Dynamic Planar Voronoi Diagrams for General Distance Functions and their Algorithmic Applications*

Haim Kaplan† Wolfgang Mulzer‡ Liam Roditty§ Paul Seiferth¶ Micha Sharir‖

October 1, 2020

Abstract

We describe a new data structure for dynamic nearest neighbor queries in the plane with respect to a general family of distance functions. These include L_p-norms and additively weighted Euclidean distances. Our data structure supports general (convex, pairwise disjoint) sites that have constant description complexity (e.g., points, line segments, disks, etc.). Our structure uses $O(n \log^3 n)$ storage, and requires polylogarithmic update and query time, improving an earlier data structure of Agarwal, Efrat and Sharir that required $O(n^\varepsilon)$ time for an update and $O(\log n)$ time for a query [SICOMP, 1999]. Our data structure has numerous applications. In all of them, it gives faster algorithms, typically reducing an $O(n^\varepsilon)$ factor in the previous bounds to polylogarithmic. In addition, we give here two new applications: an efficient construction of a spanner in a disk intersection graph, and a data structure for efficient connectivity queries in a dynamic disk graph.

To obtain this data structure, we combine and extend various techniques from the literature. Along the way, we obtain several side results that are of independent interest. Our data structure depends on the existence and an efficient construction of *"vertical" shallow cuttings* in arrangements of bivariate algebraic functions. We prove that an appropriate level in an arrangement of a random sample of a suitable size provides such a cutting. To compute it efficiently, we develop a randomized incremental construction algorithm for computing the lowest k levels in an arrangement of bivariate algebraic functions (we mostly consider here collections of functions whose lower envelope has linear complexity, as is the case in the dynamic nearest-neighbor context, under both types of norm). To analyze this algorithm, we also improve a longstanding

*A preliminary version appeared as H. Kaplan, W. Mulzer, L. Roditty, P. Seiferth, and M. Sharir. *Dynamic Planar Voronoi Diagrams for General Distance Functions and their Algorithmic Applications*, Proc. 28th SODA, pp. 2495–2504, 2017. Work by Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth has been supported by grants 1161/2011 and (with Micha Sharir) 1367/2016 from the German-Israeli Science Foundation. Work by Haim Kaplan has also been supported by grants 822-10 and 1841-14 from the Israel Science Foundation, and by the Israeli Centers for Research Excellence (I-CORE) program (center no. 4/11). Work by Wolfgang Mulzer and Paul Seiferth has also been supported by grant MU/3501/1 from Deutsche Forschungsgemeinschaft (DFG) and by ERC StG 757609. Work by Micha Sharir has been supported by Grant 2012/229 from the U.S.-Israel Binational Science Foundation, by Grants 892/13 and 260/18 from the Israel Science Foundation, by the Israeli Centers for Research Excellence (I-CORE) program (center no. 4/11), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

†Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; haimk@tau.ac.il.

‡Institut für Informatik, Freie Universität Berlin, Berlin 14195, Germany; mulzer@inf.fu-berlin.de.

§Department of Computer Science, Bar-Ilan University, Ramat Gan 52900, Israel; liamr@macs.biu.ac.il.

¶Institut für Informatik, Freie Universität Berlin, Berlin 14195, Germany; pseiferth@inf.fu-berlin.de.

‖Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; michas@tau.ac.il.

bound on the combinatorial complexity of the vertical decomposition of these levels. Finally, to obtain our structure, we combine our vertical shallow cutting construction with Chan's algorithm for efficiently maintaining the lower envelope of a dynamic set of planes in $\mathbb{R}^3$. Along the way, we also revisit Chan's technique and present a variant that uses a single binary counter, with a simpler analysis and improved amortized deletion time (by a logarithmic factor; the insertion and query costs remain asymptotically the same).

In loving memory of Ricky Pollack, one of the founding fathers of the field, and a dear friend.

1 Introduction

Nearest neighbor searching in the plane is one of the most fundamental problems in computational geometry [8]. Given a finite set S of *sites* in $\mathbb{R}^2$, the goal is to construct a data structure that can find the "closest" site for any given query object. If S is fixed, Voronoi diagrams and their many variants provide a simple and well-understood solution [5, 8], with linear storage and logarithmic query time. However, in many applications, the set S may change dynamically as sites get inserted and deleted. Now, we want to answer nearest-neighbor queries interleaved with the updates. This setting is much less understood.

If S consists of singleton points and distances are measured in the Euclidean metric, we can achieve polylogarithmic update and query time [13, 14], with $O(n \log^3 n)$ storage. However, we are often confronted with more general distance functions (e.g., L_p-norms or additively weighted Euclidean distances). Examples include the dynamic maintenance of a bichromatic closest pair of sites, constructing a Euclidean minimum-weight red-blue matching, constructing a Euclidean minimum spanning tree, computing the intersection of unit balls in three dimensions, or computing the smallest stabbing disk of a family of simply shaped compact strictly-convex sets in the plane; as well as computing a single-source shortest-path tree in a unit-disk graph (see Section 9 for details and references). Despite the numerous motivating applications, there has been virtually no progress on the basic problem since the 1990s. The state of the art is work by Agarwal et al. from 1999 [3]. It provides $O(n^\varepsilon)$ update and $O(\log n)$ query time, for any fixed $\varepsilon > 0$, while using $O(n^{1+\varepsilon})$ storage.[1] We present a new solution that gives polylogarithmic update and query time, while using $O(n \log^3 n)$ storage, for a wide range of distance functions. We assemble a broad set of techniques, such as randomized incremental construction, relative (p, ε)-approximations, shallow cuttings for xy-monotone surfaces in $\mathbb{R}^3$, and several advanced data structuring techniques.

We now describe our notions more thoroughly. Let S be a set of n pairwise disjoint *sites*. Each site is a simply-shaped compact convex region in the plane (points, line segments, disks, etc.). Let $\delta : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$ be a continuous *distance function* between points in the plane. For a site $s \in S$, define the *distance to* s, $f_s : \mathbb{R}^2 \to \mathbb{R}_{\geq 0}$, as $f_s(x, y) = \delta((x, y), s) = \min_{p \in s} \delta((x, y), p)$ (the minimum exists since s is compact and δ is continuous). We assume that δ and the sites in S have *constant description complexity*. This means that they are defined by a constant number of polynomial equations and inequalities of constant maximum degree. Set $F = \{f_s \mid s \in S\}$. The *lower envelope* $\mathcal{E}_F$ of F is the pointwise minimum $\mathcal{E}_F(x, y) = \min_{f \in F} f(x, y)$, and its xy-projection is called the *minimization diagram* of F, denoted by $\mathcal{M}_F$. The *combinatorial complexity* of $\mathcal{E}_F$ or of $\mathcal{M}_F$ is the total number of their vertices, edges and faces. The book by Sharir and Agarwal [48] provides a comprehensive treatment of these concepts.

[1] Here and later, the constants in such bounds depend on ε.

Now, given a query point $q \in \mathbb{R}^2$, in order to find a δ-nearest neighbor for q in S, we must identify a site s with $\mathcal{E}_F(q) = f_s(q)$. This translates to a *vertical ray shooting query* in $\mathcal{E}_F$: find the intersection of $\mathcal{E}_F$ and the z-vertical line through q, or, alternatively, locate q in the planar map $\mathcal{M}_F$, where each two-dimensional face $\varphi \in \mathcal{M}_F$ is labeled with the site s for which f_s attains the minimum over φ. (Edges and vertices can be labeled by the set of labels of their adjacent faces.)

The structure and the complexity of $\mathcal{E}_F$ and of $\mathcal{M}_F$, as well as algorithms for their construction and manipulation, have been studied for several decades (again, see [48]). To summarize, under the above assumptions, the combinatorial complexity of $\mathcal{E}_F$ (or of $\mathcal{M}_F$) is $O(n^{2+\varepsilon})$, for any fixed $\varepsilon > 0$.[2] However, in many interesting cases, including the case where the functions f_s are linear (i.e., their graphs are non-vertical planes), the complexity of $\mathcal{E}_F$ is $O(n)$. The case of planes arises, after simple algebraic manipulations, for point sites under the Euclidean distance. Then $\mathcal{M}_F$ is the Euclidean Voronoi diagram of S. There are many variants of Voronoi diagrams, for other classes of sites and distance functions, for which the complexity of $\mathcal{E}_F$ remains linear; see, e.g., the book by Aurenhammer, Klein, and Lee [5].

Coming back to nearest neighbor search, if we assume that $\mathcal{E}_F$ has linear complexity and can be constructed efficiently, all we need to do, in the so-called "static" case, is to preprocess $\mathcal{M}_F$ for fast planar point location. Then, a query takes $O(\log n)$ time. If sites in S can be inserted or deleted, i.e., if F changes dynamically, then $\mathcal{E}_F$ may change rather drastically after an update. Maintaining an explicit representation of $\mathcal{M}_F$ thus becomes overwhelmingly expensive. Hence, the goal, in this paper and in earlier work, is to store an *implicit* representation of $\mathcal{E}_F$ that still supports efficient vertical ray shooting in the current envelope $\mathcal{E}_F$ (or point location in the current $\mathcal{M}_F$).

In all the applications of dynamic nearest neighbor search that are studied in this paper, the lower envelope $\mathcal{E}_F$ has linear complexity. This is typically the case when S consists of point sites, and the distance functions are typically L_p-metrics, for $1 \leq p \leq \infty$, or additively weighted Euclidean metrics, where each point site $s \in S$ has a weight $w_s \in \mathbb{R}$, and $\delta(q, s) = |qs| + w_s$, where $|qs|$ is the Euclidean distance between q and s. See, e.g., [5, 37] for details concerning the linear complexity of $\mathcal{E}_F$ in these cases. The lower envelope also has linear complexity for general classes of pairwise-disjoint compact convex sites of constant description complexity, and for fairly general metrics.

Our main result is an efficient data structure that dynamically maintains a set F of bivariate functions as above, under insertions and deletions of functions, and supports efficient vertical ray shooting queries into $\mathcal{E}_F$. Assuming, as above, that the complexity of $\mathcal{E}_F$ is linear, the worst-case cost of a query, as well as the amortized expected cost of an update, is polylogarithmic, and the storage used by the structure is $O(n \log^3 n)$. As a consequence, we obtain faster solutions to all the applications mentioned above, and many more, essentially reducing an $O(n^{\varepsilon})$ factor in the complexity to polylogarithmic. Our results also generalize to the case where the lower envelope complexity is not linear.

A brief context. Suppose first that all functions in F are linear. (As noted, this applies to point sites in the Euclidean metric.) A classical solution for this case is due to Agarwal and Matoušek [4]. They show how to maintain dynamically an implicit representation of $\mathcal{E}_F$, with amortized update time $O(n^{\varepsilon})$, for any fixed $\varepsilon > 0$; vertical ray shooting queries take $O(\log n)$ worst-case time. Here, n denotes an upper bound on $|F|$. The case of more general bivariate functions, as described above, was studied by Agarwal et al. [3]. If $\mathcal{E}_F$ has linear complexity, their technique has amortized update

[2] Again, the constant of proportionality depends on ε.

time $O(n^\varepsilon)$, for any fixed $\varepsilon > 0$, and worst-case query time $O(\log n)$, matching (asymptotically) the result for planes [4].

For more than ten years after the work of Agarwal and Matoušek [4], it was open whether the $O(n^\varepsilon)$ update time can be improved. In SODA 2006, Chan [13] presented an ingenious construction for the case of planes where both the (amortized) update time and the (worst-case) query time are polylogarithmic. More precisely, Chan's structure (combined with the recent deterministic construction of shallow cuttings by Chan and Tsakalidis [16]) supports insertions in $O(\log^3 n)$ amortized time, deletions in $O(\log^6 n)$ amortized time, and queries in $O(\log^2 n)$ worst-case time. However, before our work, it remained unknown whether a similar result (with polylogarithmic update and query time) is possible for arbitrary bivariate functions with constant description complexity and linear envelope complexity. We provide an algorithm that meets all these performance goals. Along the way, we also improve the deletion time for Chan's data structure for planes by a factor of $\log n$, and the bound of Agarwal et al. [3] for the complexity of the vertical decomposition of the $(\leq k)$-level in an arrangement of surfaces in $\mathbb{R}^3$ by a factor of k^ε. Very recently, after the original submission of our paper, by combining a faster cutting construction with our observations, Chan [14] further improved the amortized deletion time for the case of planes to $O(\log^4 n)$ and the amortized insertion time to $O(\log^2 n)$.

2 Our results and techniques

Our data structure combines a multitude of techniques. We first give a broad overview of how these techniques play together; see Figure 1, where the terms in the figure are explained below.

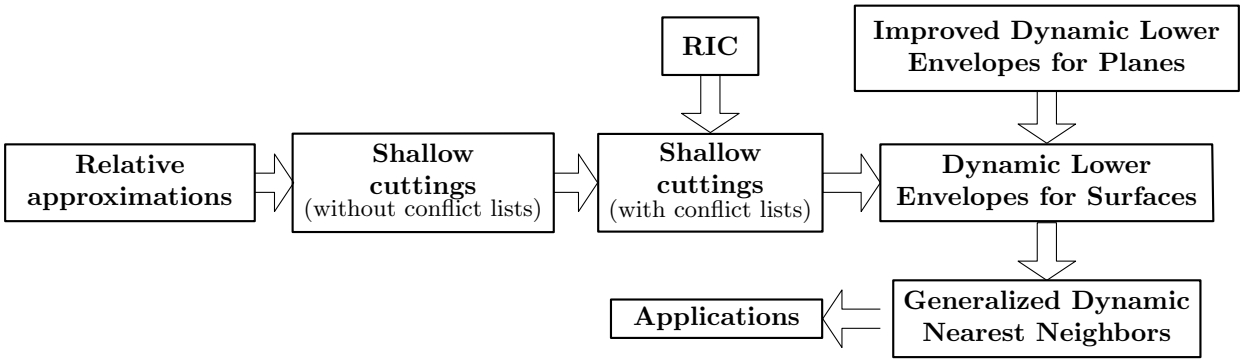

Figure 1: The main tools used for our data structure.

Maybe the most crucial observation is that the whole geometric part in Chan's data structure [13, 14] is in the construction of small *shallow cuttings* for planes. Thus, once we have an analogous result for surfaces, we can maintain their dynamic lower envelope, or equivalently, solve the generalized dynamic planar nearest neighbor problem. It turns out that random sampling and the theory of relative (p, ε)-approximations easily yield a construction for the required cuttings. However, we must also find the corresponding conflict lists quickly. For this, we present an algorithm that uses randomized incremental construction (RIC) for the $(\leq k)$-level in an arrangement of surfaces. Together with an improved variant of Chan's result, this gives the generalized nearest neighbor data structure. We show the impact of this structure by presenting numerous applications thereof, both old and new. In what follows, we describe the specific parts in more detail.

The geometric core of Chan's data structure consists of an efficient construction of small-sized *vertical shallow cuttings* [12, 16]. Let F be a set of n functions in $\mathbb{R}^3$, identified in this paper with their graphs, and let $\mathcal{A}(F)$ denote the *arrangement* of F. We recall the notion of a *k-level* in $\mathcal{A}(F)$, for a parameter $0 \leq k \leq n-1$. It is the closure of the set of points q such that q lies on some function graph and exactly k graphs pass strictly below q.

Roughly speaking (more details follow below), for suitable parameters k and $r \approx n/k$, a *vertical k-shallow $(1/r)$-cutting* is a collection of pairwise openly disjoint semi-unbounded vertical prisms, where each prism consists of all points that lie vertically below some triangle. Furthermore, (i) these top triangles form a polyhedral terrain that is sandwiched between the k-level and the k'-level of the arrangement, for a suitable parameter k' close to k; (ii) the number of prisms is close to $O(r)$; and (iii) each prism is crossed by approximately k function graphs.

Once a fast construction of vertical shallow cuttings of sufficiently small size is available, we can plug it into Chan's machinery for planes [13], in almost black-box fashion. This gives a fast data structure for dynamic maintenance of the lower envelope in the general setting. Agarwal et al. [3] prove the existence of shallow cuttings of optimal size for general functions, but their cuttings are not "vertical", in the above sense, and a direct algorithmic implementation of their ideas yields an additional $O(n^{\varepsilon})$ factor for both the size and the construction time of the cutting. When applied to the dynamic maintenance problem, this gives (amortized) update cost $O(n^{\varepsilon})$ rather than polylogarithmic. Refining this bound is one of the main goals of the present paper.

Thus, we design a different algorithm for computing a vertical shallow cutting. For this, we develop several technical results that we believe to be of independent interest. We use *relative approximations* [30] to show that, with high probability, we get an *ε-approximation* of the k-level of $\mathcal{A}(F)$ by choosing a random sample S_k of $(cn/\varepsilon^2 k)\log n$ functions from F and by taking the t-level of $\mathcal{A}(S_k)$, for $t \in [(1+\varepsilon/3)\lambda,\ (1+\varepsilon/2)\lambda]$, $\lambda = c\varepsilon^{-2}\log n$, and c a suitable constant. This means that any such t-level of $\mathcal{A}(S_k)$ lies between levels k and $(1+\varepsilon)k$ of $\mathcal{A}(F)$. We show that for random t, the expected complexity of the t-level is $O\left((n/\varepsilon^5 k)\log^2 n\right)$.

Having computed such a t-level, we project it onto the xy-plane, construct the standard planar vertical decomposition of the faces of the projection, lift each resulting trapezoid φ back to a trapezoidal subface φ^* embedded in a surface on the original level, and associate it with the semi-unbounded vertical prism that extends below φ^*. We show that this collection of prisms is a vertical k-shallow $(1/r)$-cutting in $\mathcal{A}(F)$ (with k and $r \approx n/k$ as above). We denote it by Λ_k.

The last hurdle is to efficiently compute Λ_k, together with the *conflict lists* of its prisms. The conflict list $\mathrm{CL}(\tau)$ of a prism $\tau \in \Lambda_k$ is the set of all functions $f \in F$ whose graphs cross the interior of τ. (Although the construction of Λ_k is performed with respect to the sample S_k, the conflict lists are defined with respect to the whole set F.)

This leads us to the classical problem of computing the t lowest levels in an arrangement of n bivariate functions of constant description complexity. A standard approach for this goes via *randomized incremental construction* (RIC), see, e.g., [8, 41]. Here, one adds the functions one by one, in random order, while maintaining some representation of the first t levels on the functions inserted so far. Following previous work, we maintain a cell decomposition of the region below the t-level of the function graphs inserted so far, and we associate with each cell a conflict list consisting of all the remaining functions that cross it. If we run this process to completion, we get a suitable decomposition of the t shallowest levels of the "final" $\mathcal{A}(F)$. If we stop after inserting the first $(cn/\varepsilon^2 k)\log n$ functions, which serve as the desired random sample S_k, we obtain, in addition to (a suitable decomposition of) the t shallowest levels of $\mathcal{A}(S_k)$, the conflict lists of its cells (with

respect to the whole F).

Our decomposition of choice is (a suitable shallow portion of) the standard *vertical decomposition* of an arrangement of surfaces in $\mathbb{R}^3$ (see [19, 48] for details). Each prism extends between two consecutive levels of the current arrangement, so this decomposition differs from the vertical shallow cutting that we are after.[3] Nevertheless, we show how to transform this decomposition into a vertical shallow cutting, including the construction of the desired conflict lists of its semi-unbounded prisms. A fairly intricate analysis shows that the shallow cutting has expected complexity $O\left((n/k)\log^2 n\right)$.

The implementation of such a RIC for the shallowest t levels of $\mathcal{A}(F)$ is far from trivial. It has been considered before for the case of planes. Mulmuley [40] described a RIC of the first t levels, when the lower envelope of the planes corresponds to the Voronoi diagram of a set of points in the xy-plane (under the standard algebraic manipulations alluded to above). Mulmuley's procedure needs $O(nt^2\log(n/t))$ expected time.[4] Agarwal et al. [2] used a somewhat less standard randomized incremental algorithm and obtained a bound of $O(n\log^3 n + nt^2)$ expected time. Their algorithm works for any set of planes. It maintains a point p in each prism, such that the level of p in $\mathcal{A}(F)$ is known, and it uses this information to prune away prisms that can be ascertained not to intersect the shallowest t levels of $\mathcal{A}(F)$. Finally, Chan [11] obtained a bound of $O(n\log n + nt^2)$ expected time with an algorithm that can be viewed as a batched randomized incremental construction. Unfortunately, it is not clear how to apply some crucial components of these algorithms when F is a set of nonlinear functions.

We present and analyze a standard randomized incremental construction algorithm for the shallowest t levels of an arrangement $\mathcal{A}(F)$ of a set F of n bivariate functions with constant description complexity and linear envelope complexity. Our algorithm runs in $O\left(nt\log(n/t)\log n\,\lambda_s(t)\right)$ expected time, where s is a constant that depends on the surfaces and $\lambda_s(t)$ is the maximum length of a Davenport-Schinzel sequence on t symbols of order s [48].[5] To get this result, we improve a bound of Agarwal et al. [3] on the complexity of the vertical decomposition of the t shallowest levels in $\mathcal{A}(F)$. Agarwal et al. proved that this complexity is $O(nt^{2+\varepsilon})$, for any fixed $\varepsilon > 0$, via a fairly complicated charging scheme. We improve this to $O(nt\lambda_s(t))$, with a simpler argument, where s is a constant that depends on the algebraic complexity of the functions of F (see below for a precise definition).

Using our randomized incremental algorithm, we construct a vertical shallow cutting of the first k levels in $\mathcal{A}(F)$, consisting of $O\left((n/k)\log^2 n\right)$ prisms, each with a conflict list of size $O(k)$. The construction time is $O(n\log^3 n\,\lambda_s(\log n))$.

Once we have an efficient mechanism for constructing vertical shallow cuttings, we apply it, following and adapting the technique of Chan for the case of planes, to obtain our dynamic data structure. Before that, we re-examine Chan's data structure, and we present it in a way that is easier to understand (in our opinion) and, at the same time, slightly faster than the original version. Our variant follows a standard route: we begin with a static data structure and extend it for insertions, using a (somewhat non-standard) variant of the well-known Bentley-Saxe binary counter technique [7]. Then, we show how to perform deletions via re-insertions of planes, using a *deletion lookahead mechanism*, the major innovation in Chan's work. We believe that our analysis sheds additional light on the inner workings of Chan's structure. We improve the amortized deletion

[3] We distinguish between the two kinds of cuttings by referring to the previous one as vertical, and to the one just introduced simply as a cutting.

[4] $O(nt^2)$ is a tight bound on the complexity of the t shallowest levels in an arrangement of n planes.

[5] As is well known [48], the function $\lambda_s(t)$ is "almost" linear, i.e., $\lambda_s(t) = t\beta_s(t)$ for some extremely slow-growing function $\beta_s(t)$ of inverse-Ackermann type.

time to $O(\log^5 n)$, i.e., by a logarithmic factor. Deletions are the costliest operations in Chan's structure and constitute the bottleneck in most of its applications. As mentioned, in recent work, Chan [14] achieved a further improvement, building upon our analysis, reducing the amortized deletion time to $O(\log^4 n)$ and the amortized insertion time to $O(\log^2 n)$.

We finally combine our shallow cutting construction with our improved version of Chan's data structure, extended to more general functions, to obtain a dynamic data structure for vertical ray shooting into the lower envelope of a dynamically changing set of bivariate functions, as above. Our (worst-case, deterministic) query time is $O(\log^2 n)$, the (amortized, expected) time for an insertion is $O(\log^5 n\, \lambda_s(\log n))$, and the (amortized, expected) time for a deletion is $O(\log^9 n\, \lambda_s(\log n))$. The larger polylogarithmic factors are a consequence of slightly weaker bounds on the complexity of an approximating level.

Plugging our new bounds into the applications in Agarwal et al. [3] and in Chan [13], we immediately improve several running times, replacing a factor of n^ε by a polylogarithmic factor. Some prominent examples are shown in the following table; details follow in Section 9. (Constants of proportionality are suppressed in the table.) The parameter s depends on the precise metric, and is defined in more detail later in the paper. Concrete values of s are given in the table for the specific respective applications.

Problem	Old Bound	New Bound
Dynamic bichromatic closest pair in general planar metric	n^ε update [3]	$\log^5 n\, \lambda_s(\log n)$ insertion, $\log^9 n\, \lambda_s(\log n)$ deletion
Minimum planar bichromatic Euclidean matching	$n^{2+\varepsilon}$ [3]	$n^2 \log^9 n\, \lambda_6(\log n)$
Dynamic minimum spanning tree in L_p-metric	n^ε update [3]	$\log^{11} n\, \lambda_s(\log n)$ update
Dynamic intersection of unit balls in $\mathbb{R}^3$	n^ε update [3] queries in $\log n$ and $\log^4 n$ (depending on the precise query)	$\log^5 n\, \lambda_6(\log n)$ insertion, $\log^9 n\, \lambda_6(\log n)$ deletion, queries in $\log^2 n$ and $\log^5 n$ (depending on precise query)

A particularly fruitful application domain for our data structure can be found in *disk intersection graphs*. These are defined as follows: Let $S \subset \mathbb{R}^2$ be a finite set of point sites, each with an associated weight $w_p > 0$, $p \in S$; a site p with weight w_p represents the disk of radius w_p centered at p. The *disk intersection graph* for S, denoted $D(S)$, has the sites in S as vertices, and there is an edge pq between two sites p, q in S if and only if $|pq| \leq w_p + w_q$, i.e., if the disk around p with radius w_p intersects the disk around q with radius w_q. If all weights are 1, we call $D(S)$ the *unit disk graph* for S. Disk intersection graphs are a popular model for geometrically defined graphs and networks, and enjoy an increasing interest in the research community, in particular due to applications in wireless sensor networks [10, 15, 27, 32, 34, 45]. The following table gives an overview of our results on disk graphs.

Problem	Old Bound	New Bound
Shortest-path tree in a unit disk graph	$n^{1+\varepsilon}$ [10]	$n \log^9 n\, \lambda_6(\log n)$
Dynamic connectivity in disk intersection graphs with radii in $[1, \Psi]$	$n^{20/21}$ update $n^{1/7}$ query [15]	$\Psi^2 \log^9 n\, \lambda_6(\log n)$ update $\log n / \log\log n$ query
BFS tree in disk intersection graphs	$n^{1+\varepsilon}$ [45]	$n \log^9 n\, \lambda_6(\log n)$
$(1+\rho)$-spanners for disk intersection graphs	$n^{4/3+\varepsilon}\rho^{-4/3} \log^{2/3} \Psi$ [27]	$n\rho^{-2} \log^9 n\, \lambda_6(\log n)$

Two of the applications listed above concern finding shortest-path trees in unit disk graphs, and BFS-trees in disk intersection graphs. Our new structures give improved bounds almost in a black-box fashion, using the respective techniques of Cabello and Jejčič [10] and of Roditty and Segal [45]. Very recently, Wang and Xue [50] presented a deterministic algorithm to find the shortest-path tree in a unit disk graph in $O(n \log^2 n)$ time. The other two applications are a bit more involved. First, we give a data structure for the dynamic maintenance of the connected components in a disk intersection graph, as disks are inserted or deleted, where we assume that all disks have radii from the interval $[1, \Psi]$. Then, we can apply our data structure in a grid-based approach that gives an update time that depends on Ψ and is polylogarithmic if Ψ is constant. The previous bound of Chan, Pătraşcu, and Roditty [15] is only slightly sublinear (albeit independent of Ψ). Queries are faster in both approaches, but the bound in [15] is a power of n whereas here it is only sub-logarithmic. Very recently, Kauer and Mulzer [35] presented method that improves the dependence on Ψ. Finally, we give an algorithm for computing a $(1+\rho)$-spanner in a disk intersection graph, for any $\rho > 0$. A $(1+\rho)$-*spanner* for $D(S)$ is a subgraph H of $D(S)$ such that the shortest path distances in H approximate the shortest path distances in $D(S)$ up to a factor of $(1+\rho)$. The previous construction by Fürer and Kasiviswanathan [27] has a running time that depends on the radius ratio Ψ, as defined above. Our new algorithm is independent of Ψ and achieves almost linear running time, improving the previous algorithm by a factor of at least $n^{1/3}$.

Paper outline. Section 3 gives further background and precise definitions. In Section 4, we describe how to obtain a terrain that approximates the k-level of $\mathcal{A}(F)$ by random sampling, via *relative* (p, ε)-*approximations* (see [30] and below). In Section 5, we define a *vertical shallow cutting*, based on our level approximation, and show how to compute it with a randomized incremental construction of the shallowest t levels in $\mathcal{A}(F)$. In Section 6, we describe in detail the randomized incremental construction and analyze it. Section 7 gives our improved variant of Chan's structure for maintaining the lower envelope of planes. Combining our cuttings with Chan's machinery as presented in Section 7, we obtain, in Section 8, an efficient procedure for dynamically maintaining the lower envelope of a collection of algebraic surfaces of constant description complexity and linear lower envelope complexity. Finally, in Section 9, we present several known applications, for which we obtain better bounds, and our new applications for disk graphs. Along the way, we also consider the case where the lower envelope complexity of F is superlinear, and extend our analysis to this more general setup.

3 Preliminaries

Let $\mathcal{F}$ be a family of bivariate functions $f : \mathbb{R}^2 \to \mathbb{R}$, and let F be a finite subset of $\mathcal{F}$. Throughout the paper, we assume that the functions in $\mathcal{F}$ are continuous, totally defined, and algebraic, and that they have *constant description complexity*. This means that the graph of each function is a semialgebraic set, defined by a constant number of polynomial equalities and inequalities of constant maximum degree. We will generally make no distinction between a function $f \in \mathcal{F}$ and its graph $\{(x, y, f(x, y)) \mid x, y \in \mathbb{R}\}$, which is a continuous xy-monotone surface, also called a *terrain*.

The *lower envelope* $\mathcal{E}_F$ of F is the graph of the pointwise minimum of the functions of F. The xy-projection of $\mathcal{E}_F$ yields a subdivision of the xy-plane called the *minimization diagram* $\mathcal{M}_F$ of F. It can be represented by a standard doubly connected edge list (DCEL, see, e.g., [8]). Each (two-dimensional) face f of $\mathcal{M}_F$ is labeled by the function in F that attains the pointwise minimum over f.

When $\mathcal{M}_F$ consists of $O(|F|)$ faces, vertices, and edges, for any finite $F \subseteq \mathcal{F}$, we say that $\mathcal{F}$ has lower envelopes of *linear complexity*. We will mostly assume this to be the case for the families $\mathcal{F}$ considered here. In particular, this assumption holds when $\mathcal{F}$ is the family of all nonvertical planes, or when $\mathcal{F}$ is a family of distance functions under some metric (or under some so-called *convex distance function* [20]), each of which measures the distance of a point in the xy-plane to some given site (cf. Section 9), where the sites are assumed to be pairwise disjoint closed convex sets.

For simplicity, we also assume that F is in *general position*, i.e., no more than three functions meet at a common point, no more than two functions meet in a one-dimensional curve, no pair of functions are tangent to each other, and no function is tangent to the intersection curve of two other functions. For example, this holds if the coefficients of the polynomials defining the functions in F are algebraically independent over $\mathbb{R}$ [48]. Furthermore, we assume that the coordinate frame is generic, so that the xy-projections of the intersection curves of any pair of functions in F are also in general position, defined in an analogous sense.

Model of computation. We assume a (by now fairly standard) algebraic model of computation, in which primitive operations that involve a constant number of functions of $\mathcal{F}$ take constant time. Such operations include: computing the intersection points of any three functions, computing (a suitable representation of) the intersection curve of any two functions, decomposing it into connected components, finding a representative point on each such component, computing the points of intersection between the xy-projections of two intersection curves, testing whether a point lies below, on, or above a function graph, and so on. This model is reasonable, because there are standard techniques in computational algebra (see, e.g., [6, 46]), and actual packages (such as the one described by Boissonnat and Teillaud [9]), that perform such operations exactly in constant time. Technically, these methods and packages determine the truth value of any Boolean predicate of constant description complexity. That is, they are not expected to provide precise values of roots of polynomial equations, but they can determine, exactly and in constant time, any algebraic relation between such roots and/or similar entities, expressed by a constant number of polynomial equations and inequalities of constant maximum degree.

Shallow cuttings. Let $\mathcal{A}(F)$ be the arrangement of a set F of n bivariate functions from $\mathcal{F}$ in $\mathbb{R}^3$. The *level* of a point $q \in \mathbb{R}^3$ in $\mathcal{A}(F)$ is the number of functions of F that pass strictly below

q. For $0 \leq k \leq n-1$, the *k-level* $L_k(F)$ of $\mathcal{A}(F)$ is the closure of the set of all points at level k that lie on a function in F. We denote by $L_{\leq k}(F)$ the union of the first k levels of $\mathcal{A}(F)$. For given parameters $k \in \{0, \ldots, n-1\}$, $r \in \{1, \ldots, n\}$, a *k-shallow $(1/r)$-cutting* in $\mathcal{A}(F)$ is a collection Λ of pairwise openly disjoint regions τ, each of constant description complexity, so that the union of all $\tau \in \Lambda$ covers $L_{\leq k}(F)$, and so that the interior of each $\tau \in \Lambda$ is intersected by at most n/r functions in F. The *size* of Λ is the number of regions in Λ.

In addition, we call Λ *vertical* if every region $\tau \in \Lambda$ is a (semi-unbounded) *pseudo-prism*. A pseudo-prism τ of this kind consists of all points that lie vertically below some *pseudo-trapezoid* $\overline{\tau}$ on a function $f \in F$. Such a pseudo-trapezoid is defined as the set

$$\overline{\tau} = \{(x, y, f(x, y)) \mid x^- \leq x \leq x^+, \psi^-(x) \leq y \leq \psi^+(x)\},$$

for real numbers $x^- < x^+$ and (semi-)algebraic functions ψ^-, $\psi^+ : \mathbb{R} \to \mathbb{R}$ of constant description complexity; some of these boundary constraints might be absent. For planes, $\overline{\tau}$ will simply be a planar y-vertical trapezoid, and we do not insist that $\overline{\tau}$ be contained in one of the input planes. Since the interior of a pseudo-prism in a vertical k-shallow $(1/r)$-cutting is intersected by at least k functions, we must have $k \leq n/r$. In our setting, we will set $r = \Theta(n/k)$, with a sufficiently small constant of proportionality, which is the case most relevant for all our applications.

Matoušek [39] proved that for any n hyperplanes in $\mathbb{R}^d$, there is a k-shallow $(1/r)$-cutting of size $O\big(q^{\lceil d/2 \rceil} r^{\lfloor d/2 \rfloor}\big)$, for $q = k(r/n) + 1$. For the most relevant case $k = \Theta(n/r)$, we get $q = O(1)$ and a cutting of size $O\big(r^{\lfloor d/2 \rfloor}\big)$, a significant improvement over the general bound $O(r^d)$ for a cutting that covers all of $\mathcal{A}(F)$, rather than just $L_{\leq k}(F)$ [17].[6] For example, for planes in three dimensions, we get cuttings of size $O(r)$ instead of $O(r^3)$. This has led to improved solutions of many range searching and related problems (see, e.g., [16] and the references therein). Matoušek [39] presented a deterministic algorithm to construct a shallow cutting in polynomial time. If $r < n^\delta$, the running time improves to $O(n \log r)$, where $\delta > 0$ is a sufficiently small constant depending on d. Later, Ramos [44] presented a (rather complicated) randomized algorithm for $d = 2, 3$, that constructs a hierarchy of shallow cuttings for a geometric sequence of $O(\log n)$ values of r (and $k = \Theta(n/r)$), in $O(n \log n)$ overall expected time. Recently, Chan and Tsakalidis [16] gave a *deterministic* algorithm for the same task. Their algorithm can be stopped early, to obtain an $O(n/r)$-shallow $(1/r)$-cutting in $O(n \log r)$ time. Interestingly, the analysis of Chan and Tsakalidis uses Matoušek's theorem on the existence of an $O(n/r)$-shallow $(1/r)$-cutting of size $O(r)$ as a black box.

Chan [12] was the first to point out the existence of vertical shallow cuttings for planes in $\mathbb{R}^3$. Such a cutting originates from a polyhedral triangulated xy-monotone terrain that lies entirely above $L_{\leq k}(F)$, so that each triangle $\overline{\tau}$ of the terrain generates a semi-unbounded triangular prism τ with $\overline{\tau}$ as its top face. These shallow cuttings have many applications, in particular in Chan's data structure for dynamic maintenance of lower envelopes [13,14], as reviewed above. The deterministic construction of Chan and Tsakalidis [16] constructs vertical shallow cuttings. Recently, Har-Peled, Kaplan, and Sharir [29] gave an alternative construction with additional favorable properties.[7]

Things become technically more involved when we allow more general algebraic functions. For example, decomposing cells of the arrangement into subcells of constant description complexity is easy for hyperplanes (where the subcells are simplices), using, e.g., a *bottom-vertex triangulation* [18, 22]. For general curves or surfaces, the only known general-purpose cell decomposition technique is

[6] These bounds are not known for general surfaces.

[7] One significant difference is that the "top terrain" in [29] approximates the corresponding level k up to any specified accuracy, whereas the structure in [16] does not.

vertical decomposition [19,48]. In the plane, the complexity of such a decomposition is proportional to the complexity of the original arrangement, and in three and four dimensions, near (but not quite) optimal upper bounds are known [19,36]. However, in dimension five and higher, there are significant gaps between the known upper and lower bounds [19]. Regarding shallow cuttings for general surfaces, we are aware only of the aforementioned result of Agarwal et al. [3], and of no work that considers *vertical* shallow cuttings for this general setup.

4 Approximate k-levels

Here and in the following section, we show the existence of vertical shallow cuttings for surfaces. Later, we address the issue of how to efficiently compute these cuttings and the conflict lists of their pseudo-prisms.

Let $\mathcal{F}$ be a family of functions as in Section 3, and let F be a collection of n functions from $\mathcal{F}$. Recall that we assume that the lower envelope complexity of $\mathcal{F}$ is linear. Agarwal et al. [3] provide a shallow cutting for $\mathcal{A}(F)$, and show that, for any fixed $\varepsilon > 0$, $0 \le k \le n-1$, and $1 \le r \le n$, there is a k-shallow $(1/r)$-cutting of size $O(q^{2+\varepsilon}r)$, where $q = kr/n+1$ (and the constant of proportionality depends on ε). This is slightly sub-optimal when q is large. However, we are interested in the case $r \approx n/k$, so $q = O(1)$ and the bound becomes $O(r)$, which is optimal. Nonetheless, the cutting of Agarwal et al. [3] is not vertical, and is therefore useless for our purposes.

Known techniques for computing vertical shallow cuttings for planes, and the conflict lists of their prisms [16,29], crucially rely on the fact that if a plane intersects a semi-unbounded prism τ, it must also intersect a vertical edge of τ. This is not true for general functions. Thus, we use a somewhat different approach that results in cuttings whose size is a (small) polylogarithmic factor off optimal. It is an interesting challenge to tighten the bound. For the time being, though, we are not aware of any alternative construction that meets our specific needs.

Let $0 < \varepsilon \le 1/2$ be a specified error parameter,[8] and let $0 \le k \le n-1$. We will approximate the level $L_k(F)$ of $\mathcal{A}(F)$ by a terrain $\overline{T}_k$ (which will actually be a level in the arrangement of some sample of F), with the following properties (for simplicity, we ignore in this paper the trivial issue of rounding).

1. $\overline{T}_k$ fully lies above $L_k(F)$ and below $L_{(1+\varepsilon)k}(F)$.
2. The complexity $|\overline{T}_k|$ of $\overline{T}_k$ is $O\left((n/\varepsilon^5 k)\log^2 n\right)$.

To construct $\overline{T}_k$, we use the notion of *relative* (p,ε)*-approximation* (see Har-Peled and Sharir [30] for more details): for a range space $(X,\mathcal{R})$ of finite VC-dimension, and for given parameters $p,\varepsilon \in (0,1)$, a set $A \subseteq X$ is called a relative (p,ε)-approximation, if, for each range $R \in \mathcal{R}$, we have

$$\left|\frac{|R\cap X|}{|X|} - \frac{|R\cap A|}{|A|}\right| \le \begin{cases} \varepsilon\frac{|R|}{|X|}, & \text{if } |R| \ge p|X| \\ \varepsilon p, & \text{if } |R| < p|X|. \end{cases} \tag{1}$$

As shown by Har-Peled and Sharir [30] (following Li et al. [38], see also Har-Peled's book [28]), for any $q \in (0,1)$, a random sample of size

$$O\left(\frac{1}{\varepsilon^2 p}\left(\log\frac{1}{p} + \log\frac{1}{q}\right)\right)$$

[8] If ε is constant (say, $\varepsilon = 1/2$), the dependence on ε can be suppressed. Nonetheless, we include it in the interest of precision, and in anticipation of future applications that might require closer level approximations.

is a relative (p, ε)-approximation with probability at least $1-q$, where the constant of proportionality depends linearly on the VC-dimension, but is independent of ε and p.

We apply this general machinery to the range space $(\mathcal{F}, \mathcal{R})$ defined as follows. An *object* o can be either a straight line, a segment, a ray, or an edge in the arrangement of a constant number of functions of $\mathcal{F}$, a face in such an arrangement, a connected portion of such a face cut off by vertical planes orthogonal to the x-axis, or a connected component of the intersection of such a face with a plane orthogonal to the x-axis. Each range $R \in \mathcal{R}$ corresponds to an object o as above, and is the set of functions of $\mathcal{F}$ that intersect o. The fact that $(\mathcal{F}, \mathcal{R})$ has finite VC-dimension, follows by standard arguments (see, e.g., [28, 48]). Thus, let $S_k \subseteq F$ be a random sample of size

$$r_k = \frac{cn}{\varepsilon^2 k} \log n, \tag{2}$$

where $c > 0$ is a suitable constant, proportional to the VC-dimension of $(\mathcal{F}, \mathcal{R})$. By the previous discussion, we can chose c such that S_k is a relative $(k/2n, \varepsilon/3)$-approximation for $(\mathcal{F}, \mathcal{R})$, with probability at least $1 - 1/n^b$, for some sufficiently large constant $b \geq 1$. Note that for this choice of r_k to make sense, we need $k = \Omega\left(\varepsilon^{-2} \log n\right)$. The case of smaller k is simpler and will be treated below.

Set $\overline{T}_k$ to a *random* level $L_t(S_k)$, where t is chosen uniformly in the range

$$\left[\left(1 + \frac{\varepsilon}{3}\right)\lambda,\ \left(1 + \frac{\varepsilon}{2}\right)\lambda\right], \text{ for } \lambda = c\varepsilon^{-2} \log n.$$

We refer to $\overline{T}_k$ as an *ε-approximation* to level $L_k(F)$. This terminology is justified in the following lemmas. From now on, we will assume that S_k is indeed a relative $(k/2n, \varepsilon/3)$-approximation. The bounds in Lemmas 4.2 and 4.3 hold notwithstanding, since the assumption fails with probability at most $1/n^b$, so that, by making b sufficiently large (as we can), the event of failure contributes a negligible amount to the relevant expectation.

Lemma 4.1. *The terrain $\overline{T}_k$ lies between levels k and $(1+\varepsilon)k$ of $\mathcal{A}(F)$.*

Proof. Let p be a point of level k in $\mathcal{A}(F)$, and let $R^{(p)}$ denote the range of those functions that pass below p. By assumption, S_k is a relative $(k/2n, \varepsilon/3)$-approximation for a range space that includes $R^{(p)}$. Since

$$\frac{k}{2n} < \frac{k}{n} = \frac{|R^{(p)}|}{n},$$

the first case in (1) implies that

$$|R^{(p)} \cap S_k| \leq \left(1 + \frac{\varepsilon}{3}\right) \frac{k}{n} r_k \leq \left(1 + \frac{\varepsilon}{3}\right) \lambda.$$

Thus, at most $\left(1 + \frac{\varepsilon}{3}\right) \lambda$ functions of S_k pass below p, i.e., p lies on or below $\overline{T}_k$. Similarly, let q be a point of level $(1+\varepsilon)k$ in $\mathcal{A}(F)$. By a symmetric argument, at least

$$\left(1 - \frac{\varepsilon}{3}\right)(1+\varepsilon)\frac{k}{n} r_k \geq \left(1 + \frac{\varepsilon}{2}\right) \lambda$$

functions of S_k pass below q (using $\varepsilon \leq 1/2$). Hence, q lies on or above $\overline{T}_k$, and the lemma follows. □

Lemma 4.2. *The expected number of vertices p of $\mathcal{A}(S_k)$ whose level in $\mathcal{A}(S_k)$ is between $(1+\varepsilon/3)\,\lambda$ and $(1+\varepsilon/2)\,\lambda$ is $O\left((n/\varepsilon^6 k)\log^3 n\right)$.*

Proof. Let p be a vertex of $\mathcal{A}(F)$, and let $\ell_{S_k}(p)$ denote the level of p in $\mathcal{A}(S_k)$. As follows from the proof of Lemma 4.1, the vertex p can satisfy $(1+\varepsilon/3)\,\lambda \le \ell_{S_k}(p) \le (1+\varepsilon/2)\,\lambda$ only if the level of p in $\mathcal{A}(F)$ lies lies between k and $(1+\varepsilon)k$. The probability that p shows up in $\mathcal{A}(S_k)$ is[9]

$$\frac{\binom{n-3}{r_k-3}}{\binom{n}{r_k}} \approx \left(\frac{r_k}{n}\right)^3 = O\left(\frac{\log^3 n}{\varepsilon^6 k^3}\right).$$

As shown by Clarkson and Shor [23], there are $O(n((1+\varepsilon)k)^2) = O(nk^2)$ vertices in $L_{\le(1+\varepsilon)k}(F)$. Hence, the expected number of vertices p of $\mathcal{A}(S_k)$ with $(1+\varepsilon/2)\,\lambda \le \ell_{S_k}(p) \le (1+\varepsilon/2)\,\lambda$ is at most $O\left((n/\varepsilon^6 k)\log^3 n\right)$, as claimed. □

Lemma 4.3. *The expected complexity of $\overline{T}_k$, over the random choices of S_k and of the level in $[(1+\varepsilon/3)\,\lambda,\ (1+\varepsilon/2)\,\lambda]$, is*

$$O\left(\frac{n}{\varepsilon^5 k}\log^2 n\right). \tag{3}$$

Proof. Since a level in $\mathcal{A}(S_k)$ is an xy-monotone terrain, and since each vertex of $\mathcal{A}(S_k)$ appears in only three (consecutive) levels, the sum of the complexities of all the $L_j(S_k)$, for $j \in \left[\left(1+\frac{\varepsilon}{3}\right)\lambda,\ \left(1+\frac{\varepsilon}{2}\right)\lambda\right]$, is proportional to the number of vertices in $\mathcal{A}(S_k)$ with level between $\left(1+\frac{\varepsilon}{3}\right)\lambda$ and $\left(1+\frac{\varepsilon}{2}\right)\lambda$. Thus, by Lemma 4.2, the expected complexity of a random level in this range is

$$\frac{1}{\varepsilon\lambda/6}\cdot O\left(\frac{n}{\varepsilon^6 k}\log^3 n\right) = \frac{1}{\frac{\varepsilon}{6}\cdot c\varepsilon^{-2}\log n}\cdot O\left(\frac{n}{\varepsilon^6 k}\log^3 n\right) = O\left(\frac{n}{\varepsilon^5 k}\log^2 n\right),$$

as claimed. □

Finally, we discuss the case $k = O(\varepsilon^{-2}\log n)$. In this case, we pick t uniformly at random in the interval $[k,(1+\varepsilon)k]$, and we set $\overline{T}_k$ to $L_t(F)$. By construction, it is clear that $\overline{T}_k$ approximates the k-level in $\mathcal{A}(F)$. Furthermore, the same Clarkson-Shor bound used in the proofs of Lemma 4.2 and 4.3 shows that $\overline{T}_k$ has expected complexity $O(nk/\varepsilon) = O\left((n/\varepsilon^5 k)\log^2 n\right)$, using our assumption on k.

Remark. The same result holds for general lower envelope complexity. Suppose that every set of m functions in $\mathcal{F}$ has lower envelope complexity at most $\psi(m)$, where we assume (or require) that $m \mapsto \psi(m)/m$ is monotonically increasing. Then, given a set F of n functions from $\mathcal{F}$, for every $\varepsilon \in (0,1/2]$ and for every $0 \le k \le n-1$, we can find a terrain $\overline{T}_k$ that lies fully between $L_k(F)$ and $L_{(1+\varepsilon)k}(F)$ and that has complexity $O(\psi(n/k)\varepsilon^{-5}\log^2 n)$.

Indeed, the argument proceeds as above, with slightly adjusted bounds. The Clarkson-Shor bound in the proof of Lemma 4.2 now shows that there are

$$O((1+\varepsilon)^3 k^3 \psi(n/(1+\varepsilon)k)) = O(k^3\psi(n/k))$$

vertices in $L_{\le(1+\varepsilon)k}(F)$, so the expected number of vertices in $\mathcal{A}(S_k)$ with level between $(1+\varepsilon/3)\lambda$ and $(1+\varepsilon/2)\lambda$ is $O(\psi(n/k)\varepsilon^{-6}\log^3 n)$. Dividing by $\varepsilon\lambda/6$, we obtain the claimed bound.

[9] Here we use the model where we sample a subset of the prescribed size, where all such subsets are equally likely to be drawn. One could also use an alternative common model, in which each function is independently chosen to be in S_k with probability r_k/n. The calculations are slightly different in the latter model, but they lead to the same conclusions and asymptotic bounds.

5 From approximate levels to shallow cuttings

Having obtained an approximate level $\overline{T}_k$ as in Section 4, we would like to turn $\overline{T}_k$ into a shallow cutting for $L_{\leq k}(F)$ by creating for each face $\overline{\varphi}$ of $\overline{T}_k$ a semi-unbounded vertical pseudo-prism φ that consists of the points vertically below $\overline{\varphi}$. For brevity, we will refer to these pseudo-prisms simply as *prisms*, and we denote them by T_k. The only issue is that the faces $\overline{\varphi}$ need not have constant complexity, so that the corresponding prisms might be crossed by too many functions in F.

Thus, we decompose each face $\overline{\varphi}$ of $\overline{T}_k$ into sub-faces of constant complexity, using two-dimensional vertical decomposition. More precisely, we project each face $\overline{\varphi}$ onto the xy-plane, and we decompose the resulting projection $\overline{\varphi}^*$ into y-vertical pseudo-trapezoids by erecting y-vertical segments from each vertex of $\overline{\varphi}^*$ and from each point of vertical tangency on its boundary, extending them either into infinity or until they hit another edge of $\overline{\varphi}^*$. By planarity, the number of pseudo-trapezoids is proportional to the complexity of $\overline{\varphi}$. We lift each resulting pseudo-trapezoid τ^* into a prism τ, consisting of all the points vertically below $\overline{\varphi}$ that project to τ^*.[10] Our cutting Λ_k consists of all these prisms τ, and we denote by $\overline{\Lambda}_k$ the terrain formed by the ceilings $\overline{\tau}$, for $\tau \in \Lambda_k$. Then, $\overline{\Lambda}_k$ is a refinement of $\overline{T}_k$. As we will shortly show, Λ_k is indeed a shallow cutting for $L_{\leq k}(F)$. For each prism $\tau \in \Lambda_k$, the *conflict list* $\mathrm{CL}(\tau)$ is the set of functions of F that intersect τ.

Lemma 5.1. *Λ_k is a shallow cutting of the first k levels of $\mathcal{A}(F)$. It consists (in expectation) of*

$$O(|\Lambda_k|) = O\left(\frac{n}{\varepsilon^5 k}\log^2 n\right)$$

prisms, and each prism in Λ_k intersects at least k and at most $(1+2\varepsilon)k$ graphs of functions of F.

Proof. Let $\tau \in \Lambda_k$, and let p be vertex of the ceiling $\overline{\tau}$ of τ. By Lemma 4.1, the level of p in $\mathcal{A}(F)$ is in $[k, (1+\varepsilon)k]$. Thus, at most $(1+\varepsilon)k$ functions in F pass below all vertices of τ. Furthermore, since $\overline{\tau}$ does not intersect any function in S_k, since S_k is a relative $(k/2n, \varepsilon/3)$-approximation for $(\mathcal{F}, \mathcal{R})$, and since $\overline{\tau}$ induces a range in $\mathcal{R}$, by the second bound in (1), it follows that at most

$$\frac{\varepsilon pn}{3} = \frac{\varepsilon k}{6}$$

functions of F cross $\overline{\tau}$. For any function $f \in F$ that intersects τ either passes below all vertices of $\overline{\tau}$ or crosses $\overline{\tau}$, we get

$$|\,\mathrm{CL}(\tau)| \leq \left(1 + \frac{7\varepsilon}{6}\right) k < (1+2\varepsilon)k.$$

The construction of Λ_k ensures that $|\Lambda_k|$ is proportional to the complexity of $\overline{T}_k$, so, by Lemma 4.3, it satisfies (in expectation) the bound asserted in the lemma. □

Remark. More generally, Agarwal et al. [3] show the following: let $\psi : \mathbb{N} \to \mathbb{N}$ be such that any m functions in $\mathcal{F}$ have a lower envelope of complexity $\psi(m)$. Let $F \subset \mathcal{F}$ be a set of n functions in F. Then, for any $k \in \{1, \ldots, n-1\}$, there exists a k-shallow $\Theta(k/n)$-cutting for F of size $O(\psi(n/k))$. Our techniques also generalize to this case. In particular, we obtain the following result.

[10] A significant difference between the machinery used here and that for the case of planes, as in [29], say, is that in the case of planes we only lift the vertices of the xy-map to the appropriate level (or to an approximation of the level), and each triangular face is lifted to the convex hull of its vertices, which in general is not contained in the level. In contrast, here we lift each pseudo-trapezoidal face from the xy-plane to lie fully on the level.

Lemma 5.2. *For any $k \in \{1, \ldots, n-1\}$, our sampling procedure yields a shallow cutting Λ_k of the first k levels of $\mathcal{A}(F)$. It consists (in expectation) of*

$$O(|\Lambda_k|) = O\left(\varepsilon^{-5}\psi(n/k)\log^2 n\right)$$

prisms, and each prism in Λ_k intersects at least k and at most $(1+2\varepsilon)k$ graphs of functions of F.

Proof. We only need a more general bound on the complexity of $\overline{T}_k$. By Clarkson-Shor, in general, there are $O(\psi(n/k)k^3)$ vertices in $L_{\leq(1+\varepsilon)k}(F)$, so we get the bound $O(\varepsilon^{-6}\psi(n/k)\log^3 n)$ in Lemma 4.2 and $O(\varepsilon^{-5}\psi(n/k)\log^2 n)$ in Lemma 4.3. Now, the result follows as before. □

6 Randomized incremental construction of the $\leq t$ level

Again, let $\mathcal{F}$ be a family of bivariate functions in $\mathbb{R}^3$ with constant description complexity and with linear lower envelope complexity. Let F be a subset of n members of $\mathcal{F}$, which we assume to be in general position, and let $0 \leq t \leq n-1$. Our goal is to construct the first t levels of $\mathcal{A}(F)$. We describe an algorithm with expected running time $O(nt\lambda_s(t)\log(n/t)\log n)$ and with expected storage $O(nt\lambda_s(t))$, where s is a constant that depends on $\mathcal{F}$, and $\lambda_s(t)$ is the familiar Davenport-Schinzel bound [48]. Our algorithm can be used to compute a vertical shallow cutting as prescribed in Section 5, together with the conflict lists of its prisms.[11]

We follow the standard *randomized incremental construction* (RIC) paradigm: we insert the surfaces of F one at a time, in random order, and maintain, after each insertion, the first t levels in the arrangement of the surfaces inserted so far (t stays fixed during the process). Number the elements of F in the random insertion order as $f_1, f_2, \ldots, f_n$, and put $F_i = \{f_1, \ldots, f_i\}$, for $i = 1, \ldots, n$. As is standard in the RIC approach, the algorithm maintains a decomposition (the standard vertical decomposition in our case) of $L_{\leq t}(F_i)$ into cells of constant description complexity (these are not necessarily the semi-unbounded prisms of the vertical shallow cutting that we are after—see below), and keeps the *conflict list* for each cell τ, i.e., the set of all functions in F that cross τ. When the next function f_{i+1} is inserted, the conflict lists can be used to retrieve the cells that are crossed by f_{i+1}. These cells are "destroyed", as they no longer appear in the new decomposition, and are partitioned by f_{i+1} into fragments. These fragments are not necessarily valid prisms for the vertical decomposition of $L_{\leq t}(F_{i+1})$, and may need to be merged and refined into the correct new cells. In addition, we have to construct the conflict lists of the new cells, which are obtained from the conflict lists of the destroyed cells.

6.1 Computing the first t levels

After each insertion, we maintain the *vertical decomposition* $\mathrm{VD}_{\leq t}(F_i)$ of $L_{\leq t}(F_i)$, the first t levels of $\mathcal{A}(F_i)$. We obtain $\mathrm{VD}_{\leq t}(F_i)$ by applying two decomposition stages to each cell of $L_{\leq t}(F_i)$. (We reiterate that this decomposition differs from the vertical shallow decomposition used above, in the sense that its prisms are in general not semi-unbounded; see below.)

We call a cell C of $L_{\leq t}(F_i)$ a *stage-0 cell*. In the first stage, we erect a vertical *wall* within C through each edge e of $L_{\leq t}(F_i)$ on ∂C. Each such wall is the union of all maximal vertical

[11] In more detail, as will be described later, we run the algorithm on the entire set F, but stop it after inserting r_k functions (where r_k is as given in (2)). This will provide us with the conflict lists of the final prisms with respect to the entire set F.

segments that lie in (the closure of) C and pass through the points of e. These walls partition C into *stage-1 cells*. Every stage-1 cell C' has a unique *ceiling* ("top" surface) and a unique *floor* ("bottom" surface). The ceiling and/or floor may be undefined if C' is unbounded. All other facets of C' lie on the vertical walls. The complexity of C' may however still be arbitrarily large. Thus, in the second stage, we take each stage-1 cell C', project it onto the xy-plane, and apply a two-dimensional vertical decomposition to the projection $(C')^*$. That is, as in Section 5, we erect a y-vertical segment through each vertex of $(C')^*$ and through each locally x-extremal point on its boundary. This partitions $(C')^*$ into y-vertical pseudo-trapezoids, and we lift each such pseudo-trapezoid τ back into $\mathbb{R}^3$ by forming the intersection $(\tau \times \mathbb{R}) \cap C'$. This yields a decomposition of C' into prism-like *stage-2 cells* of constant description complexity, referred to, for simplicity, just as *prisms*.[12] More details can be found in [19, 48]. Collectively, all the stage-2 cells, over all cells C and all subcells C', constitute the vertical decomposition $\mathrm{VD}_{\leq t}(F_i)$.

6.1.1 Complexity of the vertical decomposition

As is well known, the complexity of $\mathrm{VD}_{\leq t}(F_i)$ is proportional to the number of tuples (q, e, e'), for e, e' edges of $L_{\leq t}(F_i)$ and for $q \in \mathbb{R}^2$, so that q belongs to the xy-projections of e and e', and the z-vertical line ℓ_q through q meets no other surface of F_i between e and e'. We call (e, e') a *vertically visible* pair, and refer to (q, e, e') as a *triple of vertical visibility*. We assume that the pair (e, e') is ordered such that e lies above e', i.e., we encounter e before e' as we travel along ℓ_q from $z = \infty$ to $z = -\infty$.

The following crucial lemma, which we regard as one of the main contributions of the paper, bounds the complexity of $\mathrm{VD}_{\leq t}(F_i)$. It improves an earlier bound of $O(nt^{2+\varepsilon})$ by Agarwal et al. [3]. We define a parameter s as follows: For any $f_1, f_2, f_3, f_4 \in F$, we let $s(f_1, f_2, f_3, f_4)$ denote the number of co-vertical pairs of points $q \in f_1 \cap f_2$, $q' \in f_3 \cap f_4$. Then $s = s_0 + 2$, where s_0 is the maximum of $s(f_1, f_2, f_3, f_4)$, over all quadruples $f_1, f_2, f_3, f_4 \in F$. By our assumptions on $\mathcal{F}$ (including general position), we have $s = O(1)$, where the constant depends[13] on the complexity of the family $\mathcal{F}$. We use $\lambda_s(t)$ to denote the maximum length of a Davenport-Schinzel sequence of order s on t symbols [48].

Lemma 6.1. *Let F be a set of n functions of $\mathcal{F}$, and let $1 \leq t \leq n-1$. The complexity of $\mathrm{VD}_{\leq t}(F)$ is $O(nt\lambda_s(t))$.*

Proof. Let e be an edge of $L_{\leq t}(F)$, and let $F_e \subseteq F$ be the functions in F that pass vertically below some point on e. Since e is not crossed by any function of F_e, each $f \in F_e$ appears below every point of e, implying that $|F_e| \leq t$. Let V_e be the vertical wall erected downward from e, all the way to $z = -\infty$. Then, the complexity of the upper envelope of F_e, clipped to V_e, is at most $\lambda_{s_0}(t)$. Indeed, using a suitable parametrization of e, the cross-sections of the functions in F_e with V_e are totally defined univariate continuous functions, each pair of which intersect at most s_0 times. This follows from the definition of s_0, since the vertices of the arrangement of these functions are exactly the intersection points of V_e with edges e' of $L_{\leq t}(F)$ that form co-vertical pairs (e, e') with e. Since the vertices of the upper envelope of these functions stand in a 1-1 correspondence with the triples of vertical visibility pairs with e as the top edge, the number of these pairs is at most $\lambda_{s_0}(t)$, as claimed.

[12] Each prism is bounded by at most six surfaces of constant desciption complexity—see below.

[13] In certain applications we can, and will, derive concrete bounds on s.

A standard application of the Clarkson-Shor technique implies that $L_{\leq t}(F)$ has $O(t^3 \cdot (n/t)) = O(nt^2)$ edges. This follows by charging the edges to their endpoints and by using the fact that there are $O(m)$ vertices on the lower envelope of any m functions of F. This already gives a (weak) bound of $O(nt^2\lambda_{s_0}(t)) \approx nt^3$ on the complexity of $\mathrm{VD}_{\leq t}(F)$.

The arguments so far follow the initial part of the analysis of Agarwal et al. [3], but the next part is new and gives a sharper bound. Fix two functions $f, f' \in F$, and let $\gamma = f \cap f'$ be their intersection curve. We cut γ at each singular and locally x-extremal point. This decomposes γ into $O(1)$ connected x-monotone Jordan subarcs. Recall that, in addition to general position of F, we also assume a generic coordinate frame, so that no resulting piece lies within some yz-parallel plane.

We cut these arcs further at their intersections with the level $L_t(F)$, and we keep those portions that lie in $L_{\leq t}(F)$. To control the number of such portions, we relax the problem a bit, replacing the level t by a larger level t' with $t \leq t' \leq 2t$, for which the complexity of $L_{t'}(F)$ is $O(nt)$. Since the overall complexity of $L_{\leq 2t}(F)$ is $O(nt^2)$ (as just noted), the average complexity of a level between t and $2t$ is indeed $O(nt)$. Thus, there is a level t' with the above properties. We will establish the asserted upper bound for $\mathrm{VD}_{\leq t'}(F)$, which then also applies to $\mathrm{VD}_{\leq t}(F)$. To keep the notation simple, we continue to denote the top level t' as t.

Let Γ be the set of all Jordan subarcs of some intersection curve that lies in $L_{\leq t}(F)$ (now with the new, potentially larger, index t). If $\gamma \in \Gamma$ does not fully lie below $L_t(F)$, it ends in at least one vertex of $L_t(F)$, so the number of these $\gamma \in \Gamma$ is $O(nt)$. Any other $\gamma \in \Gamma$ is charged either to one of its endpoints, or, if it is unbounded (and x-monotone), to its intersection with a plane at infinity, say $V_\infty : x = +\infty$. If γ reaches V_∞, it appears there as a vertex of the first t levels of the cross-sections of the functions in F with V_∞. An application of the Clarkson-Shor technique to this planar arrangement shows that there are $O(nt)$ such vertices, so this also bounds the number of these arcs in Γ. Finally, we bound the number of $\gamma \in \Gamma$ with a singular or locally x-extremal endpoint by charging γ to this endpoint. The number of these points lying in $L_{\leq t}(F)$ is bounded by yet another application of the Clarkson-Shor technique. Noting that each such point is now defined by only two functions of F, this leads to the upper bound $O(nt)$. Thus, $|\Gamma| = O(nt)$.

Fix an arc $\gamma \in \Gamma$, and let $\mu(\gamma)$ be the number of edges in $L_{\leq t}(F)$ on γ. In general, $\mu(\gamma) \geq 1$. We decompose γ into $\xi(\gamma) := \lceil \mu(\gamma)/t \rceil$ pieces, each consisting of at most t consecutive edges. By general position, if e_1 and e_2 are consecutive edges along γ, the set of functions of F that appear below e_1 and the set of functions that appear below e_2 differ exactly by the third function incident to the common endpoint of e_1 and e_2. This implies that, for a piece δ of γ, the overall number of functions that appear below δ is at most $2t$. Some of these functions are now only partially defined. Arguing as above, the number of vertically visible pairs whose top edge lies on δ is at most $\lambda_{s_0+2}(2t) = O(\lambda_s(t))$. Hence, the overall number of triples of vertical visibility in $L_{\leq t}(F)$ is

$$\left(\sum_{\gamma\in\Gamma} \xi(\gamma)\right) \cdot O(\lambda_s(t)) \leq \left(\sum_{\gamma\in\Gamma} \left(\frac{\mu(\gamma)}{t} + 1\right)\right) \cdot O(\lambda_s(t)) = \left(\frac{1}{t}\sum_{\gamma\in\Gamma} \mu(\gamma) + |\Gamma|\right) \cdot O(\lambda_s(t)).$$

We have already seen that $|\Gamma| = O(nt)$. Furthermore, $\sum_{\gamma\in\Gamma} \mu(\gamma)$ is simply the number of edges in $L_{\leq t}(F)$, which, as already argued, is $O(nt^2)$. It follows that the number of vertically visible pairs in $L_{\leq t}(F)$ is $O\left(nt\lambda_s(t)\right)$. □

Remark. Our analysis also works if the lower envelope complexity is not necessarily linear. Let $\psi : \mathbb{N} \to \mathbb{N}$ be such that any m functions in $\mathcal{F}$ have a lower envelope of complexity at most $\psi(m)$.

Then we obtain the following bound.

Lemma 6.2. *Let $\mathcal{F}$ be a family of functions with lower envelope complexity bounded by $\psi(m)$, let F be a set of n functions of $\mathcal{F}$, and let $1 \leq t \leq n-1$. The complexity of $\mathrm{VD}_{\leq t}(F)$ is $O(t^2\psi(n/t)\lambda_s(t))$.*

Proof. The proof proceeds exactly as the proof of Lemma 6.1, but with more general bounds for the various structures associated with $\mathcal{A}(F)$. In particular, by the Clarkson-Shor technique, the overall complexity of $L_{\leq 2t}(F)$ now is $O(t^3\psi(n/t))$, and hence we can find a level $t \leq t' \leq 2t$ such that $L_{t'}$ has complexity $O(t^2\psi(n/t))$. The arguments for bounding $|\Gamma|$ remain valid, but now the complexity of the first t levels of the planar arrangement on V_∞, and the number of singular or locally x-extremal points in $L_{\leq t}$, are both $O(t^2\psi(n/t))$.[14] Proceeding with these bounds as before, we obtain the claimed result. □

The randomized incremental construction. Although the high-level description of the randomized incremental construction is fairly routine, the finer details are somewhat intricate, and their description is rather lengthy. We present the construction, with full details, in Appendix A. As we will show, the expected running time of the procedure is proportional to the expectation of

$$\sum_{\tau \in \Pi^*} (1 + |\,\mathrm{CL}(\tau)|) \log n, \tag{4}$$

where Π^* is the set of all prisms that are generated during the incremental process, and where $\mathrm{CL}(\tau)$ is the conflict list of prism τ.

6.2 Analysis

We now bound the expected value of (4). Let Π be the set of all possible pseudo-prisms. That is, we consider all possible sets F_0 of at most six surfaces in F, and for each such F_0, we construct the entire vertical decomposition $\mathrm{VD}(F_0)$ and add the resulting prisms to Π.

We associate two *weights* with each prism $\tau \in \Pi$. The first weight $w_0(\tau)$ is the size of its conflict list, that is $w_0(\tau) = |\,\mathrm{CL}(\tau)|$. The second weight $w^-(\tau)$ equals the number of surfaces that pass fully below τ. For simplicity, we focus on prisms that are defined by exactly six functions; the treatment of prisms defined by fewer functions is fully analogous. Let $\Xi(\tau)$ denote the set of defining functions of τ. As just said, we only consider the case $|\Xi(\tau)| = 6$.

Following a standard approach to the analysis of RICs, we proceed in two steps. First, we estimate the probability that a prism with given weights ever appears in $L_{\leq t}(F_i)$. Then, we estimate the number of prisms τ with weights $w^-(\tau) \leq a$ and $w_0(\tau) \leq b$, using the Clarkson-Shor technique and several other considerations. Finally, we combine the bounds to get the desired estimate on the expected running time and storage of the algorithm.

Estimating the probability of a prism to appear. For the first step, let $\tau \in \Pi$ be a prism with six defining functions and with $w^-(\tau) = a$, $w_0(\tau) = b$. That is, τ has $w^-(\tau) = a$ *lower surfaces* and $w_0(\tau) = b$ *crossing surfaces*. Neither of $w^-(\tau)$, $w_0(\tau)$ counts any of the defining functions of τ, although some of these functions might pass below τ. The number of such 'lower'

[14] To be precise, the bound on the complexity of the first t levels of the planar arrangement on V_∞ is only $O(t^2\lambda_{s'}(n/t))$, where $\lambda_{s'}$ is a Davenport-Schinzel-factor that accounts for the maximum number of times that two surfaces intersect. However, this is dominated by the other contributions, so we ignore this refinement here.

defining functions is always between 1 and 5, because the floor is always such a lower function, and the ceiling is always excluded; see Figure 2.

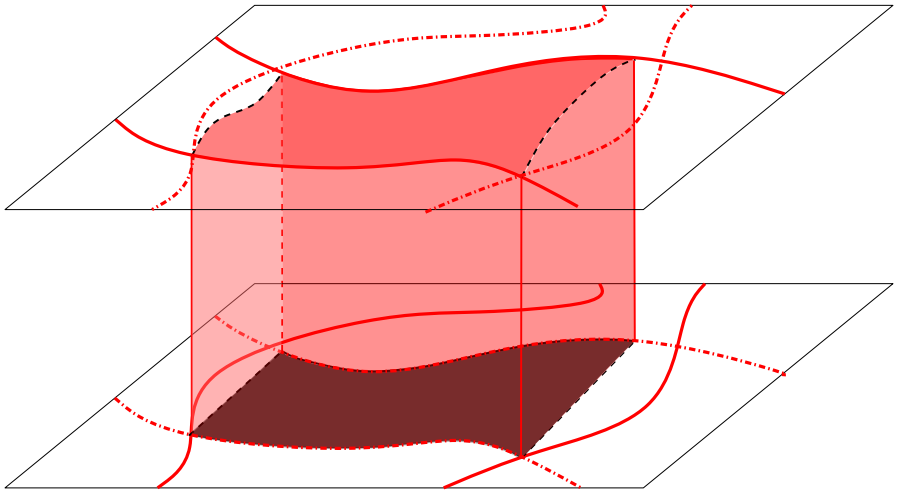

Figure 2: A schematic illustration of a prism with three lower defining functions. The solid lines represent actual intersections between surfaces, the dotted lines represent shadow edges or vertical edges. To belong to $L_{\le t}(F_i)$, at most $t-4$ lower non-defining functions may belong to F_i.

The prism τ appears in some $\mathrm{VD}_{\le t}(F_i)$ if and only if (i) the last function in $\Xi(\tau)$, denoted as f_6, is inserted before any of the b crossing surfaces; and (ii) at most $t' = t - \xi$ of the a lower non-defining surfaces are inserted before f_6, where $\xi \ge 1$ is the number of defining functions of τ that pass below τ, one of which is the floor of τ.

The probability p_τ of this event can be calculated as follows. Restrict the random insertion permutation to the $a+b+6$ relevant surfaces (the a surfaces below τ, the b surfaces crossing τ, and the six surfaces in $\Xi(\tau)$). To get a restricted permutation that satisfies (i) and (ii), we first choose which function in $\Xi(\tau)$ is f_6, then we choose some $j \le \min\{a, t'\}$ of the a lower surfaces to precede f_6, then we mix these j surfaces with the five in $\Xi(\tau) \setminus \{f_6\}$, and finally we place the remaining $a-j$ lower surfaces and all b crossing surfaces after f_6. We thus get

$$p_\tau = \sum_{j=0}^{\min\{a,t'\}} \frac{6\binom{a}{j}(j+5)!(a+b-j)!}{(a+b+6)!}. \tag{5}$$

We rewrite and bound each summand in (5) as follows.

$$\begin{aligned}
\frac{6\binom{a}{j}(j+5)!(a+b-j)!}{(a+b+6)!} &= \frac{6a!(j+5)!(a+b-j)!}{j!(a-j)!(a+b+6)!} \\
&= \frac{6}{a+b+6} \cdot \frac{(a-j+1)\cdots(a-j+b)}{(a+1)\cdots(a+b)} \cdot \frac{(j+1)\cdots(j+5)}{(a+b+1)\cdots(a+b+5)} \\
&\le \frac{6}{a+b+6} \cdot \left(\frac{a-j+b}{a+b}\right)^b \cdot \left(\frac{j+5}{a+b+5}\right)^5 \\
&\le \frac{6}{a+b+6} \cdot \left(\frac{j+5}{a+b+5}\right)^5 \cdot e^{-jb/(a+b)}.
\end{aligned}$$

Let

$$\varphi_{a,b}(j) = \frac{6}{a+b+6} \cdot \left(\frac{j+5}{a+b+5}\right)^5 \cdot e^{-jb/(a+b)}$$

be our bound on the jth summand of (5). Note that with a, b fixed, $\varphi_{a,b}(x)$ peaks at $x = 5(a+b)/b - 5 = 5a/b$ (the positive root of the derivative of $\varphi_{a,b}(x)$ satisfies $5(x+5)^4 - \big(b(x+5)^5\big)/(a+b) = 0$). We estimate p_τ by replacing the sum by an integral. That is,

$$\begin{aligned} p_\tau &= \sum_{j=0}^{\min\{a,t'\}} \frac{6\binom{a}{j}(j+5)!(a+b-j)!}{(a+b+6)!} \\ &\le \sum_{j=0}^{\min\{a,t'\}} \varphi_{a,b}(j) \le e \int_0^{\min\{a,t'\}+1} \varphi_{a,b}(x)dx \qquad (6) \\ &= \frac{6e}{a+b+6} \cdot \int_0^{\min\{a,t'\}+1} \left(\frac{x+5}{a+b+5}\right)^5 \cdot e^{-xb/(a+b)}dx; \end{aligned}$$

to justify the inequality between the sum and the integral in (6), it suffices to note that, for $x \in [j, j+1]$,

$$\frac{\varphi_{a,b}(j)}{\varphi_{a,b}(x)} = \left(\frac{j+5}{x+5}\right)^5 e^{-b(j-x)/(a+b)} \le e^{b/(a+b)} \le e,$$

for every $j \ge 0$. To estimate the integral, we substitute $y = xb/(a+b)$. The upper limit in the integral becomes

$$c := (\min\{a, t'\} + 1) \cdot \frac{b}{a+b},$$

and we get

$$\begin{aligned} p_\tau &\le \frac{6e(a+b)}{b(a+b+6)} \cdot \int_0^c \left(\frac{y(a+b)/b+5}{a+b+5}\right)^5 \cdot e^{-y}dy \\ &= \frac{6e(a+b)}{b(a+b+6)} \cdot \left(\frac{a+b}{b(a+b+5)}\right)^5 \int_0^c (y + 5b/(a+b))^5 \, e^{-y}dy \\ &\le \frac{6e}{b^6} \int_0^c (y + 5b/(a+b))^5 \, e^{-y}dy. \qquad (7) \end{aligned}$$

The integral in (7) is at most

$$\int_0^\infty (y+5)^5 \, e^{-y}dy = O(1).$$

Thus,[15] $p_\tau = O(1/b^6)$. For large c, we cannot improve this bound. However, if c is sufficiently small, bounding the integral in (7) by an absolute constant may be wasteful. For $a \le t$ we will not refine the bound and use $p_\tau = O(1/b^6)$. Consider now the case $a > t > t'$, so $c = (\min\{a, t'\}+1)b/(a+b) = (b(t'+1))/a + b$. As is easily checked, the function $\varphi(y) = (y + 5b/(a+b))^5 \, e^{-y}$ is increasing on $[0, 5a/(a+b)]$, so when

$$c = \frac{b(t'+1)}{a+b} \le \frac{5a}{a+b}, \qquad \text{or} \qquad t' \le \frac{5a}{b} - 1 \ ,$$

[15] Technically, we should write this as $O(1/(b+1)^6)$, to cater also for the case $b = 0$. We gloss over this trifle issue, as is common in other works too, to simplify the notation.

we bound the integral in (7) by $c \cdot \varphi(c)$, and get[16]

$$p_\tau \le \frac{6e}{b^6} \cdot \frac{b(t'+1)}{a+b} \left(\frac{b(t'+6)}{a+b} \right)^5 \cdot e^{-b(t'+1)/(a+b)} = O\left(\frac{t^6}{(a+b)^6} e^{-bt/(a+b)} \right) = O\left(\frac{t^6}{a^6} \right).$$

Thus, we can bound p_τ in terms of a and b. Denoting this bound by $p(a,b)$, we have

$$p(a,b) = \begin{cases} O\left(1/b^6\right), & \text{for } a \le bt/5 \text{ or } a \le t \\ O\left(t^6/a^6\right), & \text{for } a > bt/5 \text{and } a > t. \end{cases} \tag{8}$$

(Unless b is very small, the constraint $a \le t$ or $a > t$ is subsumed by the other respective constraint.)

Bounding the number of prisms of small weights. We next estimate the number of prisms $\tau \in \Pi$ with $w^-(\tau) \le a$ and $w_0(\tau) \le b$. We denote this quantity as $N_{\le a, \le b}$. We also use the notation $N_{a,b}$ for the number of prisms $\tau \in \Pi$ with $w^-(\tau) = a$ and $w_0(\tau) = b$.

Lemma 6.3. *The number of prisms τ with $w^-(\tau) \le a$ and $w_0(\tau) \le b$ is $O(nb^5)$, for $a \le b$, and $O(nab^4 \lambda_s(a/b))$, for $a > b$.*

Proof. Set $p = 1/b$, and let $R \subseteq F$ be a random sample of size pn.

Case 1: $a \le b$. We assume that $b \le n/10$, so that $p = 1/b \ge 10/n$. Fix a prism $\tau \in \Pi$, defined by six functions, with $w^-(\tau) = i$ and $w_0(\tau) = j$, with $i \le a$, $j \le b$. Let q_τ be the probability that R contains (i) the defining set $\Xi(\tau)$; (ii) none of the j crossing functions; and (iii) none of the i lower functions. By elementary probability,

$$\begin{aligned} q_\tau = \frac{\binom{n-6-i-j}{np-6}}{\binom{n}{np}} &\ge \frac{\binom{n-6-a-b}{np-6}}{\binom{n}{np}} \\ &= \prod_{k=0}^{a+b-1} \frac{n-np-k}{n-6-k} \cdot \prod_{l=0}^{5} \frac{np-l}{n-l} \\ &\ge \prod_{k=0}^{a+b-1} \left(1 - \frac{np}{n-k}\right) \cdot \prod_{l=0}^{5} \frac{np-l}{n} \\ &\ge \left(1-\frac{p}{2}\right)^{a+b} \cdot \left(\frac{p}{2}\right)^6 \ge \left(1-\frac{1}{2b}\right)^{2b} \cdot \left(\frac{1}{2b}\right)^6 = \Omega(1/b^6); \end{aligned}$$

this follows since we set $p = 1/b$ and assumed that $a \le b$ and $b \le n/10$, so that $n-k \ge n-a-b \ge n-2b \ge n/2$ and $np - l \ge np - 5 \ge np/2$. If the event holds, τ becomes a prism in $\mathrm{VD}_{\le 6}(R)$. By Lemma 6.1, the number of such prisms is $O(|R|) = O(n/b)$. This yields, as a variant of the Clarkson–Shor theory, $N_{\le a, \le b} = O(nb^5)$. This bound also holds trivially if $b > n/10$, since there are at most $O(n^6)$ prisms in total.

[16] Again, we should write $a+1$ in the final expression.

Case 2: $a > b$. Again, we assume that $b \le n/10$, so we have $p = 1/b \ge 10/n$. Also, we require that n is more than a large enough constant. We put $\xi_0 = 2a/b$ and $\xi = \xi_0 + \nu$, where ν is the number of defining functions of τ that pass below τ. As before, fix a prism $\tau \in \Pi$, defined by six functions, with $w^-(\tau) = i$ and $w_0(\tau) = j$, with $i \le a$, $j \le b$. The probability q_τ that τ appears in $\mathrm{VD}_{\le\xi}(R)$ is the probability that R contains (i) the defining set $\Xi(\tau)$; (ii) none of the j crossing functions; and (iii) at most ξ_0 of the i lower functions. Similarly to Case 1, the probability that (i) and (ii) hold is at least $(p/2)^6(1-p/2)^b = \Omega(1/b^6)$. Conditioned on (i) and (ii) holding, (iii) is the event that when choosing $np-6$ out of $n-6-j$ functions, we obtain at most ξ_0 of the i lower functions. The number of the lower functions in the sample follows a hypergeometric distribution, with expected value

$$i\frac{np-6}{n-6-j} \le i\frac{np}{n-6-b} \le i\frac{11p}{9} \le \frac{11}{9}\frac{a}{b},$$

using our assumption that $b \le n/10$ and that n is large enough. Now, the Chernoff bound for the hypergeometric distribution (see [21] and [42, Theorem 5.2 and Corollary 4.4]) implies that the failure probability, of choosing more than $\xi_0 = 2a/b$ lower functions, is at most $e^{-a/(10b)} \le e^{-1/10}$. Hence, we have $q_\tau = \Omega(1/b^6)$. To complete the Clarkson-Shor analysis, we need an upper bound on the number of prisms in $\mathrm{VD}_{\le\xi}(R)$. By Lemma 6.1, this is $O(|R|\xi\lambda_s(\xi))$. The analysis thus yields

$$N_{\le a,\le b} = O\left(b^6 np\cdot\xi\lambda_s(\xi)\right) = O\left(b^6(n/b)(a/b)\lambda_s(a/b)\right) = O\left(nab^4\lambda_s(a/b)\right),$$

as asserted. Again, the bound holds trivially if $b > n/10$ or if n is constant. □

Remark. As usual, the bounds generalize to superlinear lower envelope complexity. Let $\psi : \mathbb{N} \to \mathbb{N}$ be such that any m functions in $\mathcal{F}$ have a lower envelope of complexity at most $\psi(m)$, and suppose that $m \mapsto \psi(m)/m$ is monotonically increasing. Then we obtain the following bound.

Lemma 6.4. *The number of prisms τ with $w^-(\tau) \le a$ and $w_0(\tau) \le b$ is $O(b^6\psi(n/b))$ for $a \le b$, and $O(a^2b^4\psi(n/a)\lambda_s(a/b))$ for $a > b$.*

Proof. The reasoning is analogous to that in the proof of Lemma 6.3, but we replace the bounds from Lemma 6.1 with the bounds from Lemma 6.2. In particular, in Case 1, the vertical decomposition $\mathrm{VD}_{\le 6}(R)$ has $O(\psi(n/b))$ prisms, and in Case 2, the vertical decomposition $\mathrm{VD}_{\le\xi}(R)$ has

$$O\left((a/b)^2\psi((n/b)(b/2a))\lambda_s(2a/b)\right) = O\left((a/b)^2\psi(n/a)\lambda_s(a/b)\right)$$

prisms. (Since we assumed that $\psi(m)/m$ is increasing, we have $\psi(n/2a) \le \psi(n/a)$.) □

We can now combine all the bounds derived so far, and bound (i) the expected number of prisms that are ever generated in the RIC, and (ii) the expected overall size of their conflict lists, which, as explained above, dominates the running time of the algorithm (with an additional logarithmic factor). The expected number of prisms is simply

$$\sum_{\tau\in\Pi} p_\tau = \sum_a\sum_b p(a,b)N_{a,b}. \tag{9}$$

Similarly, the expected overall size of the conflict lists is

$$\sum_{\tau\in\Pi} w_0(\tau)p_\tau = \sum_a\sum_b bp(a,b)N_{a,b}. \tag{10}$$

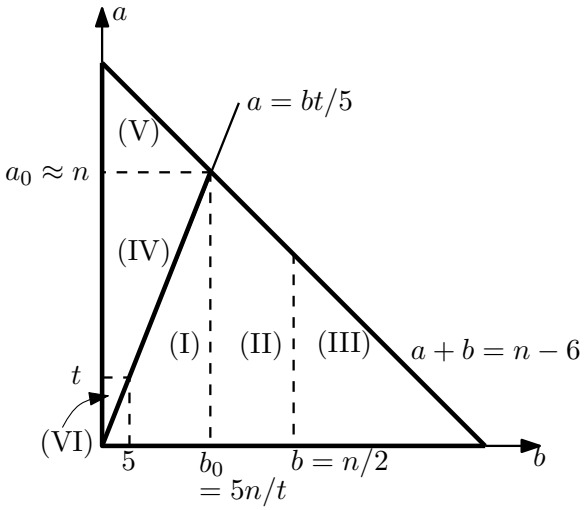

Figure 3: The decomposition of the (a, b)-range into subranges.

We bound these sums separately for pairs (a, b) within each of the six regions depicted in Figure 3. Together, these regions cover the entire range $a, b \geq 0$, $a + b \leq n - 6$. As will follow from the forthcoming analysis, the most expensive prisms are those for which (a, b) lies in region (I) or region (IV).

Region (I). In this region, $5 \leq b \leq 5n/t$ and $0 \leq a \leq bt/5$. We cover the region by vertical slabs of the form $S_j := \{(a, b) \mid b_{j-1} \leq b \leq b_j\}$, for $j \geq 1$, where $b_j = 5 \cdot 2^j$, for $j = 0, \ldots, \lceil \log(n/k) \rceil$. Within each slab S_j, the maximum value of p_τ is $O(1/b_{j-1}^6) = O(1/2^{6j})$, and we bound $\sum_{(a,b) \in S_j} N_{a,b}$ by $N_{\leq b_j t/5, \leq b_j}$ which, by Lemma 6.3, is

$$O\left(n(b_j t/5) b_j^4 \lambda_s(t/5)\right) = O\left(n b_j^5 t \lambda_s(t)\right) = O\left(2^{5j} n t \lambda_s(t)\right).$$

Hence, the contribution of S_j to (9) is at most

$$O\left(\frac{nt\lambda_s(t)}{2^j}\right),$$

and, summing this over j, we get that the contribution of region (I) to (9) is $O(nt\lambda_s(t))$. Similarly, the contribution of S_j to (10) is at most

$$O\left(b_j \cdot \frac{nt\lambda_s(t)}{2^j}\right) = O\left(nt\lambda_s(t)\right).$$

We need to multiply this bound by the number of slabs, which, as is easily checked, is $O(\log(n/t))$. Hence, the contribution of region (I) to (10) is $O(nt\lambda_s(t) \log(n/t))$.

Region (II). In this region, $5n/t \leq b \leq n/2$ and $0 \leq a \leq n - 6 - b$. Here too we cover the region by vertical slabs of the form $S'_j := \{(a, b) \mid b'_{j-1} \leq b \leq b'_j\}$, for $j \geq 1$, where $b'_j = (5n/t) \cdot 2^j$, for $j = 0, \ldots, \lceil \log(t/10) \rceil$. Within each S'_j, the maximum value of p_τ is $O(1/(b'_{j-1})^6) = O(t^6/(n^6 2^{6j}))$, and

we bound $\sum_{(a,b)\in S'_j} N_{a,b}$ by $N_{\le n-b'_{j-1}-6,\le b'_j}$ which, by Lemma 6.3, is (upper-bounding $n-b'_{j-1}-6$ by n)

$$O\left(n^2(b'_j)^4\lambda_s(n/b'_j)\right) = O\left(n^2(n/t)^4 2^{4j}\lambda_s(t/2^j)\right) = O\left(2^{3j}n^6\lambda_s(t)/t^4\right).$$

Hence, the contribution of S'_j to (9) is at most

$$O\left(t^2\lambda_s(t)/2^{3j}\right),$$

and, summing this over j, we get that the contribution of region (II) to (9) is $O(t^2\lambda_s(t)) = O(nt\lambda_s(t))$. Similarly, the contribution of S'_j to (10) is at most

$$O\left(b'_j t^2\lambda_s(t)/2^{3j}\right) = O\left(nt\lambda_s(t)/2^{2j}\right),$$

and, summing this over j, we get that the contribution of region (II) to (10) is $O(nt\lambda_s(t))$.

Region (III). In this region, $n/2 \le b \le n$ and $0 \le a \le n-6-b$. We treat this region as a single entity. The maximum value of p_τ here is $O(1/n^6)$, and we bound $\sum_{(a,b)\in(\mathrm{III})} N_{a,b}$ by the overall number of prisms, which is $O(n^6)$, getting a negligible contribution to (9) of only $O(1)$. A similar argument shows that the contribution of this region to (10) is $O(n)$, again negligible compared with the other regions.

Region (IV). In this region, $t \le a \le a_0 \approx n$ and $0 \le b \le 5a/t$. We cover the region by horizontal slabs of the form $S''_j := \{(a,b) \mid a_{j-1} \le a \le a_j\}$, for $j \ge 1$, where $a_j = t\cdot 2^j$, for $j = 0,\ldots,\lceil\log(a_0/t)\rceil = O(\log(n/t))$. Within each slab S''_j, the maximum value of p_τ is $O(t^6/a_{j-1}^6) = O(1/2^{6j})$, and we bound $\sum_{(a,b)\in S''_j} N_{a,b}$ by $N_{\le a_j,\le 5a_j/t}$ which, by Lemma 6.3, is

$$O\left(na_j(5a_j/t)^4\lambda_s(t/5)\right) = O\left(na_j^5\lambda_s(t)/t^4\right) = O\left(2^{5j}nt\lambda_s(t)\right).$$

Hence, the contribution of S''_j to (9) is at most

$$O\left(\frac{nt\lambda_s(t)}{2^j}\right),$$

and, summing this over j, we get that the contribution of region (IV) to (9) is $O(nt\lambda_s(t))$. Similarly, the contribution of S''_j to (10) is at most

$$O\left((5a_j/t)\frac{nt\lambda_s(t)}{2^j}\right) = O\left(nt\lambda_s(t)\right).$$

Since the number of slabs is $O(\log(n/t))$, region (IV) contributes $O(nt\lambda_s(t)\log(n/t))$ to (10).

Region (V). In this region, $a_0 \le a \le n$ and $0 \le b \le n-a-6$. We treat this region as a single entity. The maximum value of p_τ in this region is $O(t^6/n^6)$, and we bound $\sum_{(a,b)\in(\mathrm{V})} N_{a,b}$ by

$$N_{\le n,\le 5n/t} = O\left(n^2(5n/t)^4\lambda_s(t/5)\right) = O(n^6\lambda_s(t)/t^4).$$

Hence, the contribution of region (V) to (9) is at most

$$O\left(t^2\lambda_s(t)\right) = O(nt\lambda_s(t)).$$

For the contribution to (10), we multiply this bound by $O(n/t)$, an upper bound on b in this region, and get

$$O\left((n/t)\cdot t^2\lambda_s(t)\right) = O(nt\lambda_s(t)).$$

Region (VI). Finally, we consider this region, which is given by $0 \leq a \leq t$ and $0 \leq b \leq 5$. Here we upper-bound p_τ simply by 1, and bound $\sum_{(a,b)\in(\mathrm{VI})} N_{a,b}$ by $N_{\leq t,\leq 5}$, which is $O(nt\lambda_s(t))$. Hence, the contribution of region (VI) to (9) is at most $O(nt\lambda_s(t))$. Since b is bounded by a constant in this region, the same expression also bounds the contribution of region (VI) to (10).

In conclusion, taking the additional logarithmic factor into account, we have the following main result of this section.

Theorem 6.5. *The first t levels of an arrangement of the graphs of n continuous totally defined algebraic functions of constant description complexity, for which the complexity of the lower envelope of any m functions is $O(m)$, can be constructed by a randomized incremental algorithm, whose expected running time is $O(nt\lambda_s(t)\log(n/t)\log n)$, and whose expected storage is $O(nt\lambda_s(t))$.*

Remark. The result for superlinear lower envelope complexity is as follows.

Theorem 6.6. *The first t levels of an arrangement of the graphs of n continuous totally defined algebraic functions of constant description complexity, for which the complexity of the lower envelope of any m functions is at most $\psi(m)$, where $m \mapsto \psi(m)/m$ increases monotonically, can be constructed by a randomized incremental algorithm, with expected running time*

$$O(t^2\psi(n/t)\lambda_s(t)\log(n/t)\log n),$$

and expected storage

$$O(t^2\psi(n/t)\lambda_s(t)).$$

Proof. We again consider the six regions for (a, b), but with the bounds from Lemma 6.4. In Region (I), the bound on $N_{\leq b_j t/5,\leq b_j}$ within a slab S_j is $O(2^{6j}t^2\psi(n/(2^j t))\lambda_s(t))$, so the contriubtion of S_j is at most $O(t^2\psi(n/(2^j t))\lambda_s(t))$, resulting in a total contribution of $O(t^2\psi(n/t)\lambda_s(t))$ for Region (I) to the number of prisms, and $O(t^2\psi(n/t)\lambda_s(t)\log(n/t))$ to the conflict size. In Region (II), the bound on $N_{\leq n,\leq b'_j}$ within a slab S'_j is $O(\psi(1)\cdot n^6 2^{3j}\lambda_s(t)/t^4)$. Thus, the bound for Region (II) follows as before. The argument for Region (III) is also the same. In Region (IV), the bound on $N_{\leq a_j,\leq 5a_j/t}$ is $O(t^2 2^{6j}\psi(n/(t2^j))\lambda_s(t))$, so the contribution of the slab S''_j is $O(t^2\psi(n/(t2^j))\lambda_s(t))$. By our assumption on ψ, this sums to a total contribution of $O(t^2\psi(n/t)\lambda_s(t))$ to the number of prisms and of $O(t^2\psi(n/t)\lambda_s(t)\log(n/t))$ to the conflict size. In Region (V), the bound on $N_{\leq n,\leq 5n/t}$ is $O(\psi(1)n^6\lambda_s(t)/t^4)$, and the argument proceeds as before. Finally, in Region (VI), the bound on $N_{\leq t,\leq 5}$ is $O(t^2\psi(n/t)\lambda_s(t))$. The claim follows. □

7 Improved Dynamic Maintenance of Lower Envelopes for Planes

We present our interpretation of Chan's technique for dynamically maintaining the lower envelope of a set H of non-vertical planes in $\mathbb{R}^3$ under insertions and deletions. In the next section, we will combine this structure with the results from the previous sections to obtain a data structure that supports polylogarithmic update and query time for maintaining the lower envelope of general surfaces. As before, maintaining the lower envelope means that we can efficiently answer *point location* queries, each specifying a point $q \in \mathbb{R}^2$ and asking for the lowest plane above q.

Our exposition proceeds in three steps: We begin with a static structure, then develop a simple variant (with a not so simple analysis) of a standard technique for handling insertions, and finally

describe how to perform deletions. With the help of a simple charging argument in the analysis of the static structure, we also improve Chan's original (amortized) deletion time by a logarithmic factor, from $O(\log^6 n)$ to $O(\log^5 n)$, the first improvement in more than ten years. Subsequently to this work, Chan found an additional improvement, resulting in amortized deletion time $O(\log^4 n)$ and amortized insertion time $O(\log^2 n)$ [14]. This further improvement is based on the improvement presented in this section, combined with an improved algorithm for the data structure used in the static case.

7.1 A static structure

Let H be a fixed set of n non-vertical planes in $\mathbb{R}^3$. We fix a constant $k_0 \geq 1$, and define the sequence $k_j = 2^j k_0$, for $j = 0, 1, \ldots, m$, where $m = \lfloor \log(n/k_0) \rfloor$. We have $k_m \in (n/2, n]$.

Next, we construct a sequence $\{\Lambda_j\}_{j \geq 1}$ of vertical shallow cuttings, for $j = m+1, \ldots, 0$, as follows: the shallow cutting Λ_{m+1} consists of a single prism τ that covers all of $\mathbb{R}^3$ and has conflict list $\mathrm{CL}(\tau) = H$ (the shallowness is vacuously satisfied here). Next, we set $H_m = H$ and $n_m = n$, and we start with $j = m$. In round j, we have a set $H_j \subseteq H$ of $n_j \leq n$ *surviving* planes, and we construct a vertical k_j-shallow $(\alpha k_j/n_j)$-cutting Λ_j for $\mathcal{A}(H_j)$, for a suitable constant $\alpha > 1$ (the same constant for all rounds; see Section 3, and also [16, 29]). The reason for using this hierarchy of cuttings will become clear when we discuss deletions, in Section 7.3. That is, Λ_j consists of $O(n_j/k_j)$ semi-unbounded vertical prisms whose ceilings form the faces of a polyhedral terrain $\overline{\Lambda}_j$. The terrain $\overline{\Lambda}_j$ lies fully above level k_j, and the *conflict list* $\mathrm{CL}(\tau)$ of each prism $\tau \in \Lambda_j$, consisting of the planes from H_j that cross τ, is of size at most αk_j. After constructing Λ_j, we identify the set $H^\times \subseteq H_j$ of planes that belong to more than $c \log n$ conflict lists in *all* cuttings $\Lambda_m, \ldots, \Lambda_j$ constructed so far;[17] here c is some sufficiently large constant that we will specify later. We remove the planes in $H^\times$ from the conflict lists of Λ_j (but not from those of the higher levels $\Lambda_{j'}$, for $j' > j$), and we set $H_{j-1} = H_j \setminus H^\times$ and $n_{j-1} = |H_{j-1}|$ for the next round. This *pruning mechanism* ensures that each plane of H appears in at most $c \log n$ (pruned) conflict lists, within the current hierarchy of cuttings, which is crucial for the efficiency of the dynamic algorithm, to be presented in the next two subsections. Note that $\overline{\Lambda}_{j-1}$ is not necessarily "lower" than $\overline{\Lambda}_j$, since the former cutting is constructed with respect to a potentially smaller set of planes (and can therefore contain points that lie above $\overline{\Lambda}_j$, even though it approximates a lower-indexed level). We stop after performing the pruning step for Λ_0. The conflict lists of Λ_0 will be used for answering queries. (To support deletions, we will also need the conflict lists of Λ_j, for $j \geq 1$; see below.) We denote by $\mathcal{D}^{(1)}$ the structure consisting of the shallow cuttings $\Lambda_{m+1}, \Lambda_m, \ldots, \Lambda_0$ and the pruned conflict lists of their prisms. We write $S(\mathcal{D}^{(1)})$ for the set of surviving planes in $\mathcal{D}^{(1)}$ (those that were not pruned at any stage), and $C(\mathcal{D}^{(1)}) = H$ for the initial set of planes. We say that $S(\mathcal{D}^{(1)})$ is *stored* in $\mathcal{D}^{(1)}$ and that $C(\mathcal{D}^{(1)})$ was used to *construct* $\mathcal{D}^{(1)}$. The planes in $C(\mathcal{D}^{(1)}) \setminus S(\mathcal{D}^{(1)})$ are the *pruned* planes in $\mathcal{D}^{(1)}$. We emphasize again that in general a pruned plane still shows up in some conflict lists of some of the cuttings Λ_j (for indices j higher than the one at which it was pruned). The difference is that the stored planes are kept only in $\mathcal{D}^{(1)}$, whereas the pruned planes are also passed to subsequent substructures, as we will soon describe. The following lemma bounds the size of $S(\mathcal{D}^{(1)})$.

[17] We note, for the expert reader, that this is the point where our construction improves over Chan's original result, since Chan's pruning strategy considers each cutting individually. Our approach ensures that each plane appears in $O(\log n)$ conflict lists in a substructure, whereas in Chan's structure the bound is $O(\log^2 n)$. Lemma 7.1 shows that the more aggressive pruning does not remove too many planes.

Lemma 7.1. *For any $\zeta \in (0,1)$ there exists a sufficiently large (but constant) choice of c (the coefficient in the threshold $c \log n$, beyond which we prune planes), so that $|S(\mathcal{D}^{(1)})| \geq (1-\zeta)n$.*

Proof. Define the potential $\Phi(j)$ as the total "stored size" of the conflict lists of $\Lambda_m, \ldots, \Lambda_j$, after the pruning step in round j, and set $\Phi(m+1) = 0$. By the stored size of a conflict list $\mathrm{CL}(\tau)$, of some prism τ of some cutting, we mean the number of planes in $\mathrm{CL}(\tau)$ that have not been pruned yet, at any of the steps $m, m-1, \ldots, j$.[18] Since Λ_j has $O(n_j/k_j)$ prisms, and each conflict list of Λ_j has $O(k_j)$ planes, the overall size of the conflict lists of Λ_j is $O(n_j) = O(n)$. Hence, in round j, we increase Φ by at most γn, for some fixed $\gamma > 0$, where the increase is caused by the conflict lists of the prisms of the new cutting. If a plane h is pruned at this stage, then h lies in at least $c \log n$ conflict lists of all stages processed so far (including those at the present stage), and has to be removed from the stored size count of these prisms, so this decreases Φ by at least $c \log n$. Since Φ is initially 0 and is never negative, and since we increase it by at most $\gamma n \log n$ units throughout the construction, it follows that we prune at most ζn planes, if we choose $c \geq \gamma/\zeta$. □

We fix the fraction $\zeta = 1/32$, and use the corresponding coefficient c in the construction. We set $H^{(1)} = H$ and $H^{(2)} = C(\mathcal{D}^{(1)}) \setminus S(\mathcal{D}^{(1)}) = H \setminus S(\mathcal{D}^{(1)})$ (that is, $H^{(2)}$ consists of the planes that we have pruned). As just argued, we have $|H^{(2)}| \leq \zeta n$. We repeat the process with the set $H^{(2)}$, obtaining an analogous structure $\mathcal{D}^{(2)}$ and a remainder set $H^{(3)} = C(\mathcal{D}^{(2)}) \setminus S(\mathcal{D}^{(2)}) = H^{(2)} \setminus S(\mathcal{D}^{(2)})$ of at most $\zeta^2 n$ planes that were pruned in $\mathcal{D}^{(2)}$. Proceeding in this manner, for $s \leq \log_{1/\zeta} n = \frac{1}{5} \log n$ steps, we obtain the complete target (static) structure $\mathcal{D} = (\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \ldots, \mathcal{D}^{(s)})$, where $\mathcal{D}^{(s)}$ involves only a constant number of planes. Thus, in the last step, the overall size of the conflict lists is (much) smaller than $c \log n$, so all of these planes are stored, and the process can 'safely' terminate. By construction, the sets $S(\mathcal{D}^{(i)})$ are pairwise disjoint and their union is H. For each i, let $m_i = \lfloor \log(|H^{(i)}|/k_0) \rfloor = O(\log n)$ be the number of cuttings in $\mathcal{D}^{(i)}$. The overall size of $\mathcal{D}^{(i)}$, including the conflict lists, is

$$O\left(\sum_{j=0}^{m_i} \frac{|H^{(i)}|}{k_j} \cdot k_j\right) = O\left(|H^{(i)}| m_i\right) = O\left(|H^{(i)}| \log n\right).$$

Since by Lemma 7.1, we have $|H^{(i)}| \leq \zeta^{i-1} n$, the overall size of $\mathcal{D}$ is $O(n \log n)$.[19] Using the algorithm of Chan and Tsakalidis [16], we construct each cutting Λ_j, including its associated conflict lists, in any $\mathcal{D}^{(i)}$, in $O(|H^{(i)}| \log n)$ time. Summing over j, we can construct $\mathcal{D}^{(i)}$ in $O(|H^{(i)}| \log^2 n)$ time, and summing over i, recalling that $|H^{(i)}|$ decreases geometrically with i, we get a total running time $O(n \log^2 n)$.[20] We write this bound as $an \log^2 n$, for a suitable concrete coefficient a.

Answering a query is easy: Given $q \in \mathbb{R}^2$, we iterate over all $O(\log n)$ substructures $\mathcal{D}^{(i)}$, and we find the prism τ of the cutting Λ_0 in $\mathcal{D}^{(i)}$ whose xy-projection contains q (or possibly the $O(1)$ prisms, if the query q is not in general position and falls on the boundary of several such prisms). This takes $O(\log n)$ time, with a suitable point-location structure for the xy-projection of $\overline{\Lambda}_0$. We then search the conflict list $\mathrm{CL}(\tau)$ of τ, in brute force, for the lowest plane over q (or possibly

[18] Some of these planes might be pruned, later, from the conflict lists of some lower-indexed cuttings, and then they will no longer be counted in the stored size of τ.

[19] Note that if we did not have to account for the conflict lists, the sum would have been $O(n)$. Later, we will reduce the storage, including the conflict lists, to linear, by representing the conflict lists implicitly.

[20] We note, again, that the recent improvement of Chan [14] comes from decreasing the running time of this preprocessing step to $O(n \log n)$, while the rest of the algorithm remains essentially unchanged.

planes, in case that q is not in general position). This requires $O(k_0) = O(1)$ time. We return the plane (or planes) lowest over q among all $O(\log n)$ candidates. The query time is thus $O(\log^2 n)$. The correctness of this procedure is obvious, and follows from the property that the ceilings of the prisms in Λ_0 (for any $\mathcal{D}_i$) pass above level $k_0 \geq 1$ of $\mathcal{A}(H^{(i)})$, so the lowest plane of $H^{(i)}$ over q belongs to the conflict list of the corresponding prism.

7.2 Handling insertions

Here, we explain how to maintain the lower envelope of a set H of non-vertical planes in $\mathbb{R}^3$ under insertions, where the notion of maintenance is as defined in the preceding section. For simplicity, at each point in time, we denote the current number of planes in the data structure by n. Furthermore, we use N to denote the power of 2 satisfying $n \in [N, 2N)$. Whenever n becomes too large, we double the value of N. Our structure uses a variant of a standard technique, introduced by Bentley and Saxe [7] and later refined in Overmars and van Leeuwen [43] (see also Erickson's notes [26]). Specifically, we maintain a sequence $\mathcal{I} = (\mathcal{B}_{i_0}, \mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_k})$ of structures, where $0 \leq i_0 < i_1 < \cdots < i_k$. The indices i_j are not fixed, are not necessarily contiguous, and may change after each insertion. Informally, we have an infinite sequence of bins, indexed by $0, 1, 2, \ldots$, and i_j is the index of the bin that stores $\mathcal{B}_{i_j}$, for $j = 0, 1, \ldots, k$. We refer to $\mathcal{B}_i$ as the structure at *location* (or bin) i. Lemma 7.2 below shows that i_k, and thus also k, are $O(\log n)$. Each $\mathcal{B}_{i_j}$ is a *substructure* $\mathcal{D}^{(u)}$ of some static structure $\mathcal{D}$, as in Section 7.1, constructed over some subset of H. We maintain the following invariants.

(I1) For each occupied index i, we have $2^{i-1} < |S(\mathcal{B}_i)| \leq 2^i$.

(I2) The sets $S(\mathcal{B}_i)$, over the occupied indices i, are pairwise disjoint, and their union is H.

For each plane $h \in H$, we say that h *is stored* at location i if $h \in S(\mathcal{B}_i)$.

Inserting a plane. To insert a plane h, we determine the smallest index of an empty bin, i.e., the smallest integer $j \geq 0$ that is not in the sequence $(i_0, i_1, \ldots, i_k)$. If $j = 0$, we store h in a trivial structure $\mathcal{B}_0$ of size 1. Otherwise, we have $(i_0, \ldots, i_{j-1}) = (0, \ldots, j-1)$, and we set $H_j := \left(\bigcup_{i=0}^{j-1} S(\mathcal{B}_i)\right) \cup \{h\}$. Assuming inductively that Invariants (I1) and (I2) hold prior to the insertion of h, we have

$$|H_j| = 1 + \sum_{i=0}^{j-1} |S(\mathcal{B}_i)| > 1 + \sum_{i=0}^{j-1} 2^{i-1} > 2^{j-1}, \tag{11}$$

and

$$|H_j| = 1 + \sum_{i=0}^{j-1} |S(\mathcal{B}_i)| \leq 1 + \sum_{i=0}^{j-1} 2^i = 2^j. \tag{12}$$

We construct over H_j a static structure $\mathcal{D}$, as in Section 7.1. Recall that $\mathcal{D}$ is a sequence of a logarithmic number of substructures, $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \ldots, \mathcal{D}^{(s)}$, where (recall Lemma 7.1)

$$s \leq \log_{1/\zeta} |H_j| = \frac{\log |H_j|}{\log(1/\zeta)} \leq \frac{j}{\log(1/\zeta)} = \frac{j}{5} < j,$$

by (12) and our choice of $\zeta = 1/32$. We remove the current structures $\mathcal{B}_0, \ldots, \mathcal{B}_{j-1}$ from $\mathcal{I}$. Then, for each substructure $\mathcal{D}^{(u)}$ constructed for H_j, for $u = 1, \ldots, s$, we set $\mathcal{B}_{i_u} := \mathcal{D}^{(u)}$ for $i_u = \lceil \log |S(\mathcal{D}^{(u)})| \rceil$, and add $\mathcal{B}_{i_u}$ to $\mathcal{I}$.

We have $C(\mathcal{D}^{(1)}) = H_j$. By Lemma 7.1 and (11), $|S(\mathcal{D}^{(1)})| \geq (1-\zeta)|H_j| \geq (1-\zeta)2^{j-1} > 2^{j-2}$ (recall that $\zeta = 1/32$), so it follows that $\mathcal{D}^{(1)}$ is placed in bin j or $j-1$. Moreover, by Lemma 7.1, we have $|S(\mathcal{D}^{(u+1)})| \leq \zeta |C(\mathcal{D}^{(u)})| \leq \frac{\zeta}{1-\zeta}|S(\mathcal{D}^{(u)})|$, so the corresponding indices satisfy

$$\begin{aligned} i_{u+1} &= \lceil \log |S(\mathcal{D}^{(u+1)})| \rceil \leq 1 + \log |S(\mathcal{D}^{(u+1)})| \\ &\leq 1 + \log\Big(\frac{\zeta}{1-\zeta}|S(\mathcal{D}^{(u)})|\Big) = 1 + \log |S(\mathcal{D}^{(u)})| - \log 31 \\ &= \log |S(\mathcal{D}^{(u)})| - \log 15.5 \leq \lceil \log |S(\mathcal{D}^{(u)})| \rceil - \log 15.5 = i_u - \log 15.5, \end{aligned}$$

since $\zeta = 1/32$. That is, since both i_u and i_{u+1} are integers, $i_{u+1} \leq i_u - 4$. Hence, each structure $\mathcal{D}^{(u)}$ is assigned a different index $i \leq j$ (with a gap of at least three empty bins between consecutive occupied ones), and Invariants (I1) and (I2) hold by construction ((I1) follows from the definition of the indices i_u).

Answering a query. To answer a query q, we find in each substructure $\mathcal{B}_i$ of $\mathcal{I}$ the prism (or prisms, in case of a non-generic query) of the corresponding lowest-index cutting Λ_0 whose xy-projection contains the query point q, and we search over the at most k_0 planes of its conflict list for the lowest ones over q. We output the lowest among all these planes, over all substructures. This takes $O(\log n)$ time per structure, as in the static case.

The correctness of this procedure follows from Invariant (I2). Indeed, if h is a lowest plane over a query point q, then $h \in S(\mathcal{D}^{(i)})$ for some unique i (by Invariant (I2)). Since h is stored at $\mathcal{D}^{(i)}$, it has not been pruned from any conflict list of this substructure. Let τ be a prism of the lowest-indexed cutting Λ_0 of $\mathcal{D}^{(i)}$ whose xy-projection contains q. By construction, the ceiling of τ lies above the k_0-level of the corresponding arrangement, which implies that h must lie in τ over q. This, and the fact that h has not been pruned, implies that h belongs to the (pruned) conflict list $\mathrm{CL}(\tau)$, so the query will encounter h, and thus will output it as the correct answer. The following lemma bounds the size of $\mathcal{I}$.

Lemma 7.2. *The largest index of an occupied bin is at most* $\log N + 1$*, and thus the number of structures in* $\mathcal{I}$ *is at most* $\log N + 2$*.*

Proof. By definition, the number of planes in the structure satisfies $n < 2N$, and by Invariant (I1), the largest index $i \in \mathcal{I}$ satisfies

$$\begin{aligned} 2^{i-1} < |S(\mathcal{B}_i)| \leq n < 2N, \\ \text{or} \quad i < 1 + \log(2N) = \log N + 2, \end{aligned}$$

that is $i \leq \log N + 1$ (since $\log N$ is an integer). □

Running Time. We now bound the running time of our insertion-only structure. In particular, we are going to show that the deterministic amortized cost of an insertion is $O(\log^3 n)$ and the deterministic worst-case cost of a query is still $O(\log^2 n)$, as in the static case.

Indeed, the bound on the query time is immediate by Lemma 7.2 and the observation preceding it, because $\log N = O(\log n)$. To analyze the amortized insertion cost, we use a charging argument.

Each plane h that is currently stored in $\mathcal{I}$ holds $b(w-i)$ *credits*, each worth $a\log^2 N$ *units*, where i is the current unique location (bin index) of the structure $\mathcal{B}_i$ for which $h \in S(\mathcal{B}_i)$, and $w := \log N + 2$ bounds, by Lemma 7.2, the maximum length of $\mathcal{I}$. Here a is the coefficient of the bound $an\log^2 n$ for the construction time of the static structure on n planes (see Section 7.1), and b is some absolute constant to be fixed shortly. Note that the number of credits held by a plane h is larger when the bin index where h is stored is smaller.

We define the *potential* Ψ of the structure as the overall number of units of credit that its planes hold. The amortized cost of an insertion is defined to be bw credits, that is, $abw\log^2 N$ units.

When a new plane is inserted, we give it bw credits, that is, $abw\log^2 N$ units of credit. The unit size depends on N, so the whole charging scheme has to be updated every time N is doubled. Specifically, when N is doubled, the increase in the size of a credit is

$$a\log^2(2N) - a\log^2 N = a(1+\log N)^2 - a\log^2 N = a(1+2\log N).$$

We have $2N$ planes in the structure at this moment, and each of them carries at most bw credits. Hence, updating the overall amount of credit in the structure in this doubling costs $O(Nw\log N)$ units, which is $O(N\log^2 N)$. There are only $O(\log n)$ doubling steps, and the N's that they involve are powers of 2, implying that the overall additional cost of updating the credit distribution during doublings of N is $O(n\log^2 n)$. In what follows we ignore this issue of updating the credit size, whose cost will be subsumed by the overall cost of the insertions, and just use the number of credits in our charging scheme.[21]

We recall that, when inserting a new plane h, we destroy a prefix of j substructures in $\mathcal{I}$, put all their planes, including h, in a subset H_j, compute a new static structure $\mathcal{D}$ for H_j, and spread its substructures $\mathcal{D}^{(u)}$ into some subset of bins with indices from j downwards. Set $t = |H_j|$. As noted above, by the analysis in Section 7.1, the real cost of the insertion is at most $at\log^2 t$, or, in other words, at most t credits. The main claim in the complexity analysis of insertions is the following lemma.

Lemma 7.3. *With the above notations, for a sufficiently large choice of the constant b, we have*

$$at\log^2 t + \Delta\Psi \le abw\log^2 N, \tag{13}$$

where $\Delta\Psi = \Psi^+ - \Psi^-$, where Ψ^+ and Ψ^- are, respectively, the values of the potential just after and just before the insertion.

Proof. Consider the sequence $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \ldots, \mathcal{D}^{(s)}$, of substructures that we construct over H_j, where $s = \lfloor \log_{1/\zeta} |H_j| \rfloor$. Recall that by (11) and (12), we have that $2^{j-1} < t = |H_j| \le 2^j$. In particular, $s \le \frac{1}{5}\log t \le j/5 < j$. As already noted, by Lemma 7.1, $|S(\mathcal{D}^{(1)})| \ge (1-\zeta)t \ge (1-\zeta)2^{j-1}$, and consequently (since $\zeta = 1/32$), the structure $\mathcal{D}^{(1)}$ is placed either in bin j or bin $j-1$. Furthermore, the lower bound in Invariant (I1) shows that before the insertion, at least $\sum_{i=1}^{j-2} 2^{i-1}+1 > 2^{j-2} \ge t/4$ planes of H_j were stored at bins $i = 0, 1, \ldots, j-2$ (the $+1$ is for bin 0, which contains exactly one plane). Since $|S(\mathcal{D}^{(1)})| \ge (1-\zeta)t$, it follows that at least $t/4 - \zeta t$ among the planes that were stored in bins $i = 0, 1, \ldots, j-2$, end up at $\mathcal{B}_j$ or $\mathcal{B}_{j-1}$ following the insertion. These planes release at least $bt(1/4-\zeta)$ credits, that is, they decrease Ψ by at least these many credits.

[21] Again, Chan's improvement [14] is reflected in this scheme by reducing the size of a credit to $O(\log N)$, leaving the rest of the analysis unchanged.

The technical issue that we need to address is that some planes in $\mathcal{D}^{(2)}, \ldots, \mathcal{D}^{(s)}$ may require more credits than what they had before the insertion, if they end up in smaller-indexed bins than the bins in which they were stored before the insertion. We claim that the overall amount of this extra credit is much smaller than the amount of released credit, so the released credit (more than) suffices to fill in the required extra credit, thereby making each plane hold the correct amount of credit, with change to spare. That is, the insertion does cause Ψ to decrease.

To show this, let $(j-1)-j_i$ be the bin index in which we put $\mathcal{D}^{(i)}$, for $2 \le i \le s$. In the worst case, each plane of $\mathcal{D}^{(i)}$ was stored in bin $j-1$ before the insertion and now requires bj_i additional credits. (Note that h does not participate in this argument: it cannot release any credit since it did not exist before the insertion; nevertheless, the credits that it gets as it is being inserted more than suffice for any bin h is eventually placed in.) Summing up, we get that the number of additional credits that we need to give the planes is at most $b\sum_{i=2}^{s} j_i |S(\mathcal{D}^{(i)})|$. From the definition of the insertion mechanism, we get that $(j-1)-j_i = \lceil \log |S(\mathcal{D}^{(i)})| \rceil$, so the total number of additional credits that we need to give these planes is at most

$$b\sum_{i=2}^{s} j_i |S(\mathcal{D}^{(i)})| = b\sum_{i=2}^{s} \Big(j-1-\lceil \log |S(\mathcal{D}^{(i)})| \rceil\Big)|S(\mathcal{D}^{(i)})| \le b\sum_{i=2}^{s} \Big(j-1-\log |S(\mathcal{D}^{(i)})|\Big)|S(\mathcal{D}^{(i)})|. \quad (14)$$

Since the function $x \mapsto (j-1-\log x)x$ is increasing for $0 < x \le 2^{j-1-1/\ln 2} \approx 2^{j-2.442}$, and $|S(\mathcal{D}^{(i)})| \le \zeta^{i-1}t < 2^{j-3}$ for every $i = 2, \ldots, s$ and for $\zeta = 1/32$, we can bound the last expression in (14) by replacing $|S(\mathcal{D}^{(i)})|$ with $\zeta^{i-1}t$, for every $i = 2, \ldots, s$, so we get that

$$b\sum_{i=2}^{s} \Big(j-1-\log |S(\mathcal{D}^{(i)})|\Big)|S(\mathcal{D}^{(i)})| \le b\sum_{i=2}^{s} \Big(j-1-\log(\zeta^{i-1}t)\Big)\zeta^{i-1}t.$$

Since $t \ge 2^{j-1}$, we can bound the right-hand side by

$$-bt\sum_{i=2}^{s} \log(\zeta^{i-1})\zeta^{i-1} = bt \log\frac{1}{\zeta}\sum_{i=2}^{s}(i-1)\zeta^{i-1} \le 5bt\sum_{i=1}^{\infty} i\zeta^i = 5bt\frac{\zeta}{(1-\zeta)^2}.$$

We conclude that, for $\zeta = \frac{1}{32}$, the sum is smaller than $bt/6$, so we still have more than

$$bt\left(\frac{1}{4}-\frac{1}{32}\right)-\frac{bt}{6} = \frac{5bt}{96}$$

free credits. By construction, this free credit is $\Psi^- - \Psi^+ = -\Delta\Psi$, to which we add the bw credits we gave h upon insertion. Hence, if b is large enough, say at least 20, we have $bw - \Delta\Psi \ge t$. Since the real insertion cost is at most t credits, it follows that the released credit suffices to pay for the real insertion cost (with change to spare), and the lemma follows. □

Corollary 7.4. *The overall cost of n insertions is $O(n \log^3 n)$.*

Proof. As already mentioned, we ignore the changes in the size of the credits caused by doublings of N; as noted, this adds only $O(n \log^2 n)$ to the overall cost. We add up the inequalities (13), over all insertions, and get that the overall actual cost of the insertions plus $\Psi_{\text{final}} - \Psi_{\text{initial}}$, is at most $O(nw \log^2 n) = O(n \log^3 n)$. Since $\Psi_{\text{final}} - \Psi_{\text{initial}} \ge 0$, this also bounds the actual cost of n insertions. □

We note that Chan's recent improvement [14] follows from this corollary, simply be reducing the size of a single credit to $O(\log N)$, and leaving the rest of the analysis intact. The following lemma summarizes the properties of our insertion-only structure.

Lemma 7.5. *The deterministic amortized cost of an insertion is $O(\log^3 n)$, and the deterministic worst-case cost of a query is $O(\log^2 n)$.*

7.3 Handling deletions

Finally, we describe how to maintain the lower envelope of a set H of non-vertical planes in $\mathbb{R}^3$ under insertions and deletions. As before, we denote the current number of planes in the data structure by n, and we use N to denote the power of 2 with $n \in [N, 2N)$. Now we add a *global rebuilding* mechanism: whenever the number of updates (insertions and deletions) since the last global rebuild becomes $N/2$, we completely rebuild the data structure from scratch for the current set H, and we adjust (double or half) the value of N, if needed, to restore the size range invariant.[22] We will argue later that the overall cost of the rebuildings is subsumed in (and actually much smaller than) the cost of the other steps of the algorithm.

The basic organization of the data structure is the same as in Section 7.2, consisting of a sequence of bins $\mathcal{I} = (\mathcal{B}_{i_0}, \mathcal{B}_{i_1}, \ldots, \mathcal{B}_{i_k})$, where $0 \le i_0 < i_1 < \cdots < i_k$, occupied by substructures of some static structures. For each such substructure, we continue to denote by $C(\mathcal{B}_j)$ the set of planes that it was constructed from, and by $S(\mathcal{B}_j) \subseteq C(\mathcal{B}_j)$ the subset of planes that survived (are stored) in it.[23]

Insertions and queries are performed in much the same way as in Section 7.2, although some aspects of their implementation and analysis are different; see below for details. We delete a plane h by visiting each substructure $\mathcal{B}_j$ with $h \in C(\mathcal{B}_j)$, and marking h as *deleted* in each conflict list of $\mathcal{B}_j$ that contains h (note that this is done also for substructures in which h has not survived the initial construction, because it was pruned at some level of the hierarchy). Each plane can get marked, at the time it is deleted, up to $O(\log^2 n)$ times, once for each conflict list that contains it at the time when the deletion takes place. (Recall that, at each $\mathcal{B}_j$ with $h \in C(\mathcal{B}_j)$, h appears in at most $O(\log n)$ (pruned) conflict lists, over the entire hierarchy constructed at $\mathcal{B}_j$.)

Actual removal of h, albeit possibly only from some of the substructures, takes place during a global rebuild or when pieces of the data structure are rebuilt, either during an insertion of a new plane, or at certain steps of the procedure where conflict lists are purged and their elements are reinserted; see below for details.

For a substructure $\mathcal{B}_i$, we denote by $A(\mathcal{B}_i) \subseteq S(\mathcal{B}_i)$ the set of *active* planes in $\mathcal{B}_i$, defined as those planes that (a) are in $S(\mathcal{B}_i)$, (b) have not been (marked as) deleted, and (c) have not been reinserted into other substructures due to the lookahead deletion mechanism, which we describe next. When a substructure $\mathcal{B}_i$ is created, we have $A(\mathcal{B}_i) = S(\mathcal{B}_i)$. Once $\mathcal{B}_i$ is created, its associated sets $C(\mathcal{B}_i)$ and $S(\mathcal{B}_i)$, as well as all non-purged conflict lists $\mathrm{CL}(\tau)$ of prisms τ in (any hierarchical stage within) $\mathcal{B}_i$, remain fixed, until $\mathcal{B}_i$ is destroyed (in a rebuild triggered by an insertion or a reinsertion, or in a global rebuild). On the other hand, conflict lists may be marked as purged by the deletion-lookahead mechanism, and the set $A(\mathcal{B}_i)$ of active planes in $\mathcal{B}_i$ may get smaller, due to deletions and the purging of conflict lists.

[22] Note that the actual value of n does not have to change much, when the sequence of insertions and deletions is reasonably balanced.

[23] We remind the reader that $\mathcal{B}_j$ may also contain non-stored, i.e., pruned planes.

Lookahead deletions. When too many planes in a conflict list $\mathrm{CL}(\tau)$, for some prism τ, are (marked as) deleted, the real lower envelope of H might rise too high, and the lowest (undeleted) plane over a query point q (with q lying in the projection of τ) does no longer have to belong to $\mathrm{CL}(\tau)$. (Note that if τ belongs to the lowest-indexed cutting of some structure $\mathcal{B}_j$, which is the only kind of prisms we access when processing a query, its ceiling lies above level k_0 of the corresponding set of planes, so, even after $k_0 - 1$ deletions from that set, the lowest plane over q still belongs to $\mathrm{CL}(\tau)$, but a larger number of deletions may cause the difficulty just noted to arise.)

To avoid this situation, which might cause us to miss the correct answer to a query, we use the following *lookahead deletion mechanism*. Suppose that, for some prism τ in a cutting Λ_i of some substructure $\mathcal{B}_j$, at least $|\mathrm{CL}(\tau)|/(2\alpha)$ planes in $\mathrm{CL}(\tau)$ have been marked as deleted,[24] where $\alpha > 1$ is our cutting parameter (i.e., each prism of Λ_i is intersected by at most $\alpha k_i = 2^i \alpha k_0$ planes, for any i). Then we *purge* the conflict list $\mathrm{CL}(\tau)$, and we reinsert (only) the planes in $\mathrm{CL}(\tau) \cap A(\mathcal{B}_i)$, one by one, using the standard insertion algorithm. After this, $A(\mathcal{B}_i)$ contains no more elements from $\mathrm{CL}(\tau)$, but elements from $\mathrm{CL}(\tau)$ may still appear in other conflict lists of $\mathcal{B}_i$ and in $S(\mathcal{B}_i)$. We keep the purged prism τ in $\mathcal{B}_i$; whenever a query accesses τ (when τ is a prism of the lowest-indexed cutting of some substructure), it realizes that $\mathrm{CL}(\tau)$ is purged and simply skips it. A plane h may be reinserted many times due to the purging of a conflict list, but only once for each substructure $\mathcal{B}_i$ in which it is active (prior to the operation). Also, the planes in $C(\mathcal{B}_i) \setminus S(\mathcal{B}_i)$, and the planes marked as deleted will never be reinserted when a conflict list is purged.

As mentioned, queries and insertions are handled as in Section 7.2. For queries, while processing a substructure $\mathcal{B}_i$ and searching in the conflict list of some prism τ of the lowest-indexed cutting Λ_0 of $\mathcal{B}_i$, we only consider planes in $\mathrm{CL}(\tau) \cap A(\mathcal{B}_i)$, and report the lowest among them over the query point. As we will show later, in Lemma 7.6, this suffices to retrieve the correct overall lowest plane. That is, the plane that is lowest over q among the reported planes is the overall lowest plane over q.

To insert a plane h, we take, as in Section 7.2, the largest contiguous prefix $\mathcal{I}'$ of occupied bins in $\mathcal{I}$, of some length j, discard the existing structures in $\mathcal{I}'$, set[25] $H_j := \left(\bigcup_{i=0}^{j-1} A(\mathcal{B}_i)\right) \cup \{h\}$, construct a new static structure for H_j, and spread its substructures within some bins of $\mathcal{I}'$, according to the rules of Section 7.2. A plane $h \in H_j$ is active after the insertion (only) at the bin where it is stored.

Since we perform the reconstruction of the structure only on the active planes in the various destroyed substructures, the planes marked as deleted really disappear at this step, but only from the structures stored at the bins of $\mathcal{I}'$; such a plane might still show up (marked as deleted) in substructures $\mathcal{B}_\ell$ with larger bin indices ℓ, which have not been touched by this instance of the insertion procedure.

It is easy to prove, by induction on the number of operations, that the following invariants are maintained:

(D1) For each i, we have $2^{i-1} < |S(\mathcal{B}_i)| \leq 2^i$.

(D2) The sets $A(\mathcal{B}_i) \subseteq S(\mathcal{B}_i)$ are pairwise disjoint, and their union is H.

Indeed, Invariant (D1) is the same as Invariant (I1), and its maintenance is argued as in Section 7.2. Invariant (D2) follows because, by induction, a plane h is active, before the current operation, at

[24] Note that $\mathrm{CL}(\tau)$ can also contain planes from $C(\mathcal{B}_i) \setminus S(\mathcal{B}_i)$, and that these planes also count for this test.

[25] Note the difference between this step and the insertion in the insert-only case, discussed in Section 7.2, where H_j includes all the planes in the sets $S(\mathcal{B}_i)$. In contrast, here only the active planes are considered.

exactly one bin. If we delete h then the invariant continues to hold, as h no longer belongs to H. If we purge h from a conflict list in some substructure $\mathcal{B}_i$, it is no longer active in $\mathcal{B}_i$, but its reinsertion makes it active again, at the unique bin where it is stored. The same reasoning applies to all planes that were active at the bins that were destroyed by the reinsertion, and a similar reasoning holds when we insert (rather than reinsert) a plane.

Note though that the lower bound (11) of Section 7.2 does not have to hold now, as the number of active planes may be much smaller that the number of stored planes. The upper bound (12) remains valid, and so does Lemma 7.2. The correctness of the data structure is a consequence of the following lemma.

Lemma 7.6. *Let $q \in \mathbb{R}^2$, and let $h \in H$ be a (non-deleted) plane on the lower envelope of H over q (for most queries, h is unique). Let $\mathcal{B}_i$ be the unique substructure for which $h \in A(\mathcal{B}_i)$. Then h belongs to the conflict list $\mathrm{CL}(\tau)$ of the prism τ of the lowest-indexed cutting Λ_0 of $\mathcal{B}_i$, whose xy-projection contains q, and τ has not been marked as purged.*

Proof. The second part of the claim is an immediate consequence of the first part, since h is active in $\mathcal{B}_i$.

Assume to the contrary that $h \notin \mathrm{CL}(\tau)$. Let q^+ be the point on h over q. By assumption, q^+ lies on the lower envelope of H, and since $h \notin \mathrm{CL}(\tau)$, but $h \in A(\mathcal{B}_i) \subseteq S(\mathcal{B}_i)$, we have that h does not cross τ, so the point q^+ lies above τ. Let t be the largest index for which q^+ lies above the top terrain $\overline{\Lambda}_t$ of the cutting Λ_t of $\mathcal{B}_i$; by what we have just argued, such a t exists, and it is possible that $t = m$ (the total number of real terrains in $\mathcal{B}_i$), but $t \neq m+1$. Let τ' be the prism of Λ_t for which q^+ lies above the ceiling $\overline{\tau}'$, or, equivalently, q lies in the xy-projection of τ'.[26] Since Λ_t is a shallow cutting of $L_{\leq k_t}(H_t)$, for a suitable set of planes H_t, we have that $\overline{\Lambda}_t$ lies fully above $L_{k_t}(H_t)$. Thus, at least k_t planes of H_t pass below q^+, and we denote the set of these planes as $\mathrm{CL}(q^+)$. Since q^+ now lies on the lower envelope of H, all the (at least k_t) planes of $\mathrm{CL}(q^+)$ must have been (marked as) deleted from H.

Note that it is possible that $\mathrm{CL}(\tau')$ has already been purged by the lookahead deletion mechanism. Note also that we do not necessarily have $\mathrm{CL}(q^+) \subseteq \mathrm{CL}(\tau')$, as some planes from H_t may have appeared in too many conflict lists after the construction of Λ_t and may have been removed by the pruning mechanism from the conflict list of Λ_t.

Consider now the prism τ'' of Λ_{t+1} that contains q^+ (with $\tau'' = \mathbb{R}^3$, if $t = m$). Since $\mathrm{CL}(q^+) \subseteq H_t$, none of its planes were pruned earlier, before constructing Λ_t. Consequently, $\mathrm{CL}(q^+) \subseteq \mathrm{CL}(\tau'')$ and, by definition of q^+, $h \in \mathrm{CL}(\tau'')$.[27] In the extreme case $t = m$ we take τ'', as just mentioned, to be the entire 3-space, and then $H_t \subseteq \mathrm{CL}(\tau'')$. If $t < m$, we have $|\mathrm{CL}(\tau'')| \leq \alpha k_{t+1} = 2\alpha k_t$, and if $t = m$, we have $|\mathrm{CL}(\tau'')| = n \leq 2\alpha k_m$. Hence, by the time q^+ has reached the lower envelope of H, at least

$$|\mathrm{CL}(q^+)| \geq k_t \geq \frac{|\mathrm{CL}(\tau'')|}{2\alpha}$$

planes of $\mathrm{CL}(\tau'')$ have been marked as deleted. Thus, the lookahead-deletion mechanism should have purged $\mathrm{CL}(\tau'')$, which contains h, but h is still in $A(\mathcal{B}_i)$, a contradiction that establishes the claim. □

[26] We already noted, in Section 7.1, that the terrains $\overline{\Lambda}_k$ are not necessarily monotone increasing in z. Nevertheless, the definition of t means that q^+ lies below all the terrains $\overline{\Lambda}_{t'}$, for $t' > t$.

[27] The planes in $\mathrm{CL}(q^+)$ certainly cross τ'', since they intersect the vertical downward ray emanating from q^+. Note also that some planes that pass below q^+ may have been pruned earlier, and thus do not belong to $\mathrm{CL}(q^+)$.

The following lemma analyzes the performance of the data structure.

Lemma 7.7. *The amortized deterministic cost of an insertion is $O(\log^3 n)$, the amortized deterministic cost of a deletion is $O(\log^5 n)$, and the worst-case deterministic cost of a query is $O(\log^2 n)$.*

Proof. The bound on the query time follows as in Lemma 7.5.

Insertions. Consider an insertion of a plane h. As in Section 7.2, each plane that is in H (i.e., has not been marked as deleted) holds $b(w-i)$ *credits*, each worth $a \log^2 N$ *units*, where i is the current unique location (bin index) of the structure $\mathcal{B}_i$ for which $h \in A(\mathcal{B}_i)$, and $w := \log N + 2$ bounds, by Lemma 7.2, the maximum length of $\mathcal{I}$. Here a is as defined in Section 7.2, and b is another constant parameter that will be chosen later.

We modify the analysis in Lemma 7.5 as follows. Let j be, as before, the index of the first empty bin just before the insertion, and let $H_j := \left(\bigcup_{i=0}^{j-1} A(\mathcal{B}_i)\right) \cup \{h\}$. Define $s = |H_j|$ and $t = \sum_{i=0}^{j-1} |S(\mathcal{B}_i)| + 1$. As in (11,12), using Invariant (D1) (which is identical to Invariant (I1)), we get that $2^{j-1} < t \leq 2^j$. On the other hand, as already remarked, it is possible that $s \ll t$, which may cause us to place the newly constructed substructures $\mathcal{D}^{(u)}$ in bins of rather small indices. The active planes in these bins will then require a larger number of credits, which the scheme in Section 7.2 cannot provide.

To cover the cost of an insertion in such a case, we observe that $s \ll t$ means that most elements in the structures that are destroyed by the insertion are not active in their respective structures. That is, they either are not stored in their substructure (call it $\mathcal{B}_i$), are (marked as) deleted, or were contained in a conflict list of $\mathcal{B}_i$ that has been purged. To exploit this observation, we proceed as follows. Consider some substructure $\mathcal{B}_i$, and let τ be a prism in $\mathcal{B}_i$ (at any level of the hierarchy). If the conflict list of τ has not been purged, we denote by $D(\tau)$ the number of planes in $\mathrm{CL}(\tau)$ that have been (marked as) deleted.[28] We define the potential of $\mathcal{B}_i$ to be

$$\Psi(\mathcal{B}_i) = \left(b' \sum_{\tau \in \Pi_{\neg p}^{(i)}} D(\tau) + b'' \sum_{\tau \in \Pi_p^{(i)}} |\,\mathrm{CL}(\tau)| \right) \log N \tag{15}$$

credits, where b' and b'', with $b' > b''$ are suitable multiples of b, to be set later, $\Pi_{\neg p}^{(i)}$ denotes the set of all non-purged prisms in $\mathcal{B}_i$ and $\Pi_p^{(i)}$ is the set of all purged prisms in $\mathcal{B}_i$. We emphasize once again that $|\,\mathrm{CL}(\tau)|$ denotes the original size of the conflict list, as it was constructed. The purpose of this potential is to provide the credit for the increase in potential (when the number of active planes in the destroyed substructures is small) and to pay for the reinsertions (when a conflict list is purged). The overall potential Ψ^* of the structure, measured in credits (rather than in units of credit), is defined as

$$\Psi^* = \Psi + \sum_{i \geq 0} \Psi(\mathcal{B}_i),$$

where Ψ is the overall number of credits held by the planes in H, as explained above. As mentioned, every plane counted by t but not by s has either been marked as deleted or was contained in a

[28] Again, we also count planes in $C(\mathcal{B}_i) \setminus S(\mathcal{B}_i)$.

conflict list that has been purged. Thus

$$t - s \leq \sum_{i=0}^{j-1} \left(\sum_{\tau \in \Pi_{\neg p}^{(i)}} D(\tau) + \sum_{\tau \in \Pi_p^{(i)}} |\operatorname{CL}(\tau)| \right).$$

The following two cases can arise:

Case (a): $s < (1-\zeta)t$. In this case $t - s > \frac{\zeta}{1-\zeta} s$; that is,

$$\sum_{i=0}^{j-1} \left(\sum_{\tau \in \Pi_{\neg p}^{(i)}} D(\tau) + \sum_{\tau \in \Pi_p^{(i)}} |\operatorname{CL}(\tau)| \right) > \frac{s}{31},$$

or, with a suitable choice of b'', and recalling that $b' > b''$,

$$\sum_{i=0}^{j-1} \Psi(\mathcal{B}_i) > \frac{b''}{31} s \log N.$$

Since all these substructures are now destroyed, this potential will no longer appear in the full potential Ψ^*, and we can safely use it to cover the cost of the insertion. As described in Section 7.2, for a sufficiently large constant b'', this will enable us to place the s planes of H_j at the bins they are to be stored in, no matter where, with the correct amount of credit assigned to each of them (and with change to spare).

Case (b): $s \geq (1-\zeta)t$. Then

$$|S(\mathcal{D}^{(1)})| \geq (1-\zeta)s \geq (1-\zeta)^2 t \geq (1-\zeta)^2 2^{j-1} > 2^{j-2},$$

(which holds for $\zeta = 1/32$), implying that $\mathcal{D}^{(1)}$ is stored at bin j or $j-1$. (Note that, right after the reconstruction caused by an insertion, $A(\mathcal{D}^{(j)}) = S(\mathcal{D}^{(j)})$ for each of the newly constructed substructures $\mathcal{D}^{(j)}$.) Furthermore, the lower bound in Invariants (D1) and our assumption that $s \geq (1-\zeta)t$ show that at least $\sum_{i=1}^{j-2} 2^{i-1} + 2 - \zeta t = 2^{j-2} - \zeta t \geq t/4 - \zeta t$ planes in H_j were stored at bins $i = 0, 1, \ldots, j-2$ prior to the insertion. By Lemma 7.1, the reconstruction following the insertion passes at most $\zeta s \leq \zeta t$ of them to bins of indices $0, 1, \ldots, j-2$, so at least $t/4 - 2\zeta t$ of these planes end up at $\mathcal{B}_j$ or $\mathcal{B}_{j-1}$. These planes release at least $bt(1/4 - 2\zeta)$ credits.

The other planes, that are passed to lower-indexed bins, may require additional credits, but the total number of these credits is bounded as in the proof of Lemma 7.5. It follows that for our choice of $\zeta = 1/32$ and for a somewhat larger choice of b (than the one in Section 7.2), the released credit suffices to pay for the real insertion cost. In all the newly constructed substructures $\mathcal{B}_i$, we have $S(\mathcal{B}_i) = A(\mathcal{B}_i)$, and no conflict list has been purged, so, by definition, $\Psi(\mathcal{B}_i) = 0$; in other words, no credits have to be reallocated for these potentials.

In conclusion, in either of the two cases, the actual cost of the insertion, plus the difference in Ψ^*, is upper bounded by the amortized cost, which, as in Section 7.2, is bw credits, or $O(\log^3 n)$ units of credit.

Deletions. Finally, we analyze the amortized deletion cost. When we delete a plane h from H, we give it $b'_0 \log^3 N$ credits, where b'_0 is some sufficiently large multiple of b', that is, $\Theta(\log^5 N)$ units of credit. To each conflict list $\mathrm{CL}(\tau)$ with $h \in \mathrm{CL}(\tau)$, that has not been purged yet, we allocate, from the credit given to h, $b' \log N$ credits to account for the increase in potential due to the deletion of h (this increase is reflected in (15)). There are at most $c \log N$ such lists in each of the $O(\log N)$ substructures $\mathcal{B}_i$, so there are enough credits to account for the increase in potential.

The marking of h as deleted may lead, via the lookahead deletion mechanism, to the purging of several conflict lists containing h, and to the reinsertion of the active planes in these conflict lists. Let $\mathrm{CL}(\tau)$ be a conflict list in some substructure $\mathcal{B}_i$ that is purged when h is deleted. At this point there are at least $\frac{1}{2\alpha}|\mathrm{CL}(\tau)|$ planes in $\mathrm{CL}(\tau)$ that have been marked as deleted, and the status of $\mathrm{CL}(\tau)$ switches from non-purged to purged. Thus, this switch releases (again, recall (15))

$$\left(b'D(\tau) - b''|\mathrm{CL}(\tau)|\right)\log N \geq \left(\frac{b'}{2\alpha} - b''\right)|\mathrm{CL}(\tau)|\log N$$

credits. The reinsertion itself proceeds exactly as in the case of insertion. With a suitable choice of b' and b'', say $b' \geq 4\alpha b''$ and b'' sufficiently large, the $\Theta(\log N)$ credits needed to support each of the at most $|\mathrm{CL}(\tau)|$ reinsertions are thus available, including the $\Theta(\log N)$ credits that each reinserted plane has to bring along. This implies, as in Section 7.2, that the amortized cost of a deletion is indeed $O(\log^3 N)$ credits, or $O(\log^5 N)$ units.

Each global rebuilding occurs after $\Theta(N)$ updates (insertions and deletions), which have occurred since the last global rebuilding. In the rebuilding, all lingering (marked as) deleted planes are fully removed from the structure. All other planes abandon their present status, and we simply build a static structure from the current planes, storing its substructures at a suitable sequence of bins. The cost of the rebuilding is $O(N \log^2 N)$, so the total cost of all rebuildings is $O(n \log^2 n)$, where n is the total number of updates, plus the size of the initial set of planes (if nonempty). This is well subsumed by the overall amortized cost of the insertions and deletions. □

Storage. So far, the structure requires $O(n \log n)$ cells of storage: $\mathcal{I}$ has $O(\log n)$ substructures, where the substructure $\mathcal{B}_i$ at index i is a hierarchy of cuttings, each approximating some level in a geometric sequence of levels (of suitable subsets of H). Put $n_i := |C(\mathcal{B}_i)| \leq 2^i$. The number of prisms in the cutting for level k is $O(n_i/k)$, and the size of each conflict list is $O(k)$, so the total storage for each level of $\mathcal{B}_i$ is $O(n_i)$, for a total storage of $O(n_i \log n_i)$. Summing over i, the total storage is $O(n \log n)$. A similar analysis shows that the total storage, excluding the conflict lists, is $O(n)$.

Using an idea that is credited to Afshani by Chan [13], we can improve the storage to linear, if for each conflict list we store only its initial size and the number of planes that were deleted in it (except for the lowest-indexed cuttings, where we keep the conflict lists explicitly, but the size of any such lowest-indexed list is only $O(1)$). Then, the storage for $\mathcal{B}_j$ is $O(n_j)$, making the overall storage $O(n)$. To make this work, we need additional mechanisms to compensate for the missing conflict lists. Specifically, when we delete a plane h, we need to find the conflict lists that contain h, and increment the deletion counter of each corresponding prism. Naively, within a substructure $\mathcal{B}_i$, the plane h has to find all the vertices of all the cuttings that lie above it. Each such vertex is a vertex of some prism(s), and h belongs to the conflict list of each such prism. However, h might lie below a vertex v and not belong to the conflict lists of the incident prisms, because h has been pruned away while processing the current or a higher-indexed cutting.

Thus, we augment $\mathcal{B}_j$ with a separate halfspace range reporting data structure for the set of vertices of each of its cuttings. We use the recent algorithm of Afshani and Chan [1] (which can be made deterministic by using the shallow cutting construction of [16]), which preprocesses a set V of points in $\mathbb{R}^3$, in $O(|V| \log |V|)$ time, into a data structure of linear size, so that the set of those points of V that lie above a querying plane h can be reported in $O(\log |V| + t)$ time, with t being the output size. The cost of augmenting $\mathcal{B}_i$ with these reporting structures is subsumed by the cost of building $\mathcal{B}_i$ itself. Now, when deleting a plane h, we access each substructure $\mathcal{B}_i$ of $\mathcal{I}$ with $h \in C(\mathcal{B}_i)$. For this, each plane h stores pointers to all these structures. Since the overall size of the sets $\mathcal{C}(\mathcal{B}_i)$ is $O(n)$, the overall number of such pointers is linear. For each substructure $\mathcal{B}_i$ with $h \in C(\mathcal{B}_i)$, we find the prisms that contain h in their conflict lists. To ensures correctness of this step, h also stores a second pointer, for each $\mathcal{B}_i$ containing it, to the level at which it was pruned; if h was not pruned, we store a null pointer. Now h accesses the halfspace range reporting structures of all the levels higher that the level at which h was pruned, and retrieves from each of these structures the prisms that contain it in their conflict lists. For each such prism τ, we increment its deletion counter by 1. If the counter becomes too large relative to the initial size, as explained above, we purge the entire conflict list, and reinsert its surviving active members into the structure (of course, this step also requires a data structure to be performed efficiently, see below). The total cost of these steps, excluding the one that purges conflict lists that have become too small, is $O(\log^3 n + t)$, where t is the overall number of prisms that store h in their conflict lists. The term $O(\log^3 n)$ arises since we access up to $O(\log n)$ substructures $\mathcal{B}_i$, access up to $O(\log n)$ halfspace range reporting structures at each of them, and pay an overhead of $O(\log n)$ for querying in each of them. Since, by construction, $t = O(\log^2 n)$, this modification, so far, adds $O(\log^3 n)$ to the total of cost of a deletion.

As noted, we also require a mechanism to compute the active members of the conflict lists that are purged in a substructure $\mathcal{B}_i$. To do so, we preprocess the planes of $S(\mathcal{B}_i)$ into a (dual version of a) halfspace reporting data structure that we keep with $\mathcal{B}_i$. We query this structure with each of the at most four vertices of τ, to obtain, in an output-sensitive manner, all the planes of $S(\mathcal{B}_i)$ that cross τ. This structure takes space linear in $|S(\mathcal{B}_i)|$, $O(|S(\mathcal{B}_i)| \log |S(\mathcal{B}_i)|)$ time to build, and can answer a query in $O(\log |S(\mathcal{B}_i)| + t)$ time, where t is the output size. The cost of answering such a query is subsumed by the cost of reinserting the planes, and the cost of constructing this reporting structure is subsumed by the cost of constructing $\mathcal{B}_i$. We thus obtain the following main summary result of this section.

Theorem 7.8. *The lower envelope of a set of n non-vertical planes in three dimensions can be maintained dynamically, so as to support insertions, deletions, and queries, so that each insertion takes $O(\log^3 n)$ amortized deterministic time, each deletion takes $O(\log^5 n)$ amortized deterministic time, and each query takes $O(\log^2 n)$ worst-case deterministic time, where n is the size of the set of planes at the time the operation is performed. The data structure requires $O(n)$ storage.*

8 Dynamic Lower Envelopes for Surfaces

We finally show how to extend the data structure from Section 7 for general surfaces. As mentioned in the introduction, the key observation is that Chan's technique (also with our improvement) is "purely combinatorial": once we have, as a black box, a procedure for efficiently constructing vertical shallow cuttings, accompanied with efficient procedures for the various geometric primitives that are used by the algorithm (which are provided in our algebraic model of computation—see

Section 3 for details), the rest of the algorithm simply organizes and manipulates the given surfaces into standard data structures. Indeed, the whole geometry needed for the deletion lookahead mechanism is encapsulated in the proof of Lemma 7.6, which relies only on the properties of conflict lists in a vertical shallow cutting, which hold for general well-behaved surfaces too. We first show how to find a *vertical* shallow cutting *with* conflict lists, as needed for Chan's technique.

Theorem 8.1. *Let F be a set of n continuous totally defined algebraic functions of constant description complexity, so that the complexity of the lower envelope of any m functions in F is $O(m)$, and let $k \in \{1, \ldots, n\}$. Then there exists a vertical shallow cutting Λ_k for $L_{\leq k}(F)$ with the following properties (where s is the vertical visibility parameter, introduced above, for F):*

1. *The number of prisms in Λ_k is $O((n/k)\log^2 n)$.*
2. *Each prism τ in Λ_k intersects at least k and at most $2k$ functions in F, and the ceiling of τ lies above $L_k(F)$.*
3. *We can find Λ_k and the conflict lists for its prisms in expected time $O(n\log^3 n\lambda_s(\log n))$, using expected space $O(n\log n\lambda_s(\log n))$, where s is the parameter introduced in Section 6.*

Proof. We combine the techniques from Sections 4, 5 and 6. First, set $\lambda = 4c\log n$, for a suitable constant c as in Section 4. Pick t randomly in $\left[\frac{7\lambda}{6}, \frac{5\lambda}{4}\right]$, and let S_k be a random subset of F of size $r_k = 4c(n/k)\log n$. If $r_k > n$, we set $r_k = n$ and we pick t randomly in $[k, 3k/2]$. Denote by $\overline{T}_k$ the t-level in $\mathcal{A}(S_k)$. By Lemma 4.3 and as argued at the end of Section 4, the expected complexity of $\overline{T}_k$ is $O((n/k)\log^2 n)$.

We compute $\overline{T}_k$ as follows: we perform the algorithm from Section 6 on F for the chosen level t, and we stop the randomized incremental construction after r_k steps. The set of functions inserted during these steps constitute the random sample $S_k \subseteq F$. By Theorem 6.5, this step takes expected time $O(nt\lambda_s(t)\log(n/t)\log n) = O(n\log^3 n\lambda_s(\log n))$, and expected space $O(nt\lambda_s(t)) = O(n\log n\lambda_s(\log n))$, where s is as above. As a result, we get the vertical decomposition $\mathrm{VD}_{\leq t}(S_k)$ of $L_{\leq t}(S_k)$ together with the conflict lists (with respect to F) of the prisms in $\mathrm{VD}_{\leq t}(S_k)$. From this, we can extract $\overline{T}_k$ by gluing together the ceilings of all prisms that are met by $L_t(S_k)$. By using the pointers that connect between adjacent prisms in $\mathrm{VD}_{\leq t}(S_k)$, and the pointers that connect the prisms with their vertices in $\mathrm{VD}_{\leq t}(S_k)$, we can do this in $O(|\overline{T}_k|)$ steps. If the complexity of $\overline{T}_k$ exceeds its expectation by more than some preset threshold constant factor, we repeat the whole process with a new random level t. By Markov's inequality, this happens a constant number of times in expectation.

Next, we compute for each function $f \in F \setminus S_k$ the intersection between f and $\overline{T}_k$. For this, we inspect each prism $\tau \in \mathrm{VD}_{\leq t}(S_k)$ that has f in its conflict list and is incident to $\overline{T}_k$, compute the intersection between f and the boundary of τ, and keep the part of this intersection that appears on $\overline{T}_k$. Finally, we glue together the resulting partial curves in order to obtain $f \cap \overline{T}_k$ (this intersection curve does not need to be connected and can be fairly complex). The total time for this step is proportional to the total size of the conflict lists of $\mathrm{VD}_{\leq t}(S_k)$ times a logarithmic factor for the gluing operation. This is $O(n\log^2 n\lambda_s(\log n))$ in expectation, by Theorem 6.5.

Finally, we construct the vertical decomposition $\overline{\Lambda}_k$ of $\overline{T}_k$, in $O(|T_k|\log n) = O((n/k)\log^3 n)$ time, by sweeping the xy-projection of $\overline{T}_k$ with a y-vertical line. By Lemma 5.1, the downward vertical extension Λ_k of $\overline{\Lambda}_k$ is a shallow cutting for the first k levels of $\mathcal{A}(F)$, with high probability. To find the conflict lists of the prisms of Λ_k, we build a planar point location structure for the

xy-projection of $\overline{\Lambda}_k$. Then, for each $f \in F \setminus S_k$, we use the planar point location structure to locate the trapezoid of $\overline{\Lambda}_k$ that contains an initial point on each connected component of $f \cap \overline{T}_k$. Then we use the planar point location structure again to trace each connected component through $\overline{\Lambda}_k$, and pay $O(\log n)$ time for each trapezoid that we cross (we use the planar point location structure to cross through the xz-faces of the prisms, where there may be a lot of prisms on the other side). Then, starting from these trapezoids, we perform another traversal of $\overline{\Lambda}_k$ to find all the trapezoids of $\overline{\Lambda}_k$ that lie fully above f. For all these trapezoids, of both kinds, f is in the conflict list of the corresponding vertical prism, and all members of the conflict lists arise in this manner. Hence, the overall time for this step is proportional to the total size of the conflict lists of Λ_k times a logarithmic factor for the point locations. Thus, perhaps somewhat pessimistically, the total expected running time for this step is $O(k(n/k)\log^3 n) = O(n\log^3 n)$, by Lemma 5.1. The functions $f \in F \setminus S_k$ for which $f \cap \overline{T}_k = \emptyset$ either lie completely above or completely below $\overline{T}_k$. In the former case, such a function is irrelevant, and we simply discard it. In the latter case, it appears in all conflict lists of Λ_k. In the last step, we check whether all prisms actually intersect between k and $2k$ functions from F. If this is not the case, we repeat the whole construction. By the discussion in Section 4 and Markov's inequality, the expected number of attempts is constant.

The total expected running time and storage is dominated by the randomized incremental construction, and hence the theorem follows. □

Remark. The variant of Theorem 8.1 for general lower envelope complexity is as follows:

Theorem 8.2. *Let F be a set of n continuous totally defined algebraic functions of constant description complexity, so that the complexity of the lower envelope of any m functions in F is at most $\psi(m)$, where $\psi(m)/m$ increases monotonically. Furthermore, let $k \in \{1, \ldots, n\}$. Then there exists a vertical shallow cutting Λ_k for $L_{\leq k}(F)$ with the following properties:*

1. *The number of prisms in Λ_k is $O(\psi(n/k)\log^2 n)$.*
2. *Each prism τ in Λ_k intersects at least k and at most $2k$ functions in F, and the ceiling of τ lies above $L_k(F)$.*
3. *We can find Λ_k and the conflict lists of its prisms in $O(\psi(n/\log n)\log^4 n\lambda_s(\log n))$ expected time, using expected space $O(\psi(n/\log n)\log^2 n\lambda_s(\log n))$.*

Proof. The argument is the same, with slightly adjusted bounds. The remark at the end of Section 4 yields the bound on the size of Λ_k. The bound on the running time follows by using Theorem 6.6 and the fact that $\psi(m)/m$ is monotone increasing. □

Now we can combine Theorem 8.1 with the construction in Section 7 to obtain the desired data structure. However, we need to adjust the bounds in our analysis to account for the fact that the cuttings that we construct are of slightly sub-optimal size ($O((n/k)\log^2 n)$ instead of $O(n/k)$), and that we need more (expected) time to construct them ($O(n\log^3 n\lambda_s(\log n))$ instead of $O(n\log n)$). Specifically, we need to apply the following adjustments: Since now the total size of the conflict lists is $O(n\log^2 n)$, when constructing the static data structure (Section 7.1), we prune a function only when it appears in $c\log^3 n$ conflict lists. This increases the overall size of the static structure to $O(n\log^3 n)$, and the construction time becomes $O(n\log^4 n\lambda_s(\log n))$ (in expectation). The query time remains $O(\log^2 n)$, since we can perform point location in general minimization diagrams in $O(\log n)$ time per query. Concerning insertions, the increased construction time implies that in

Lemma 7.5, we need to allocate $\Theta(\log^4 N\lambda_s(\log N))$ units for each credit. Then, the remaining analysis in the proof of Lemma 7.5 continues to hold, and we have an amortized insertion cost of $O(\log^5 N\lambda_s(\log N))$. Finally, we analyze the deletion cost: since now each deleted element can appear in $O(\log^4 n)$ conflict lists, we must equip it with $\Theta(\log^5 N)$ credits to pay for the reinsertions. With the adjusted number of units per credit, this means that each deleted element needs $\Theta(\log^9 N\lambda_s(\log N))$ units to pay for the reinsertions. Since the storage for the dynamic structure is proportional to the storage for the static structure, we need $O(n\log^3 n)$ space overall.

Our efforts so far can thus be immediately reaped into the following main result.

Theorem 8.3. *The lower envelope of a set of n totally defined continuous bivariate functions of constant description complexity in three dimensions, so that the lower envelope of any subset of the functions has linear complexity, can be maintained dynamically, so as to support insertions, deletions, and queries, so that each insertion takes $O(\log^5 n\lambda_s(\log n))$ amortized expected time, each deletion takes $O(\log^9 n\lambda_s(\log n))$ amortized expected time, and each query takes $O(\log^2 n)$ worst-case deterministic time, where n is the number of functions currently in the data structure. The data structure requires $O(n\log^3 n)$ storage in expectation.*

Remark. With the obvious adjustment to the bounds, we get the following theorem for general lower envelope complexity:

Theorem 8.4. *Let F be a finite set of totally defined continuous bivariate functions of constant description complexity in three dimensions, so that the complexity of the lower envelope of any m functions of F is at most $\psi(m)$, where $m \mapsto \psi(m)/m$ is monotonically increasing. Then the lower envelope of F can be maintained dynamically, so as to support insertions, deletions, and queries, so that each insertion takes $O(\frac{\psi(n/\log n)}{n}\log^6 n\lambda_s(\log n))$ amortized expected time, each deletion takes $O(\frac{\psi(n/\log n)}{n}\log^{10} n\lambda_s(\log n))$ amortized expected time, and each query takes $O(\log^2 n)$ worst-case deterministic time, where n is the number of functions currently in the data structure. The data structure requires $O(\psi(n)\log^3 n)$ storage in expectation.*

9 Applications

Let $S \subset \mathbb{R}^2$ be a finite set of pairwise disjoint *sites*, each a simply-shaped convex planar region, e.g., points, line segments, disks, etc. The problem of finding, for a point $q \in \mathbb{R}^2$, its nearest neighbor in S under any norm or convex distance function δ [20] translates to ray shooting in the lower envelope of the set of functions $F = \{f_s(x) = \delta(x, s) \mid s \in S\}$. Thus, if the lower envelope of F has linear complexity, Theorem 8.3 yields a dynamic nearest neighbor data structure for S. We note that the minimization diagram of the lower envelope of F is the Voronoi diagram of S under δ [5, 24].

Dynamic nearest neighbor search has several applications that we are going to mention, but first we introduce two classes of distance functions that are of particular interest.

- **The L_p-metrics**: Let $p \in [1, \infty]$. We define, for $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$, the L_p metric

$$\delta_p((x_1, y_1), (x_2, y_2)) = \begin{cases} (|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p} & \text{for } p < \infty \\ \max\{|x_1 - x_2|,\ |y_1 - y_2|\} & \text{for } p = \infty. \end{cases}$$

It is well known that δ_p is a metric, for any such p, and thus it induces lower envelopes of linear complexity, for any set of sites as above [37]. The precise value for the parameter s defined in Section 6 depends on the choice of p. To ensure a reasonable bound on s, we assume that p is an integer (or $p = \infty$). For a finite integer value of p, we have $s = O(p^2)$, and for $p = \infty$, we have $s = 4$ (as two L_∞-bisectors can intersect at most twice).

- **Additively weighted Euclidean metric**: Let $S \subset \mathbb{R}^2$ be a set of point sites, and suppose that each $s \in S$ has an associated weight $w_s \in \mathbb{R}$. We define a distance function $\delta : \mathbb{R}^2 \times S \to \mathbb{R}$ by $\delta(p, s) = w_s + |ps|$, where $|\cdot|$ denotes the Euclidean distance. This distance function also induces lower envelopes of linear complexity, i.e., the additively weighted Voronoi diagram of point sites has linear complexity [5]. The bisectors for the additively weighted Voronoi diagram are hyperbolic arcs, so each pair of bisectors intersects at most 4 times. Thus, in this case, we have $s = 6$, for the parameter s defined in Section 6.

9.1 Direct applications of dynamic nearest neighbor search

Now we can improve several previous results by plugging our new bounds into known methods.

Dynamic bichromatic closest pair. Let $\delta : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ be a planar distance function,[29] and let $R, B \subset \mathbb{R}^2$ be two sets of point sites in the plane. The *bichromatic closest pair* of R and B with respect to δ is a pair $(r, b) \in R \times B$ that minimizes $\delta(r, b)$. We get the following improved version of Theorem 6.8 in Agarwal et al. [3], which is obtained by combining Eppstein's method [25] with the dynamic lower envelope structure from Theorem 8.3.

Theorem 9.1. *Let R and B be two sets of points in the plane, with a total of at most n points. We can store $R \cup B$ in a dynamic data structure of size $O(n \log^3 n)$, that maintains a closest pair in $R \times B$ under any L_p-metric or any additively weighted Euclidean metric, in $O(\log^{10} n \lambda_s(\log n))$ amortized expected time per insertion and $O(\log^{11} n \lambda_s(\log n))$ amortized expected time per deletion.*

In fact, Chan [14] recently showed how to adapt the data structure from Section 7 directly for the dynamic bichromatic closest pair problem, without incurring the polylogarithmic overhead that is inherent in Eppstein's method [25]. As in Section 8, this improvement also carries over to the case of surfaces. For the interested reader, we explain how Chan's scheme plays out in our presentation.

Theorem 9.2. *Let R and B be two sets of points in the plane, with a total of at most n points. We can store $R \cup B$ in a dynamic data structure of size $O(n \log^3 n)$, that maintains a closest pair in $R \times B$ under any L_p-metric, with p an integer or $p = \infty$, or any additively weighted Euclidean metric, in $O(\log^5 n \lambda_s(\log n))$ amortized expected time per insertion and $O(\log^9 n \lambda_s(\log n))$ amortized expected time per deletion, where s is as defined in Section 6; for additively weighted Euclidean distances we have $s = 6$.*

Proof. We maintain two copies of the dynamic structure from Sections 7 and 8, one for the red points and one for the blue points. In addition, we have a global min-heap H that contains a set of some bichromatic pairs from the current set $R \times B$, where the key for a pair (r, b) is the distance

[29] For the definition of the bichromatic closest pair, δ can be an arbitrary function, but in applications, we usually assume that it is a metric.

$\delta(r, b)$. More precisely, our data structure consists of $O(\log N)$ *red bins* $\mathcal{B}_0^R, \mathcal{B}_1^R, \ldots$ that store the (surfaces corresponding to) the red points, and of $O(\log N)$ *blue bins* $\mathcal{B}_0^B, \mathcal{B}_1^B, \ldots$ that store the (surfaces corresponding to) the blue points, satisfying the same invariants as in Section 7. Here, as before, N is an appropriate power of 2 close to n. As we will see, it is possible to update H such that it contains, at each point in time, the current closest pair in $R \times B$, which we thus can retreive in $O(1)$ time.

To insert a new red point r into R, we add r into the red bins as in Section 7. Then, we update the heap H as follows: for each non-empty blue bin $\mathcal{B}_i^B$, we perform a point-location query to find the prism (or prisms) of the lowest-indexed cutting Λ_0 in $\mathcal{B}_i^B$ whose xy-projection contains r. Next, for each active site b in the conflict list of this prism (or prisms), we insert the pair (r, b) into H, using $\delta(r, b)$ as the key. An insertion into B is symmetric.

To delete a red point r from R, we remove from H all pairs (r, b') that involve r (we can find them quickly by maintaining a list of back-pointers with each element in $R \cup B$). After that, we perform the deletion from the red bins as in Section 7. If this causes a red point r' to be reinserted, we first remove all occurrences of r' from H, as we did for r, and then we reinsert r' into the structure as described in the preceding paragraph. A deletion from B proceeds symmetrically.

These modifications do not affect the asymptotic amortized running times and the space requirement from Sections 7 and 8. Indeed, the deletions from H can be charged to the insertions into H. The cost for the insertions into H (and the accompanying point locations in the xy-projections of the cuttings) can be covered with the units of credit that are spent for the insertion or the reinsertion (see Section 7). The space overhead for H is $O(n \log N)$, since each insertion or reinsertion of a site s can lead to only $O(\log N)$ new pairs in H, while removing all previous pairs involving s.

It remains to argue that the algorithm is correct. More precisely, we show that after each insertion and each deletion, the current closest pair is in H (and has the minimum key).[30]

First, if an update does not change the current closest pair, the invariant is clearly maintained: Indeed, an insertion does not remove pairs from H. The first step of a deletion removes only pairs that involve the site that is to be deleted. If the deletion-lookahead mechanism causes a member of the current closest pair to be reinserted, the closest pair will be added back to H during this step, since by Lemma 7.6, it will be discovered by one of the queries into the lowest-indexed cuttings.

Second, if the current closest pair changes due to an insertion, one of its two sites has just been added, and, as just discussed, Lemma 7.6 ensures that the new closest pair is inserted into H. Finally, suppose the current closest pair changes due to a deletion. Let (r^*, b^*) be the new closest pair. Assume, without loss of generality, that r^* was inserted or last reinserted after b^* was inserted or last reinserted. This means that when r^* was (re)inserted, there was a (unique) blue bin $\mathcal{B}_i^B$ that had the surface for b^* in $A(\mathcal{B}_i^B)$, and $\mathcal{B}_i^B$ was queried with r^*. If this query found (r^*, b^*), then this pair was inserted into H, and the invariant follows. Suppose this query did not find (r^*, b^*). Then, the surface for b^* was not in the conflict list of the prism in the lowest-indexed cutting Λ_0 of $\mathcal{B}_i^B$ whose xy-projection contains r^*. However, by the time (r^*, b^*) becomes the closest pair, Lemma 7.6 ensures that the unique bin $\mathcal{B}_j^B$ where the surface for b^* is active must have exactly this property. Thus, the bin $\mathcal{B}_j^B$ in which b^* is active must have changed since the last (re)insertion of r^*. However, the active bin of b^* can only change due to a reinsertion of b^*. This contradicts our assumption that r^* was inserted or last reinserted after b^* was inserted or last reinserted. □

[30] We assume here that the closest pair is unique, but the argument easily carries over for the more general degenerate case.

Minimum Euclidean bichromatic matching. Let R and B be two sets of n points in the plane (the *red* and the *blue* points). A *minimum Euclidean bichromatic matching* M of R and B is a set M of n line segments that go between R and B such that each point in $R \cup B$ is an endpoint of exactly one line segment in M and such that the total length of the segments in M is minimum over all such sets. Agarwal et al. [3, Theorem 7.1] show how to compute such a minimum Euclidean bichromatic matching in total time $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$, building on a trick by Vaidya [49]. The essence of the algorithm lies in a dynamic bichromatic closest pair data structure for a suitably defined additively weighted Euclidean metric. The algorithm makes $O(n^2)$ updates to this structure. Thus, using Theorem 9.2 (and the fact that $s = 6$ for the case of the additively weighted Euclidean metric), we get the following improvement:

Theorem 9.3. *Let R and B be two sets of points in the plane, each with n points. We can find a minimum Euclidean bichromatic matching for R and B in $O(n^2 \log^9 n\lambda_6(\log n))$ expected time.*

Dynamic minimum spanning trees. Following Eppstein [25], Theorem 9.2 immediately gives a data structure for the dynamic maintenance of the edge set of a minimum spanning tree for any L_p-metric, under insertions and deletions of points. As described by Agarwal et al. [3, Theorem 6.9], it suffices to maintain a suitable collection of bichromatic closest pair structures such that each point appears in $O(\log^2 n)$ such structures, and such that each insertion or deletion of a point requires $O(\log^2 n)$ updates of bichromatic closest pairs. We thus get the following improved version of Theorem 6.9 in Agarwal et al. [3, Theorem 6.9].

Theorem 9.4. *Let p be an integer. We can maintain a minimum spanning tree of a set of at most n points in the plane, under the L_p-metric, such that each insertion and deletion takes $O(\log^{11} n\lambda_s(\log n))$ amortized expected time, using $O(n \log^5 n)$ space, where $s = O(p^2)$.*

Maintaining the intersection of unit balls in three dimensions. Agarwal et al. [3] show how to use dynamic lower envelopes to maintain the intersection of unit balls in three dimensions, so that certain queries on the union can be supported. Their algorithm uses parametric search on the query algorithm in a black box fashion. Thus, we obtain the following improvement over Theorem 8.1 in Agarwal et al. [3]. Since the xy-projection of the intersection of the boundaries of two unit balls in $\mathbb{R}^3$ is a quadratic curve in plane, and since two such curves can intersect in at most four points, we have $s = 6$ where s is the parameter from Section 6.

Theorem 9.5. *The intersection $B^\cap$ of a set B of at most n unit balls in $\mathbb{R}^3$ can be maintained dynamically by a data structure of size $O(n \log^3 n)$, so that each insertion (resp., deletion) takes $O(\log^5 n\lambda_6(\log n))$ (resp., $O(\log^9 n\lambda_6(\log n))$) amortized expected time, and the following queries can be answered: (a) for any query point $p \in \mathbb{R}^3$, we can determine, in $O(\log^2 n)$ deterministic worst-case time, whether $p \in B^\cap$, and (b) after performing each update, we can determine, in $O(\log^5 n)$ deterministic worst-case time, whether $B^\cap \neq \emptyset$.*

Maintaining the smallest stabbing disk. Let $\mathcal{C}$ be a family of simply shaped compact strictly-convex sets in the plane. We wish to dynamically maintain a finite subset $C \subseteq \mathcal{C}$, under insertions and deletions, such that, after each update, we have a smallest disk that intersects all the sets of C (see Agarwal et al. [3, Section 9] for precise definitions). Our structure yields the following improved version of Theorem 9.3 in [3] (the precise value of s depends on the choice of $\mathcal{C}$):

Theorem 9.6. *A set C of at most n (possibly intersecting) simply shaped compact convex sets in the plane can be stored in a data structure of size $O(n \log^3 n)$, so that a smallest stabbing disk for C can be computed in $O(\log^5 n)$ additional deterministic worst-case time after each insertion or deletion. An insertion takes $O(\log^5 n \lambda_s(\log n))$ amortized expected time and a deletion takes $O(\log^9 n \lambda_s(\log n))$ amortized expected time.*

Shortest path trees in unit disk graphs. Let $S \subset \mathbb{R}^2$ be a set of n point sites. The *unit disk graph* $\mathrm{UD}(S)$ of S has vertex set S and an edge between two distinct sites $s, t \in S$ if and only if $|st| \leq 1$. Cabello and Jejčič [10] show how to compute a shortest path tree in $\mathrm{UD}(S)$ for any given root vertex $r \in S$, in time $O(n^{1+\varepsilon})$, for any $\varepsilon > 0$, using the bichromatic closest pair structure for the weighted Euclidean distance from Agarwal et al. [3, Theorem 6.8]. With our improved Theorem 9.2, we get the following result.

Theorem 9.7. *Let $S \subset \mathbb{R}^2$ be a set of n sites. For any $r \in S$, we can compute a shortest path tree with root r in $\mathrm{UD}(S)$ in expected time $O(n \log^9 n \lambda_6(\log n))$.*

We note that Theorem 9.7 has very recently been improved by Wang and Xue [50]. They show how to compute a shortest path tree in a weighted unit disk graph with n sites in $O(n \log^2 n)$ deterministic time.

9.2 Dynamic disk graph connectivity

Next, we describe three further applications of our data structure with improved bounds for problems on disk graphs: Let $S \subset \mathbb{R}^2$ be a finite set of point sites, each with an assigned weight $w_s \geq 1$. Every $s \in S$ corresponds to a disk with center s and radius w_s. The *disk graph* $D(S)$ is the intersection graph of these disks , i.e., $D(S)$ has vertex set S and an edge connects two sites s, t if and only if $|st| \leq w_s + w_t$. In this section, we assume that all weights lie in the interval $[1, \Psi]$, for some $\Psi \geq 1$, and we call Ψ the *radius ratio*. First, we show how to dynamically maintain $D(S)$ under insertions and deletions of weighted vertices (i.e., disks), such that we can answer *reachability queries* efficiently: given $s, t \in S$, is there a path in $D(S)$ from s to t? The amortized expected update time is $O(\Psi^2 \log^8 n)$ for insertions and $O(\Psi^2 \log^{12} n)$ for deletions, and the worst-case cost of a query is $O(\log n / \log \log n)$. Previous results have update time $O(n^{20/21})$ and query time $O(n^{1/7})$ for general disk graphs, and update time $O(\log^{10} n)$ and query time $O(\log n / \log \log n)$ for the unit disk case [15].

Our approach is as follows: let $\mathcal{G}$ be a planar grid whose cells are pairwise openly disjoint axis-aligned squares with diameter (i.e., diagonal) 1. For any grid cell $\sigma \in \mathcal{G}$, since $w_s \geq 1$ for every $s \in S$, the sites of $\sigma \cap S$ induce a clique in $D(S)$. The *neighborhood* $\mathrm{N}(\sigma)$ of a cell $\sigma \in \mathcal{G}$ is the $\big(\lceil 4\sqrt{2}\Psi \rceil + 1\big) \times \big(\lceil 4\sqrt{2}\Psi \rceil + 1\big)$ block of cells in $\mathcal{G}$ with σ at its center. We call two cells *neighboring* if they are in each other's neighborhood. By construction, the endpoints of any edge in $D(S)$ lie in neighboring cells: the side length of a cell is $\sqrt{2}/2$, and there can be at most $\lceil 2\Psi/(\sqrt{2}/2) \rceil = \lceil 2\sqrt{2}\Psi \rceil$ cells between two cells that contain adjacent sites. We define an abstract graph G whose vertices are the *nonempty* cells $\sigma \in \mathcal{G}$, i.e., the cells with $\sigma \cap S \neq \emptyset$. We pick the following edges for G: consider any pair of neighboring grid cells $\sigma, \tau \in \mathcal{G}$. We have an edge between σ and τ if and only if there are two sites $s \in \sigma \cap S$ and $t \in \tau \cap S$ with $|st| \leq w_s + w_t$. By construction, and since the sites inside each cell form a clique, the connectivity between two sites s, t in $D(S)$ is the same as for the corresponding containing cells in G:

Lemma 9.8. *Let $s, t \in S$ be two sites and let σ and τ be the cells of $\mathcal{G}$ containing s and t, respectively. There is an s-t-path in $D(S)$ if and only if there is a path between σ and τ in G.*

To maintain G, we use the following result by by Holm, De Lichtenberg and Thorup, that supports dynamic connectivity queries with respect to *edge* updates [31].

Theorem 9.9 (Holm et al., Theorem 3)**.** *Let G be a graph with n vertices. There exists a deterministic data structure such that (i) we can insert or delete edges into/from G in amortized time $O(\log^2 n)$, and (ii) we can answer reachability queries in worst-case time $O(\log n / \log\log n)$.*

Even though Theorem 9.9 assumes that the number of vertices is fixed, we can use a standard rebuilding method to maintain G dynamically within the same asymptotic amortized time bounds, by creating a new data structure whenever the number of nonempty grid cells changes by a factor of 2. When a site s is inserted into or deleted from S, at most $O(\Psi^2)$ edges in G change, since only the neighborhood of the cell of s is affected. Thus, once this set E of changing edges is determined, we can update G in amortized time $O(\Psi^2 \log^2 n)$, by Theorem 9.9. It remains to describe how to find E. For this, we maintain a *maximal bichromatic matching* (MBM) between the sites in each pair of non-empty neighboring cells, similar to Eppstein's method [25]. The definition is as follows: Let $R \subseteq S$ and $B \subseteq S$ be two sets of sites. An MBM M between R and B is a maximal set of edges in $(R \times B) \cap D(S)$ that form a matching. Using Theorem 8.3, we can easily maintain MBMs.

Lemma 9.10. *Let $R, B \subseteq S$ be two sets with a total of at most n sites. There exists a dynamic data structure that maintains a maximal bichromatic matching of the disk graph $D(R \cup B)$, allowing the insertion or deletion of sites in expected amortized time $O(\log^9 n \lambda_6(\log n))$ per update.*

Proof. We have two dynamic lower envelope structures, one for R and one for B, as in Theorem 8.3, with the weighted distance function $\delta(p, s) = |ps| - w_s$, that allow us to perform nearest neighbor search with respect to δ (i.e., vertical ray shooting at the corresponding lower envelope). We denote by NN_R the structure for R and by NN_B the structure for B. We store in NN_R the currently unmatched points in R, and in NN_B the currently unmatched points in B. When inserting a site r into R, we query NN_B with r to get an ummatched point $b \in B$ that minimizes $|rb| - w_b$. If $|rb| \leq w_r + w_b$, we add the edge rb to M, and we delete b from NN_B. Otherwise we insert r into NN_R. By construction, if there is an edge between r and an unmatched site in B, then there is also an edge between r and b. Hence, with a symmetric procedure for insertions into B, the insertion procedure maintains an MBM. Now suppose we want to delete a site r from R. If r is unmatched, we simply delete r from NN_R. Otherwise, we remove the edge rb from M, and we reinsert b as above, looking for a new unmatched site in R for b. A symmetric procedure handles deletions from B. Since each insertion and deletion of a site requires $O(1)$ insert, delete and query operations in NN_R or NN_B, the lemma follows. □

We create a data structure as in Lemma 9.10 for each pair of non-empty neighboring grid cells. Whenever we insert or delete a site s in a grid cell σ, we update the MBMs for σ and all neighboring cells. Observe that there is an edge between σ and τ if and only if their MBM is not empty. Thus, if s is inserted, we add to G an edge between any pair σ, τ whose MBM changes from empty to non-empty. If s is deleted, we delete all edges between pairs of cells whose MBM changes from non-empty to empty. We thus obtain the following theorem:

Theorem 9.11. *Let $\Psi \geq 1$. We can dynamically maintain the disk graph of a set S of at most n sites in the plane with weights in $[1, \Psi]$ such that (i) we can insert or delete sites in expected*

amortized time $O(\Psi^2 \log^9 n \lambda_6(\log n))$, and (ii) we can determine for any pair of sites s, t whether they are connected by a path in $D(S)$, in determistic worst-case time $O(\log n / \log\log n)$.

As stated above, polylogarithmic bounds were previously known only for the case of unit disk graphs. More precisely, Chan, Pătraşcu, and Roditty mention that one can derive from known results an update time of $O(\log^{10} n)$ [15]. An extension of our method leads to significantly improved bounds for this case, too. Namely, for unit disks, we can obtain amortized expected update time $O(\log^2 n)$ with worst-case query time $O(\log n / \log\log n)$, and amortized expected update time $O(\log n \log\log n)$ with worst-case query time $O(\log n)$ [33]. Very recently, Kauer and Mulzer [35] showed how to improve the dependence on Ψ in Theorem 9.11, at the cost of slightly slower queries.

9.3 Breadth-first-search in disk graphs

As observed by Roditty and Segal [45] in the context of unit disk graphs, a dynamic nearest neighbor structure can be used for computing exact BFS-trees in disk graphs. More precisely, let $D(S)$ be a disk graph with n sites as in Section 9.2, and let $r \in S$. To compute a BFS-tree with root r in $D(S)$, we build a dynamic nearest neighbor data structure for the weighted Euclidean distance (the weights correspond to the radii) and we insert all points from $S \setminus \{r\}$. At each point in time, the dynamic nearest neighbor data structure contains those sites that are not yet part of the BFS-tree. To find the new neighbors of a site p of the partial BFS-tree T, we repeatedly find and delete a nearest neighbor of p in $S \setminus T$, until the next nearest neighbor is not adjacent to p in $D(S)$. By construction, the other farther disks are also not neighbors of p. The successful queries are charged to the BFS-edges, and the last unsuccessful query is charged to p. Thus, the total number of operations on the data structure is $O(n)$. We get the following theorem:

Theorem 9.12. *Let S be a set of n weighted sites in the plane, and let $r \in S$. Then, we can compute a BFS-tree in $D(S)$ with root r in total expected time $O(n \log^9 n \, \lambda_6(\log n))$.*

9.4 Spanners for disk graphs

Finally, we discuss how to use our data structure to efficiently compute *spanners* in disk graphs; see also Seiferth's thesis [47] for more details. Let $D(S)$ be a disk graph with n sites as in Section 9.2, and let $\varepsilon > 0$. A $(1+\varepsilon)$-*spanner* for $D(S)$ is a subgraph $H \subseteq D(S)$ such that, for any $s, t \in S$, the shortest-path distance $d_H(s,t)$ between s and t in H is at most $(1+\varepsilon)d(s,t)$, where $d(s,t)$ is the shortest-path distance in $D(S)$. Fürer and Kasiviswanathan [27] show that a simple construction based on the *Yao graph* [51] yields a $(1+\varepsilon)$-spanner for $D(S)$ with $O(n/\varepsilon)$ edges. It goes as follows: let $\mathcal{C}$ be a set of $k = O(1/\varepsilon)$ cones, each with opening angle $2\pi/k$ that partition the plane. For each site $t \in S$, we translate $\mathcal{C}$ to t, and for each translated cone C, we select a site $s \in S$ with the following properties (if it exists): (i) s lies in C and st is an edge of $D(S)$; (ii) we have $w_s \geq w_t$; and (iii) among all sites with properties (i) and (ii), s minimizes the distance to t. We add the edge st to H. Fürer and Kasiviswanathan show that this construction yields a $(1+\varepsilon)$-spanner for $D(S)$ [27, Lemma 1]. However, it is not clear how to implement this construction efficiently. Therefore, Fürer and Kasiviswanathan show that it is sufficient to relax property (iii) and to require only an *approximate* shortest edge in each cone. Using this, they construct such a relaxed spanner in time $O(n^{4/3+\delta}\varepsilon^{-4/3} \log^{2/3} \Psi)$, where $\delta > 0$ can be made arbitrarily small and all radii lie in the interval $[1, \Psi]$.

We can improve this running time by combining our new dynamic nearest neighbor structure with techniques that we have developed for *transmission graphs* [32]. Let S be a set of n weighted point sites as above. The *transmission graph* of S is a *directed* graph on S with an edge from s to t if and only if $|st| \leq w_s$, i.e., t lies in the disk of s. A similar Yao-based construction as above yields a $(1+\varepsilon)$-spanner for directed transmission graphs: take for each site $t \in S$ and each cone C the shortest incoming edge to t in C. Again, it is not clear how to obtain this spanner efficiently. To solve this problem, Kaplan et al. [32] proceed as Fürer and Kasiviswanathan and describe strategies to compute relaxed versions of this spanner using only an approximate shortest edge in each cone.

One strategy to obtain a running time of $O(n \log^4 n)$ is as follows:[31] we compute a *compressed quadtree* T for S [28]. Let σ be a cell in T, and let $|\sigma|$ be the diameter of σ. We augment T such that for every edge st in the transmission graph, there exist cells σ, τ in T with diameters $|\sigma| = |\tau| = \Theta(\varepsilon|st|)$ and with $s \in \sigma$, $t \in \tau$. In particular, if st is the shortest edge in a cone with apex t, then any edge $s't$ with $s' \in \sigma$ is sufficient for our relaxed spanner, see Figure 4. Kaplan et al. show that this augmentation requires adding $O(n)$ additional nodes to T, which can be found in $O(n \log n)$ time. Furthermore, we compute for each cell σ the set $W_\sigma = \{s \in \sigma \cap S \mid w_s = \Theta(|\sigma|/\varepsilon)\}$. Our strategy is to select spanner edges between sites in cells $\sigma, \tau \in T$ with $|\sigma| = |\tau|$ whose distance is $\Theta(|\sigma|/\varepsilon)$. Since a site can be contained in many cells of T, we consider for each pair σ, τ only the sites in W_σ for outgoing edges. This avoids checking sites in σ whose radius is too small to form an edge with sites in τ. Sites whose radius is too large to be in W_σ can be handled easily; see below. By definition of the sets W_σ each site appears in a constant number of such sets, which is crucial in obtaining an improved running time.

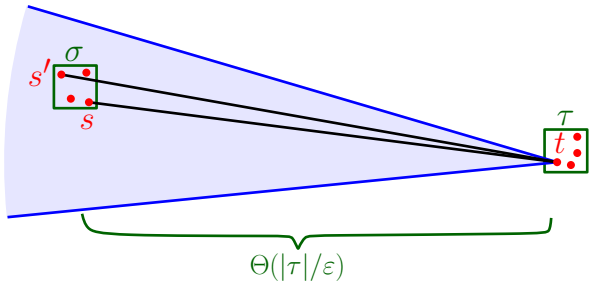

Figure 4: A cone C (blue) and the shortest edge st in this cone. Any edge $s't$ with $s' \in \sigma$ has approximately the same length as st.

Now we can sketch the construction algorithm for the spanner H. We go through all cones $C \in \mathcal{C}$. For each C, we perform a level order traversal of the cells in T, starting with the lowest level. For each cell τ in T, we find the approximate incoming edges of length $\Theta(|\tau|/\varepsilon)$ with respect to C that go into the *active* sites in $S \cap \tau$, i.e., those sites in τ for which no such edge has been found in a previous level. See Algorithm 1 for pseudocode how to process a pair C, τ. To do this, we consider the cells of T that have diameter $|\tau|$ and distance $\Theta(|\tau|/\varepsilon)$ from τ and that intersect the translated copy of C whose center is in the center of τ. For each such cell σ, we first check whether the disk corresponding to the largest site s in σ contains τ completely. If so, we add edges from s to all active sites in $t \in \tau$. This step covers all edges from sites in σ whose radius is too large to be in W_σ. Otherwise, we check for each site in W_σ whether it has edges to active sites in τ.

[31] The original result claims a running time of $O(n \log^5 n)$, but with Chan's recent improvement for dynamic Euclidean nearest neighbors [14], this result also improves.

This test is performed with a dynamic Euclidean nearest neighbor data structure that stores the active sites in τ: while the nearest neighbor t in τ for the current site $s \in W_\sigma$ has $|st| \leq w_s$, we add the edge st to H, and we remove t from the nearest neighbor structure. Otherwise, we proceed to the next site in W_σ. The resulting graph H has $O(n/\varepsilon^2)$ edges and contains for each site t and for each cone C attached to t an approximately shortest incoming edge for t.

```
1  let γ be the child of τ whose nearest neighbor structure NN_γ contains the most sites
2  for each child γ' ≠ γ of τ, insert all sites in γ' into NN_γ; let NN_τ = NN_γ
3  foreach σ ∈ T with |σ| = |τ| and distance O(|τ|/ε) from τ that intersects C do
4  |  if disk of site s ∈ σ with largest weight contains τ then
5  |  |  for each t ∈ NN_τ add the edge st to H
6  |  else
7  |  |  foreach s ∈ W_σ do
8  |  |  |  t ← NN_τ(s) ;                          // query NN structure of τ with s
9  |  |  |  while |st| ≤ w_s and t ≠ ∅ do
10 |  |  |  |  add the edge st to H; delete t from NN_τ; t ← NN_τ(s)
11 |  |  |  reinsert all deleted points into NN_τ
12 delete all sites t from NN_τ for which at least one edge st was found (i.e., make them
     inactive)
```

Algorithm 1: Selecting incoming edges for the points of a node τ of T and a cone C.

The nearest neighbor structures can be maintained with logarithmic overhead throughout the level-order traversal: we initialize them at the leaves of T, and when going to the next level, we obtain the nearest neighbor structure for each cell by inserting the elements of the smaller child structures into the largest child structure. For more details, we refer to Kaplan et al. [32] and to the thesis of Seiferth [47]. They prove that the running time is dominated by the $O(n \log n)$ insertions and $O(n/\varepsilon^2)$ deletions to the dynamic nearest neighbor structure.

A similar strategy works for disk graphs. Given S, we compute an augmented quadtree T for S as above in order to obtain an approximate representation of the distances in S. Furthermore, we compute for each cell σ in T an appropriate set W_σ of assigned sites s from $S \cap \sigma$ with $w_s = \Theta(|\sigma|/\varepsilon)$, as above. To construct the spanner, we perform the level order traversals of the cells in T as before, going through all cones $C \in \mathcal{C}$ and through all cells in T from bottom to top. Now, suppose we visit a cell τ of T, and let σ be a cell of T with diameter $|\sigma| = |\tau|$ and distance $\Theta(|\tau|/\varepsilon)$ from τ that intersects the translated copy of C with apex in the middle of τ. As in Algorithm 1, our goal is to find all "incoming" edges from W_σ for the active sites in τ, where an *incoming* edge for τ now is an edge st with $t \in \tau$ and $w_s \geq w_t$ (recall property (ii) from the original construction of Fürer and Kasiviswanathan). We store the active sites of τ in a dynamic nearest neighbor data structure NN_τ for the metric $\delta(s,t) = |st| - w_t$, instead of the Euclidean metric. To ensure that we find only edges from larger to smaller disks, we sort the disks in W_σ by radius, and besides NN_τ, we also maintain a list L_τ of all sites in $\tau \cap S$ sorted by radius during the traversal of T.

We change lines 7–11 in Algorithm 1 as follows: we query the sites from W_σ in order from small to large. Before querying a site s, we use L_τ to insert into NN_τ all active sites with weight at most w_s that are not yet in NN_τ. We keep querying NN_τ with s as long as the resulting nearest neighbor t corresponds to a disk that intersects the disk of s, and we add these edges st to H. After that, we proceed to the site $s' \in W_\sigma$ with the next larger radius, and we again insert all remaining active sites

with weight at most $w_{s'}$ from L_τ into NN_τ (all these sites t have $w_s \leq w_t \leq w_{s'}$). After processing W_σ, we proceed with the next cell σ'. To ensure that our nearest neighbor queries still return only smaller disks, we need to delete all sites in NN_τ whose weight is larger than the smallest weight in $W_{\sigma'}$. This can be done by deleting all sites in W_τ from NN_τ, and reinserting only the relevant ones. By definition of W_τ, for each site the additional insertions and deletions to maintain NN_τ occur only for a constant number of pairs σ, τ, accounting for an additional $O(n)$ insertions and deletions per site. An analysis similar to the one performed by Kaplan et al. [32] for transmission graphs now shows that H can be constructed in time $O(n \log n)$ plus the time for $O(n \log n)$ insertions and $O(n/\varepsilon^2)$ deletions in the dynamic nearest neighbor structure. By Theorem 8.3, we thus obtain the following result:

Theorem 9.13. *Let S be a set of n weighted sites in the plane, and let $\varepsilon > 0$. Then, we can construct a $(1+\varepsilon)$ spanner for $D(S)$ in expected time $O((n/\varepsilon^2) \log^9 n \lambda_6(\log n))$.*

References

[1] Peyman Afshani and Timothy M. Chan. On approximate range counting and depth. *Discrete Comput. Geom.*, 42(1):3–21, 2009.

[2] Pankaj K. Agarwal, Mark de Berg, Jiří Matoušek, and Otfried Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. *SIAM J. Comput.*, 27(3):654–667, 1998.

[3] Pankaj K. Agarwal, Alon Efrat, and Micha Sharir. Vertical decomposition of shallow levels in 3-dimensional arrangements and its applications. *SIAM J. Comput.*, 29(3):912–953, 1999.

[4] Pankaj K. Agarwal and Jiří Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995.

[5] Franz Aurenhammer, Rolf Klein, and Der-Tsai Lee. *Voronoi Diagrams and Delaunay Triangulations*. World Scientific Publishing, Singapore, 2013.

[6] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, Berlin Heidelberg, second edition, 2006.

[7] Jon Louis Bentley and James B. Saxe. Decomposable searching problems. I. Static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980.

[8] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and applications*. Springer-Verlag, Berlin Heidelberg, third edition, 2008.

[9] Jean-Daniel Boissonnat and Monique Teillaud. *Effective Computational Geometry for Curves and Surfaces*. Mathematics and Visualization. Springer-Verlag, Berlin Heidelberg, 2007.

[10] Sergio Cabello and Miha Jejčič. Shortest paths in intersection graphs of unit disks. *Comput. Geom.*, 48(4):360–367, 2015.

[11] Timothy M. Chan. Random sampling, halfspace range reporting, and construction of ($\leq k$)-levels in three dimensions. *SIAM J. Comput.*, 30(2):561–575, 2000.

[12] Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM J. Comput.*, 34(4):879–893, 2005.

[13] Timothy M. Chan. A dynamic data structure for 3-D convex hulls and 2-D nearest neighbor queries. *J. ACM*, 57(3):16:1–15, 2010.

[14] Timothy M. Chan. Dynamic geometric data structures via shallow cuttings. In *Proc. 35th Int. Sympos. Comput. Geom. (SoCG)*, page to appear, 2019.

[15] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011.

[16] Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.*, 56(4):866–881, 2016.

[17] Bernard Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[18] Bernard Chazelle. *The Discrepancy Method*. Cambridge University Press, 2000.

[19] Bernard Chazelle, Herbert Edelsbrunner, Leonidas J. Guibas, and Micha Sharir. A singly-exponential stratification scheme for real semi-algebraic varieties and its applications. In *Proc. 31st Internat. Colloq. Automata Lang. Program. (ICALP)*, pages 179–193, 1989.

[20] L. Paul Chew and Robert L. (Scot) Drysdale III. Voronoi diagrams based on convex distance functions. In *Proc. 1st Annu. Sympos. Comput. Geom. (SoCG)*, pages 235–244, 1985.

[21] Vasek Chvátal. The tail of the hypergeometric distribution. *Discrete Mathematics*, 25(3):285–287, 1979.

[22] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM J. Comput.*, 17(4):830–847, 1988.

[23] Kenneth L. Clarkson and Peter W. Shor. Applications of random sampling in computational geometry. II. *Discrete Comput. Geom.*, 4(5):387–421, 1989.

[24] Herbert Edelsbrunner and Raimund Seidel. Voronoi diagrams and arrangements. *Discrete Comput. Geom.*, 1(1):25–44, 1986.

[25] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995.

[26] Jeff Erickson. Static-to-dynamic transformations. Lecture notes. `http://jeffe.cs.illinois.edu/teaching/datastructures/notes/01-statictodynamic.pdf`.

[27] Martin Fürer and Shiva Prasad Kasiviswanathan. Spanners for geometric intersection graphs with applications. *J. Comput. Geom.*, 3(1):31–64, 2012.

[28] Sariel Har-Peled. *Geometric Approximation Algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, Providence RI, 2011.

[29] Sariel Har-Peled, Haim Kaplan, and Micha Sharir. Approximating the k-level in three-dimensional plane arrangements. In M. Loebl, J. Nešetřil, and R. Thomas, editors, *Journey through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 467–504. Springer-Verlag, Berlin Heidelberg, 2017.

[30] Sariel Har-Peled and Micha Sharir. Relative (p, ϵ)-approximations in geometry. *Discrete Comput. Geom.*, 45(3):462–496, 2011.

[31] Jacob Holm, Kristian de Lichtenberg, and Mikkel Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *J. ACM*, 48(4):723–760, 2001.

[32] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners and reachability oracles for directed transmission graphs. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 156–170, 2015.

[33] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Dynamic connectivity for unit disk graphs. In *Proc. 32nd European Workshop Comput. Geom. (EWCG)*, pages 183–186, 2016.

[34] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth. Spanners for directed transmission graphs. *SIAM J. Comput.*, 47(4):1585–1609, 2018.

[35] Alexander Kauer and Wolfgang Mulzer. Dynamic disk connectivity. In *Proc. 35th European Workshop Comput. Geom. (EWCG)*, pages 50:1–6, 2019.

[36] Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004.

[37] D. T. Lee. Two-dimensional Voronoĭ diagrams in the L_p-metric. *J. ACM*, 27(4):604–618, 1980.

[38] Yi Li, Philip M. Long, and Aravind Srinivasan. Improved bounds on the sample complexity of learning. *J. Comput. System Sci.*, 62(3):516–527, 2001.

[39] Jiří Matoušek. Reporting points in halfspaces. *Comput. Geom.*, 2(3):169–186, 1992.

[40] Ketan Mulmuley. On levels in arrangements and Voronoi diagrams. *Discrete Comput. Geom.*, 6:307–338, 1991.

[41] Ketan Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice-Hall, 1994.

[42] Wolfgang Mulzer. Five proofs of Chernoff's bound with applications. *Bulletin of the EATCS*, 124, 2018.

[43] Mark H. Overmars and Jan van Leeuwen. Two general methods for dynamizing decomposable searching problems. *Computing*, 26(2):155–166, 1981.

[44] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Annu. Sympos. Comput. Geom. (SoCG)*, pages 390–399 (electronic), 1999.

[45] Liam Roditty and Michael Segal. On bounded leg shortest paths problems. *Algorithmica*, 59(4):583–600, 2011.

[46] Jacob T. Schwartz and Micha Sharir. On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds. *Adv. in Appl. Math.*, 4(3):298–351, 1983.

[47] Paul Seiferth. *Disk Intersection Graphs: Models, Data Structures, and Algorithms*. PhD thesis, Freie Universität Berlin, 2016.

[48] Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications*. Cambridge University Press, 1995.

[49] Pravin M. Vaidya. Geometry helps in matching. *SIAM J. Comput.*, 18(6):1201–1225, 1989.

[50] Haitao Wang and Jie Xue. Near-optimal algorithms for shortest paths in weighted unit-disk graphs. In *Proc. 35th Int. Sympos. Comput. Geom. (SoCG)*, page to appear, 2019.

[51] Andrew Chi-Chih Yao. On constructing minimum spanning trees in k-dimensional spaces and related problems. *SIAM J. Comput.*, 11(4):721–736, 1982.

A The randomized incremental construction

We provide the details of the randomized incremental construction from Section 6.

A.1 Setup

Each three-dimensional cell τ of $\mathrm{VD}_{\leq t}(F_i)$ is a *pseudo-prism*, with up to six faces. We have already used "floor" and "ceiling" to refer, respectively, to the bottom and the top face of τ; we also call them the *xy-faces* of τ, as they extend (in a suitable sense) in the x- and y-directions. The *forward xz-face* and the *backward xz*-face are the "curtains" that extend in the x- and z-directions and bound τ in the respective positive and negative y-directions. They are each erected from an edge of τ that is a portion of an intersection curve between the surface supporting the floor or ceiling of τ with another surface. We collectively refer to these curtains as the *xz-faces* of τ. Similarly, we refer to the left and right faces collectively as *yz-faces*. They originate from lifting the vertical edges of the planar vertical decomposition of the stage-1 cells; see Figure 5 for an illustration. A similar notation applies to the edges of τ. There are three kinds of edges: (i) an *x-edge*, which is the common edge of an xy-face and an xz-face of τ. It is either (a portion of) a *real* intersection curve, or a *shadow edge*, that lies vertically below or above a real intersection edge on the other (floor or ceiling) side of τ; (ii) a *y-edge*, which is the common edge of an xy-face and a yz-face of τ; and (iii) a z-edge, a straight z-parallel segment, which is a common edge of an xz-face and a yz-face.

To navigate in $\mathrm{VD}_{\leq}(F_i)$, each prism $\tau \in \mathrm{VD}_{\leq}(F_i)$ maintains pointers to the *adjacent* prisms that share (part of) a yz-face with it. By general position, there are only $O(1)$ such prisms. Moreover, for each vertex v of a stage-1 cell (i.e., an intersection of three surfaces in F_i; the projection of such

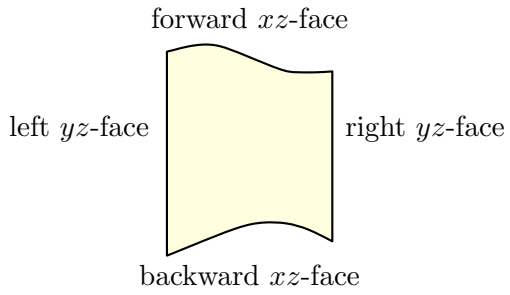

Figure 5: A top view of a prism τ and its xz-faces and yz-faces.

an intersection onto a vertically visible surface in F_i above or below; or the projection of a vertical visibility between two edges onto the top or the bottom edge of the pair), we maintain pointers to all *incident* prisms that have v as a vertex, and in each prism, we maintain reverse pointers to the incident stage-1 vertices. Again by general position, we need only $O(1)$ pointers with each vertex and with each prism. These pointers allow us to perform a BFS-search within the prisms of a stage-1 cell, and to switch from one stage-1 cell to another one with a common vertex. Finally, we maintain a pointer to a vertex that lies on the t-level $L_t(F_i)$ of the current arrangement. This vertex will serve as a starting point when we need to traverse the t-level.

A.2 Inserting a surface

Suppose we add a new surface $f = f_{i+1}$ to some prefix F_i of F. We have the vertical decomposition $\mathrm{VD}_{\leq t}(F_i)$, where each prism $\tau \in \mathrm{VD}_{\leq t}(F_i)$ has an associated conflict list $\mathrm{CL}(\tau)$, consisting of the surfaces of $F \setminus F_i$ that cross τ. Our task is to obtain $\mathrm{VD}_{\leq t}(F_{i+1})$ with the conflict lists of its prisms, each consisting of surfaces in $F \setminus F_{i+1}$. We do this in three steps. First, we find the vertical decomposition of the part of the arrangement $\mathcal{A}(F_{i+1})$ that is below the t-level of $\mathcal{A}(F_i)$; we denote this portion by $\mathcal{A}^{+f}_{\leq t}(F_i)$. Second, we construct the conflict lists of the new prisms. Third, we discard prisms that lie above $\mathcal{A}_{\leq t}(F_{i+1})$ from the vertical decomposition obtained in the first step.

First step: Constructing the vertical decomposition of $\mathcal{A}^{+f}_{\leq t}(F_i)$. When f is inserted, we retrieve the set Π_f of all *old* prisms of $\mathrm{VD}_{\leq t}(F_i)$ that f crosses, using back pointers to the conflict lists that contain f. Note that any old prism $\tau \notin \Pi_f$ remains valid in the vertical decomposition of $\mathcal{A}^{+f}_{\leq t}(F_i)$, but, in case f passes fully below τ, the level of τ in $\mathcal{A}(F_{i+1})$ (compared to its level in $\mathcal{A}(F_i)$) increases by 1. Hence, τ may find itself above the t-level, in which case we discard it in the third step. On the other hand, any old prism $\tau \in \Pi_f$ is destroyed, being split into several *fragments*. More precisely, we compute the vertical decomposition for f locally within τ (considered as a closed set). This splits τ into $O(1)$ smaller pseudo-prisms (the fragments), and we compute the conflict list for each fragment, by brute-force inspection of $\mathrm{CL}(\tau)$. This takes $O\big(|\Pi_f| + \sum_{\tau\in\Pi_f} |\,\mathrm{CL}(\tau)|\big)$ time. Our strategy is to use these fragments to find the region of $A^{+f}_{\leq t}(F_i)$ that is affected by f and to construct the *new* prisms in this region from scratch. The conflict lists of the fragments will be useful in finding the conflict lists for the new prisms.

Each new prism τ' must involve f as one of its (up to) six defining surfaces. That is, it must contain a bounding feature that lies on f. This feature could be a face (when f forms the floor or

ceiling of τ'); an x-edge (where f intersects the floor or ceiling of τ' at a boundary edge, but does not meet the interior of τ'); or a vertex (which is either an intersection point of f with an edge of τ' at its endpoint, or a locally x-extremal point of an intersection curve on f, which defines a yz-face of τ'; see Figure 6).

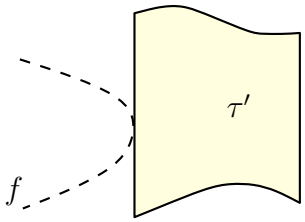

Figure 6: A yz-face of a new prism τ' (seen here as the left side of this top view) formed by a locally x-extremal point of some intersection edge with f.

New prisms with an xy-face on f. We first consider the construction of new prisms of this type. We begin by finding and tracing the x-edges of $\mathrm{VD}(\mathcal{A}^{+f}_{\leq t}(F_i))$ along f. More precisely, we collect the edges along f of stage 1 of the vertical decomposition of $\mathcal{A}^{+f}_{\leq t}(F_i)$. Following the terminology just introduced, each such edge is either a real intersection edge between f and an older surface, or a shadow edge, i.e., the vertical projection of the portions of some real intersection edge that are vertically visible from f.

We think of f as two-sided, having a top side and a bottom side, appearing as two disjoint copies of f, infinitesimally separated from one another. The real intersection edges are drawn on both sides of f, but each shadow edge is drawn only on one side (top or bottom) of f; it is the top (resp., bottom) side of f if the edge is a shadow of a real edge above (resp., below) f. Thus, we obtain two different maps on f, called M^t_f and M^b_f, drawn on the top and bottom sides of f, respectively. The maps M^t_f and M^b_f have three kinds of vertices. The first kind arises when an intersection edge e between two other surfaces f' and f'' crosses f, generating a real vertex v of $\mathcal{A}^{+f}_{\leq t}(F_i)$. We break e into two subedges e^+, e^-, where e^+ lies above f and e^- lies below f, locally near v. Then, the vertical projection of e^+ on f is drawn only in M^t_f, and that of e^- only in M^b_f. Both projections are arcs emanating from v. See, e.g., Figure 7, where the intersection curve between the surfaces a and b intersects f in a vertex v. The second kind of vertex arises from a real vertex w of $\mathcal{A}^{+f}_{\leq t}(F_i)$, incident to three surfaces f_1, f_2, f_3, so that w is vertically visible from f and lies, say, above f. Then, w is incident to three intersection edges of pairs from $\{f_1, f_2, f_3\}$. Each such edge is split at w into two portions, one visible from f and one invisible (hidden by the third function), so we draw in M^t_f three respective projected arcs, all emanating from the projection of w. See Figure 7 for an illustration, where the real vertex w defined by a, d, and g and the real vertex w' defined by d, h, and k form the shadow vertices $\tilde{w}$ and $\tilde{w}'$ on f. The third kind of vertex arises from a vertical visibility between a real edge e on f and another edge e' lying, say, above f. Let z be the point on e where this vertical visibility occurs. Then e' is visible from f only on one side of z, as the other surface forming with f the edge e rises above f on the other side and hides e'. We draw, on M^t_f only, the projection of e' on f, and terminate it at z. See Figure 7, where the vertices $z_1, \ldots, z_5$ represent such pairs of vertical visibility. Finally, there might be shadow edges that do not cross

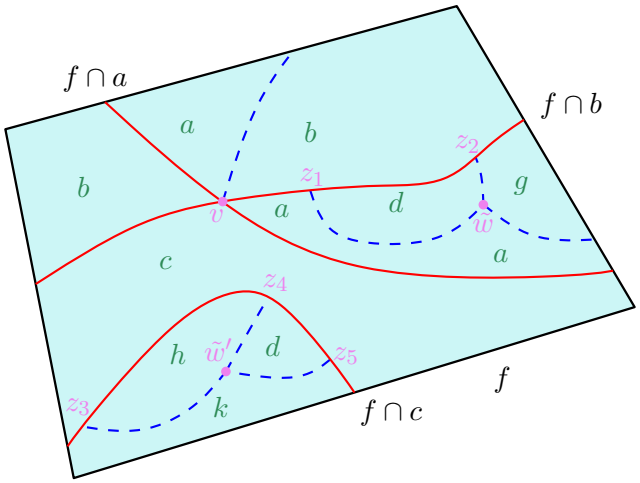

Figure 7: The first stage of the vertical decomposition on (the top side of) the newly inserted surface f. Red (solid) arcs depict real intersection edges and blue (dashed) arcs depict shadow arcs. The label of each face denotes the surface that appears vertically above f and is visible from f over that face. The three types of vertices are depicted: v is a real vertex, $\tilde{w}$ and $\tilde{w}'$ are shadow vertices, and $z_1, \ldots, z_5$ represent vertical visibilities.

any real edge (so they are not involved in any vertically visible pair). These edges are projections of full (and fully visible) edges of $\mathcal{A}(F_i)$, which are either closed or unbounded Jordan curves.

To construct M_f^t and M_f^b, we go through all old prisms $\tau \in \Pi_f$, and we compute the intersection of f with the boundary $\partial\tau$. The pieces of the real intersection edges are obtained from the intersections of f with the floor and/or the ceiling of τ. Each such intersection consists of $O(1)$ connected subarcs. When such an edge e leaves τ (at an endpoint of some intersection subarc), it can do so either through a y-edge or through an x-edge. In the former case (crossing a y-edge), we must glue e to a suitable portion of its continuation into the appropriate old prism adjacent to τ, whereas in the latter case (crossing an x-edge) the crossing point v is either a real vertex of $\mathcal{A}^{+f}_{\leq t}(F_i)$ on e (an endpoint of e), or part of a vertically visible pair in $\mathcal{A}^{+f}_{\leq t}(F_i)$ consisting of (a suitable extension of) e and the real intersection edge of τ on its other side (floor or ceiling). When v is a real vertex, it delimits two edges of $\mathcal{A}^{+f}_{\leq t}(F_i)$, lying on the same intersection curve, one of which is e (within τ) and the other enters an adjacent prism; in this case v is a feature of both M_f^t and M_f^b. When v encodes a vertical visibility pair, it appears only in one of the maps M_f^t or M_f^b; see Figure 8.

To obtain the shadow edges, we consider the intersections of the xz-faces of τ with f, and we use the local information at τ. For example, if an intersection edge e (between two surfaces of F_i) lies on the top side of τ, we take the xz-face φ bounded from above by e, intersect f with φ, and for each connected arc e' of that intersection, we draw the shadow of e in M_f^t over e' as e' itself; see Figure 9. The other portions of e (e.g., the middle portion in Figure 9) are not handled in τ, since the information at τ does not let us know whether these pieces are at all visible (in their entirety) from f; these pieces are handled within other nearby prisms from Π_f that have e (or an edge overlapping e) as an x-edge. (Note that, in Figure 9, any visible part (to f) of the middle portion of e is drawn on the bottom map M_f^b.) Real intersection edges on the bottom side of τ

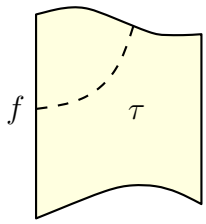

Figure 8: The intersection curve of f with the floor or ceiling of a prism τ (view from above). At the left exit point of the curve (through a y-edge) we glue it to its continuation within an adjacent prism. The top exit point (through an x-edge) is a real feature (vertex or part of a vertically visible pair) of the new decomposition.

are handled in a fully symmetric manner, and their relevant portions are drawn as shadow edges in M_f^b.

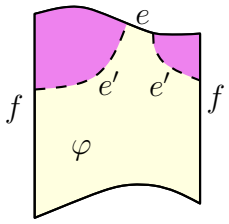

Figure 9: Creating shadow edges (projections of suitable portions of e) on the top side of f within a vertical curtain φ (a side view).

We perform two planar sweeps to assemble the pieces of real and shadow edges on the two sides of f, and to obtain DCEL-representations of M_f^t and of M_f^b. For each stage-1 cell of the vertical decomposition of $\mathcal{A}_{\leq t}^{+f}(F_i)$ (i.e., the cells obtained by erecting vertical curtains through the edges of the arrangement) that has its floor or its ceiling on f, there is a face in M_f^t (resp., in M_f^b). Note that M_f^t and M_f^b may have additional faces that correspond to patches of f that lie above $\mathcal{A}_{\leq t}(F_i)$. Consider for specificity only M_f^t. Recall that each such stage-1 cell has a unique floor (a portion of f in this case) and a unique ceiling (a portion of another surface), but its complexity can be arbitrarily large (its floor and ceiling need not even be simply connected, although, by construction, they are always connected). Denote by H the collection of these stage-1 cells that have their floor on f. Each cell $B \in H$ is the union of fragments of prisms in Π_f (each obtained as we decompose prisms of Π_f into smaller "local" prisms) with their floor on f. Each prism from Π_f generates $O(1)$ such fragments, and we use a planar point location structure for M_f^t to find, for each fragment that has its floor on f, the cell of H that contains the fragment. Thus, we can, for each $B \in H$, compute the conflict list $\mathrm{CL}(B)$ of B by taking the union of the conflict lists of fragments that make up B. The bottom map M_f^b is handled similarly. The total time for this whole step is $O\big(|\Pi_f| \log n + \sum_{\tau \in \Pi_f} |\,\mathrm{CL}(\tau)|\big)$.

Constructing the new stage-2 prisms. Next, we perform stage 2 of the vertical decomposition within each cell B of H, by constructing the yz-faces that partition it prisms of constant complexity. To do this, we take the floor B_f of B (which is a portion of f), project it onto the xy-plane, and compute the vertical decomposition, denoted $\mathrm{VD}(B_f)$, of B_f using a planar sweep. The y-edges of $\mathrm{VD}(B_f)$, when lifted to three dimensions and intersected with B, define the yz-faces of the desired vertical decomposition of B, which we denote as $\mathrm{VD}(B)$. In this step, we also compute the navigational pointers between adjacent prisms within a stage-2 cell and between the prisms and their incident stage-1 vertices. The cells for the bottom map M_f^b are handled similarly. The total running time for this step is $O\big(|\Pi_f| \log n\big)$.

Prisms for which f defines an x-edge or a yz-face (that passes through a vertex on f). Consider the new prisms of this kind whose bottom faces lie on some $g \in F_i$. To construct these prisms, we draw on g the intersection edges of g with f (which we have already computed). Let e be such an intersection edge. Vertically above g, on one side of e, we have prisms whose ceiling is on f, which we have already computed. On the other side of e, the surface g is the floor of new prisms that are obtained from suitable fragments of old prisms that were cut by f and intersect e. See, e.g., the middle portion of the prism depicted in Figure 9, and see also below.

Thus, we collect all fragments of prisms from Π_f that have their floor on g, draw their projections on g, and trace these projections with a vertical sweep to obtain the old real or shadow edges that appear on these prisms, as done in the previous case. This produces new (partial) stage-1 cells that have their floor on g and for which f only defines an edge or a vertex, together with their conflict lists. We then construct the planar vertical decomposition of their projections onto the xy-plane, and lift each resulting trapezoid into a stage-2 prism, which collectively yield the set of new prisms of this kind. This proceeds very much like the processing in the former case, and we also handle the conflict lists in a similar way. We repeat this process for each side of each surface intersected by f. The total running time is $O\big(|\Pi_f| \log n + \sum_{\tau \in \Pi_f} |\,\mathrm{CL}(\tau)|\big)$; see Figure 10.

Second step: Constructing the conflict lists. We next find, for each new stage-1 cell B, the conflict lists of the new stage-2 prisms $\tau \in \mathrm{VD}(B)$. Assume, without loss of generality, that the floor of B is on f, and let f^+ denote the surface on the ceiling of B. Let B_0 denote the xy-projection of B. Let g be a surface that crosses B, and let $B(g)$ denote the portion of B_0 over which g lies between f and f^+ (that is, g is in B). Clearly, $g \in \mathrm{CL}(\tau)$ precisely for those prisms $\tau \in \mathrm{VD}(B)$ whose xy-projections intersect $B(g)$. Note that it is possible that $B(g) = B_0$, in which case g does not cross f or f^+ at all over B_0. Then g belongs to the conflict lists of all new prisms in $\mathrm{VD}(B)$; see Figure 11.

In general, though, $\gamma := \partial B(g) \cap \mathrm{int}(B_0)$ is a collection of algebraic arcs and closed or unbounded curves, each of which belongs to either $g \cap f$ or $g \cap f^+$. Note that $\gamma \cup \partial B_0$ partitions B_0 into connected regions, over each of which g either 'floats' between f and f^+, or lies below f, or lies above f^+. We need to place g in the conflict list of precisely those prisms whose projections overlap a region of the first kind (the union of these regions is $B(g)$).

Our strategy is to trace γ through $\mathrm{VD}(B_0)$ (which is the xy-projection of $\mathrm{VD}(B)$). Clearly, g belongs to $\mathrm{CL}(\tau)$ for every prism τ whose projection meets γ. We then perform a search over the adjacency graph of the prisms (which connects those pairs of prisms that have overlapping yz-faces), and 'broadcast' g to all the prisms that we reach, placing g in their conflict list. This last step is easy to implement, in total time proportional to the total size of the conflict lists, so

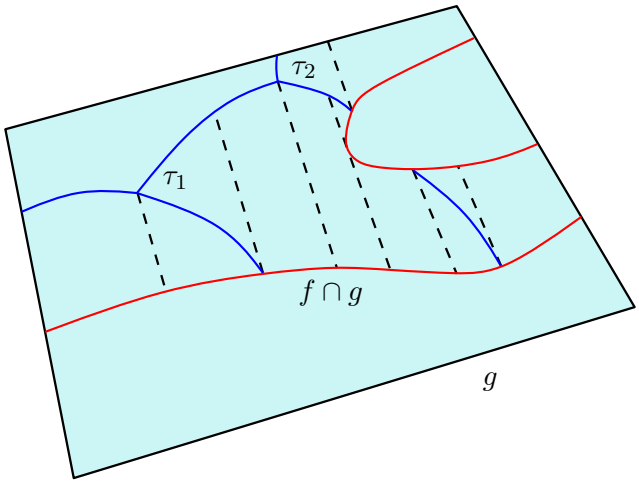

Figure 10: Collecting prisms to which f contributes only an x-edge or a vertex. The figure depicts floors of such prisms on an older surface g. The red arcs depict $f \cap g$, and the blue arcs depict (portions of) older real or shadow arcs. For the prisms τ_1 and τ_2, f only contributes a vertex. For all the other prisms, f contributes an x-edge. f passes below g in the region decomposed into prisms, and above g in the complementary regions, which are decomposed into prisms that have f as a ceiling, and have already been constructed.

we focus on the step of tracing γ.

Consider the task of tracing the portion of γ formed by the xy-projection of $g \cap f$. For simplicity of presentation, continue to refer to it as γ. Each connected component of γ is either a closed (or unbounded) curve, or an arc whose endpoints lie on ∂B_0. We compute a point on each connected component, and then locate, for each point, the trapezoid of $\mathrm{VD}(B_0)$ that contains it. We then trace γ from these points and trapezoids, in both directions, within $\mathrm{VD}(B_0)$, through the adjacency graph of the trapezoids, and place g in the conflict list of each prism whose projection forms a trapezoid that we reach. To obtain a point on each component, we go over all prisms $\tau \in \Pi_f$ for which $g \in \mathrm{CL}(\tau)$, and we check whether g meets f (the floor of B) within τ. If so, we take one point in each component of the intersection $g \cap f \cap \tau$, and use it as a starting point. If no such τ is found, we know that $B(g) = B_0$, and we proceed as above (i.e., place g in the conflict lists of all new prisms). Note that this may generate several starting points along the same component of γ. To avoid tracing the component multiple times, we find, for each point, the new prism that contains it, and mark that prism as already visited (by the current component of γ). In this way, the tracing of the component from some starting point terminates when we reach such a marked prism.

The cost of this procedure is $O\big(\sum_{\tau \in \Pi_f} |\,\mathrm{CL}(\tau)|\log n + \sum_{\tau' \in \Pi'_f} |\,\mathrm{CL}(\tau')|\big)$, where Π'_f is the set of new prisms. The $\log n$-factor comes from the cost of locating the starting points in the new prisms. The number of starting points is at most the total conflict size of the old prisms. We find the conflict lists of the new prisms whose ceiling is on f analogously, using M_f^b instead of M_f^t. The same strategy works also for those new prisms that have f as defining only an edge or a vertex.

Third step: Removal of prisms that are above $\mathcal{A}_{\le t}(F_{i+1})$. Consider an old prism τ whose ceiling was part of $L_t(F_i)$. If f passes fully below τ, the level of τ increases to $t+1$, and τ has to

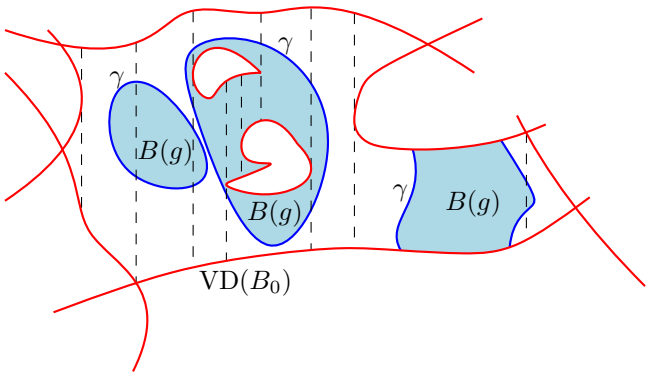

Figure 11: The xy-projection B_0 of a new stage-1 cell B (involving f), and its interaction with a crossing surface g. The shaded region is $B(g)$ and its boundary γ consists of projections of portions where g intersects either the floor f, or the ceiling f^+.

be removed. Similarly, some new prisms may be at level $t+1$, and we also need to remove them. If we could explicitly keep track of the level of each prism, and update these counters after each insertion, the removal of these "overhanging" prisms would be trivial. However, there may be many prisms that lie fully above f, and broadcasting to all of them that their level has increased by 1 is too expensive in general.

Instead, we discover the prisms to be discarded on the fly. There are three cases. The first case occurs when f lies completely above $L_t(F_i)$. Then, we have $\Pi_f = \emptyset$, and there is nothing to do. In the second case, f lies completely below $L_t(F_i)$, and we must discard all prisms whose ceiling touches the top boundary of $\mathcal{A}^{+f}_{\leq t}(F_i)$. This case will be very similar to the next case, so we defer it for now. In the third case, f intersects the t-level $L_t(F_i)$. This intersection manifests itself as a collection of real intersection edges on M^t_f between f and surfaces from F_i, such that each edge is incident on only one side to a stage-1 cell $B \in H$ that has its floor on f (and on the other side, there are no prisms in Π_f that intersect f). We can identify these intersection edges by inspecting all real intersection edges e on M^t_f and by checking whether there are incident cells from H on only one side of e. The resulting intersection edges bound the connected regions on the top boundary of $\mathcal{A}^{+f}_{\leq t}(F_i)$ where f dips below the original t-level $L_t(F_i)$. In each such region, we must remove all prisms whose ceiling touches the top boundary of $\mathcal{A}^{+f}_{\leq t}(F_i)$.

To better understand this process, we note that each stage-1 cell C of the vertical decomposition is such that all prisms in $\mathrm{VD}(C)$ are at the same level of $\mathcal{A}^{+f}_{\leq t}(F_i)$ and are adjacent to each other only through overlapping yz-faces. Hence, if one prism from $\mathrm{VD}(C)$ has its ceiling on the top boundary of $\mathcal{A}^{+f}_{\leq t}(F_i)$, they all do. Furthermore, if we have one such prism in $\mathrm{VD}(C)$, we can find all these prisms in $O(|\,\mathrm{VD}(C)|)$ time, using the navigational pointers between adjacent stage-2 prisms. Thus, we can start the process by identifying all stage-1 cells $B \in H$ on M^t_f whose ceiling touches the top-boundary of $\mathcal{A}^{+f}_{\leq t}(F_i)$, and by collecting the prisms in $\mathrm{VD}(B)$ through a simple traversal. However, this does not suffice, because there could still be stage-1 cells to be removed that are not visible on M^t_f, i.e., stage-1 cells whose floor does not lie on f. These stage-1 cells can be found through a traversal from the stage-1 cells that we have already identified. Recall that we also maintain pointers between the vertices of the stage-1 cells and the prisms that are incident to

them. These cells can be used to pass from one stage-1 cell B to adjacent stage-1 cells in the same stage-0 cell (i.e., stage-1 cells that share a (partial) vertical curtain with B), and also to adjacent stage-1 cells (that share an x-edge where the floor and the ceiling meet) in another stage-0 cell. During the traversal, we mark all prisms that we encounter, to avoid multiple visits to the same prism. Thus, the total running time is proportional to the number of prisms that we discard.

The third case is very similar, only that we do not start the traversal from M_f^t, but from the vertex on $L_t(F_i)$ that that we maintain throughout the algorithm. Clearly, after discarding the superfluous prisms, we can update this vertex with no additional overhead. To conclude, the overall expected running time of the above procedure is proportional to the overall size of all the conflict lists that have been generated during the incremental process plus the overall number of generated prisms times a logarithmic factor.