dimensional range reporting [10] can be preprocessed deterministically.

Orthogonal Approximate Range Counting. Preprocessing the data structures for approximate dominance counting [17, 6] is also expensive only due to the construction of the cuttings. In this case though it is required to construct multiple cuttings such that they abide by Theorem 2. Since the algorithm of Section 4 constructs the cuttings simultaneously and since it suffices to store only the cells (apices) and not their

conflict list, we obtain the first $O(n \log n)$ deterministic time and linear space algorithm for preprocessing the structures for approximate 3D dominance counting in the pointer machine [6], and in the word-RAM [17] for the case where the error is given in advance. Moreover, in the word-RAM we obtain the first deterministic algorithms for preprocessing the structures for approximate 3D dominance and range counting [17].

6 Conclusion

We have presented optimal and yet simple algorithms for deterministically constructing single and multiple shallow cuttings for three-dimensional orthogonal dominance ranges. It would be interesting to see if our techniques are applicable to other variants of range spaces. Unfortunately, our algorithms are not I/O-efficient due to the lack of efficient external counterparts of the used data structures.

References

[1] Peyman Afshani. On dominance reporting in 3D. In *Proc. 16th European Symposium on Algorithms*, pages 41–51, 2008.

[2] Peyman Afshani, Lars Arge, and Kasper D. Larsen. Orthogonal range reporting: Query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proc. 26th ACM Symposium on Computational Geometry*, pages 240–246, 2010.

[3] Peyman Afshani, Lars Arge, and Kasper G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proc. 21st ACM Symposium on Computational Geometry*, pages 323–332, 2012.

[4] Peyman Afshani, Gerth S. Brodal, and Norbert Zeh. Ordered and unordered top-K range reporting in large data sets. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 390–400, 2011.

[5] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM/SIAM Symposium on Discrete Algorithms*, pages 180–186, 2009.

[6] Peyman Afshani, Chris Hamilton, and Norbert Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Computational Geometry*, 43(8):700 – 712, 2010. Special Issue on the 25th Annual Symposium on Computational Geometry (SoCG'09).

[7] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM Journal on Computing*, 29(3):912–953, 2000.

[8] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of $(< k)$-levels in three dimensions. *SIAM Journal on Computing*, 30(2):561–575, 2000.

[9] Timothy M. Chan. Persistent predecessor search and orthogonal point location in the word RAM. In *Proc. 22nd ACM/SIAM Symposium on Discrete Algorithms*, pages 1131–1145, 2011.

[10] Timothy M. Chan, Kasper G. Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Symposium on Computational Geometry*, pages 1–10, 2011.

[11] Timothy M. Chan and Bryan Wilkinson. Adaptive and approximate orthogonal range counting. In *Proc. 24th ACM/SIAM Symposium on Discrete Algorithms*, pages 241–251, 2013.

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

[13] Christos Makris and Konstantinos Tsakalidis. An improved algorithm for static 3D dominance reporting in the pointer machine. In *Proc. 23rd International Symposium on Algorithms and Computation*, pages 568–577, 2012.

[14] Jiří Matoušek. Cutting hyperplane arrangements. *Discrete and Computational Geometry*, 6:385–406, 1991.

[15] Jiří Matoušek. Reporting points in halfspaces. *Computational Geometry*, 2(3):169 – 186, 1992.

[16] Yakov Nekrich. A data structure for multi-dimensional range reporting. In *Proc. 23rd ACM Symposium on Computational Geometry*, pages 344–353, 2007.

[17] Yakov Nekrich. Data structures for approximate orthogonal range counting. In *Proc. 20th Latin American Theoretical Informatics Symposium*, volume 5878, pages 183–192, 2009.

[18] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th ACM Symposium on Computational Geometry*, pages 390–399, 1999.

[19] Darren E. Vengroff and Jeffrey S. Vitter. Efficient 3-D range searching in external memory. In *Proc. 28th ACM Symposium on Theory of Computation*, pages 192–201, 1996.