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

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

April 12, 2016

Abstract

We describe a new data structure for dynamic nearest neighbor queries in the plane with respect to a general family of distance functions that includes L_p-norms and additively weighted Euclidean distances, and for general (convex, pairwise disjoint) sites that have constant description complexity (line segments, disks, etc.). Our data structure has a 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 [3]. Our data structure has numerous applications, and in all of them it gives faster algorithms, typically reducing an $O(n^\varepsilon)$ factor in the bounds to polylogarithmic. To further demonstrate its power, 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 and 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 an appropriate 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). To analyze this algorithm, we improve a longstanding 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 $\mathbb{R}^3$. While doing this, 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.

*Work by Haim Kaplan, Wolfgang Mulzer, Liam Roditty, and Paul Seiferth has been supported by grant 1161/2011 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). Work by Micha Sharir has been supported by Grant 2012/229 from the U.S.-Israel Binational Science Foundation, by Grant 892/13 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.

1 Introduction

Nearest neighbor searching in the plane is one of the most fundamental problems in computational geometry, and it has been studied since the very beginning of the field [8]. Given a set S of *sites* in the plane, we would like to construct a data structure that allows us to find the "closest" site for any given query object. If the set of sites is fixed, Voronoi diagrams and their many variants provide a simple and well-understood solution to this problem [8]. However, in many applications, the sites may change dynamically, while we keep inserting and deleting sites into/from S, and we want to answer nearest-neighbor queries interleaved with the updates. This setting is much less understood.

If S consists of points and distances are measured in the Euclidean metric, it is known how to achieve polylogarihmic update and query time [13]. However, there are many geometric problems in which dynamic nearest neighbor searching forms a key component in an efficient solution, and in these applications it is often crucial to perform nearest neighbor queries for more general distance functions (e.g., L_p-norms or additively weighted Euclidean distances). These applications include the problems of dynamically maintaining a bichromatic closest pair of sites, a minimum-weight Euclidean red-blue matching, a Euclidean minimum spanning tree, the intersection of unit balls in three dimensions, and the smallest stabbing disk of a family of simply shaped compact strictly-convex sets in the plane. Another recent application is an algorithm for computing shortest path trees in unit-disk graphs (see Section 9 for more details and references). Despite the wide range of applications, there has been virtually no progress on dynamic nearest neighbor search in the plane for general distance functions since the last millennium. The state of the art solution dates from 1999 and provides $O(n^{\varepsilon})$ update time with $O(\log n)$ time queries [3]. We describe a new data structure that improves this bound to polylogarithmic update and query time for a wide range of distance functions. For this, we need to bring together a diverse set of techniques such as randomized incremental construction, relative (p, ε)-approximations, shallow cuttings for xy-monotone surfaces in $\mathbb{R}^3$, and data structuring tricks.

We now give a more detailed description of the setting. Let $S \subset \mathbb{R}^2$ be a set of n pairwise disjoint sites, each being a simply-shaped compact convex region in the plane (such as points, line segments, disks, etc.), and let δ be some given continuous distance function between points in the plane. For a site $s \in S$, define $f_s(x, y) = \delta((x, y), s)$, namely, $f_s(x, y) = \min\{\delta((x, y), p) \mid p \in s\}$ (compactness of the objects in S, and continuity of δ, ensure that the minimum exists). We assume that δ and the sites in S have constant description complexity, i.e., they are defined by a constant number of polynomial equations and inequalities of constant maximum degree. Let F denote the collection of the bivariate functions $\{f_s\}_{s \in S}$. The *lower envelope* $\mathcal{E}_F$ of F is the pointwise minimum $\mathcal{E}_F(x, y) = \min_{s \in S} f_s(x, y)$, and its xy-projection is called the *minimization diagram* of F, and is denoted as $\mathcal{M}_F$. The *combinatorial complexity* of $\mathcal{E}_F$ or of $\mathcal{M}_F$ is the number of their vertices, edges and faces. See [42] for a comprehensive treatment of these concepts.

Finding the δ-nearest neighbor in S of a query point $q \in \mathbb{R}^2$ calls for identifying the site s for which $\mathcal{E}_F(q) = f_s(q)$. Such a query translates to a *vertical ray shooting* query in $\mathcal{E}_F$, where we seek the intersection point of the z-vertical line through q with $\mathcal{E}_F$, or, alternatively, we want to locate q in $\mathcal{M}_F$, where each face φ of this planar map is labeled with the site s for which f_s attains the minimum over φ.

The structure and complexity of $\mathcal{E}_F$ and of $\mathcal{M}_F$, as well as algorithms for their construction and manipulation, have been studied extensively for several decades (again, see [42]). Briefly, under the assumptions made above, the combinatorial complexity of $\mathcal{E}_F$, measured in terms of the number of

vertices, edges, and faces of this surface (or of its corresponding minimization diagram) is $O(n^{2+\varepsilon})$, for any $\varepsilon > 0$ (where the constant of proportionality depends on ε). But in many interesting special cases, the most ubiquitous of which is the case where the functions f_s are all linear (i.e., their graphs are non-vertical planes), the complexity of $\mathcal{E}_F$ is only linear in n. The case of planes arises, after some trivial 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 other distance functions, for which the complexity of $\mathcal{E}_F$ remains linear; see, e.g., the recent book by Aurenhammer, Klein, and Lee [5].

Assuming linear complexity of $\mathcal{E}_F$, and the availability of an efficient algorithm for constructing it, all we need to do, in the so-called "static" case, is to preprocess $\mathcal{M}_F$ for fast planar point location, and then locate each query point in $\mathcal{M}_F$, in $O(\log n)$ time.

However, when the sites in S can be inserted or deleted, this corresponds to the setup in which F changes dynamically, by insertions and deletions of functions. The main issue in this situation is that, upon an insertion or a deletion of a function, $\mathcal{E}_F$ might change rather drastically, and maintaining an explicit representation of $\mathcal{M}_F$ after each update might be overwhelmingly expensive. The goal, pursued in this paper, as well as in several earlier works (reviewed below) is to store some implicit representation of $\mathcal{E}_F$ that still supports efficient execution of vertical ray shooting queries into the current envelope (or point location queries in the current $\mathcal{M}_F$).

In all applications of dynamic nearest neighbor searching mentioned above, the lower envelope of the corresponding set F of bivariate functions has linear complexity. The distance functions that arise are typically L_p-metrics, for some $1 \leq p \leq \infty$, or additively weighted Euclidean metrics, where each (say, point) site $s \in S$ has an associated 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., [5,33] for details concerning the linear complexity of the envelope in these cases. This property of having lower envelope of linear complexity also holds for general classes of pairwise-disjoint compact convex sites of *constant description complexity*.

Our main result is an efficient data structure that dynamically maintains a set F of bivariate functions, of the kind assumed above, under insertions and deletions of functions, and supports efficient vertical ray shooting queries into the lower envelope of F. Assuming, as above, that the complexity of the lower envelope is linear, the worst-case cost of a query, as well as the amortized cost of an update, is polylogarithmic in our data structure. Applying this data structure, we obtain faster solutions to all the applications mentioned above, essentially replacing an $O(n^{\varepsilon})$ factor in the complexity of earlier solutions by a polylogarithmic factor.

A brief context. Consider first the case where the graphs of the bivariate functions in F are planes. (As already noted, this case corresponds to the dynamic nearest neighbor problem for a set S of point sites with respect to the Euclidean metric.) A classic solution for this special case is due to Agarwal and Matoušek [4]. They show how to maintain dynamically, in an implicit manner, the lower envelope of a set F of at most n planes, with amortized update time $O(n^{\varepsilon})$, where $\varepsilon > 0$ can be made arbitrarily small (and where the constant of proportionality depends on ε); vertical ray shooting queries take $O(\log n)$ worst-case time. The case of more general bivariate functions, of the sort considered in this paper, was studied by Agarwal et al. [3]. In cases where the complexity of the lower envelope is linear (such as those reviewed above), the technique of Agarwal et al. [3] has amortized update (insertion or deletion) time $O(n^{\varepsilon})$, and worst-case query time $O(\log n)$, matching the known bounds for planes by Agarwal and Matoušek [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, in which both the (amortized) update time and the (worst-case) query time are polylogarithmic, for the case of planes. More precisely, Chan's data structure (combined with the recent deterministic construction of shallow cuttings by Chan and Tsakalidis [15]) 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, so far it has still been unknown whether a similar result (with polylogarithmic update time) is possible for arbitrary bivariate functions of constant description complexity with linear envelope complexity. In this paper we settle this question, by providing an algorithm that meets all these performance goals. Along the way, we also improve the deletion time for Chan's data structure for the case of planes by a logarithmic factor 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.

2 Our results and techniques

As already mentioned in the introduction, our dynamic nearest neighbor structure for general distance functions requires a multitude of techniques that need to be combined carefully. We now first give an broad overview of how these techniques play together, and then we provide a more detailed description on how we use them. See Figure 1 for an illustration, the various concepts that appear in the figure will be explained in the subsequent text.

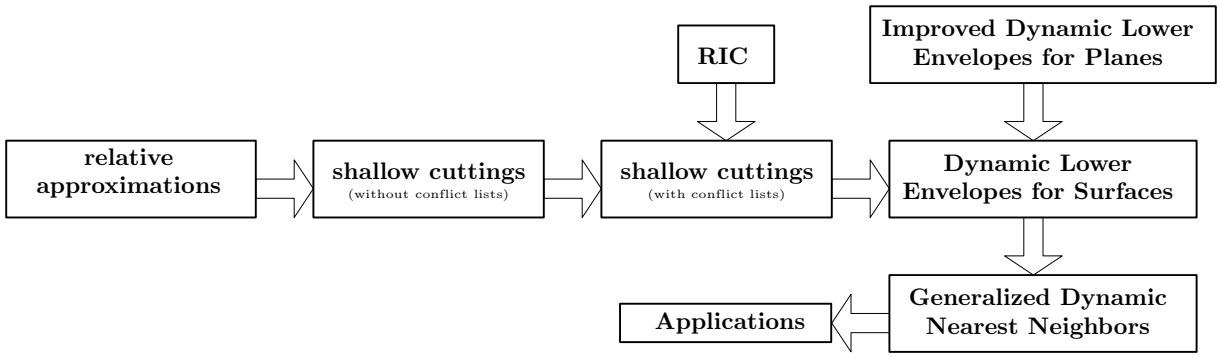

Figure 1: An overview of how our techniques relate to each other.

Maybe the most crucial observation is that all the geometry required in Chan's data structure for dynamic lower envelopes of planes in $\mathbb{R}^3$ lies in the construction of small-sized *shallow cuttings* for planes (of a certain special type). Thus, once we have small-sized shallow cuttings for surfaces, we are able to maintain dynamically the lower envelope of surfaces, or equivalently, solve the generalized dynamic nearest neighbor problem in the plane. It turns out that using relative (p, ε)-approximations, we can find the required cutting quite easily. However, at this point we do not know how to obtain the conflict lists for such a cutting in an efficient manner. To solve this issue, we give an algorithm that is based on randomized incremental construction (RIC) to compute the $(\leq k)$-level in an arrangement of surfaces. This algorithm can be used to efficiently compute the required shallow cutting and its conflict lists. Together with an improved version of Chan's dynamic lower envelope structure, this gives the generalized nearest neighbor data structure. We show the impact of this structure by providing several applications (with new bounds), old and new. In what

follows, we describe the specific parts in more detail.

As mentioned above, the geometric core of Chan's data structure consists of an efficient construction of small-size *shallow cuttings* of a particularly favorable kind, which we refer to as *vertical shallow cuttings* [12, 15]. To define these constructs, we first recall the notion of a *level* in an arrangement of n function graphs in three dimensions. For a parameter $0 \leq k \leq n-1$, the *k-level* in the arrangement is the closure of the set of points q such that q lies on some function graph and exactly k of the graphs pass strictly below q.

Roughly speaking (precise definitions and further discussion are given below), for suitably chosen parameters k and $r \approx n/k$, a vertical k-shallow $(1/r)$-cutting consists of pairwise openly disjoint semi-unbounded vertical prisms, where each prism consists of all points that lie vertically below some triangle, such that (i) these top triangles form a polyhedral terrain that lies above the k-level of the arrangement $\mathcal{A}(F)$, and below a level k' that is close to k, (ii) the number of prisms is close to $O(r)$, and (iii) each of them is crossed by approximately k of the function graphs.

Essentially, once a fast construction of vertical shallow cuttings of sufficiently small size is available, we can combine it with the machinery developed by Chan [13] for the case of planes, in an almost black-box fashion, to obtain a fast data structure for dynamic maintenance of $\mathcal{E}_F$ 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 construction yields an additional $O(n^\varepsilon)$ factor for both the size of the cutting and the time required to construct it. When applied to the dynamic maintenance problem, this makes the (amortized) cost of an update $O(n^\varepsilon)$ rather than polylogarithmic, and refining this bound is one of the main goals of the present paper.

We handle these issues by designing a different algorithm for computing a vertical shallow cutting. To obtain this algorithm, we first prove several technical results that we believe to be of independent interest.

We first use the notion of *relative approximation*, as in Har-Peled and Sharir [28], to conclude that, by choosing a random sample S_k of $\frac{cn}{\varepsilon^2 k}\log n$ of the functions in $\mathcal{F}$, and by constructing level t of $\mathcal{A}(S_k)$ where t is in the range $\left[(1+\frac{\varepsilon}{3})\lambda,\ (1+\frac{\varepsilon}{2})\lambda\right]$, for $\lambda = \frac{c\log n}{\varepsilon^2}$ and c being a suitable absolute constant, we get an *ε-approximation* of level k of $\mathcal{A}(F)$, with high probability. This means that any such level t of $\mathcal{A}(S_k)$ lies between levels k and $(1+\varepsilon)k$ of $\mathcal{A}(F)$. Furthermore, we show that if we choose t uniformly at random in the above range, the expected complexity of the corresponding level is $O\left(\frac{n}{\varepsilon^5 k}\log^2 n\right)$.

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

The last hurdle that we face is how to efficiently compute Λ_k, together with the *conflict lists* of its prisms, where, for a prism τ, the conflict list $\mathrm{CL}(\tau)$ of τ is the set of all functions $f \in F$ that cross the interior of τ. (Note that, although the construction of Λ_k is performed with respect to the sample S_k, the conflict lists of its prisms are defined with respect to the entire collection F.)

For this we consider the classical problem of computing the t shallowest levels in an arrangement of n bivariate functions of constant description complexity, as above. A standard approach to performing this construction is via *randomized incremental construction* (RIC, in short); see,

e.g., [8,37]. In this approach, one adds the functions one by one, in a random order, and maintains some representation of the desired structure (the first t levels in our case) of the subset of functions inserted so far. Following one of the standard ways of performing a RIC, we maintain a decomposition of the region below the t-level of the arrangement of the current subset of the functions, and associate with each of its cells a conflict list, of all the functions not yet inserted that cross that cell. If we run this process to completion, we get a suitable decomposition of the t shallowest levels of the "final" $\mathcal{A}(F)$, but if we stop after inserting the first $\frac{cn}{\varepsilon^2 k}\log n$ functions, which can be regarded as constituting 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 (with respect to the whole F) of its cells.

We note that the decomposition that we use is (a suitable shallow portion of) the standard *vertical decomposition* of an arrangement of surfaces in 3-space (see [18,42] for details). Each prism in this decomposition extends between two consecutive levels of the present arrangement, so the prisms cannot be used to form a vertical shallow cutting. Nevertheless, as we show, it is possible to transform our decomposition into a vertical shallow cutting, including the new conflict lists of its semi-unbounded prisms. The resulting shallow cutting is of expected complexity $O\left(\frac{n}{k}\log^2 n\right)$ (where we use an *ε-approximation* of the k-level of $\mathcal{A}(F)$ for some constant value of ε).

The implementation of such a randomized incremental construction of the shallowest t levels of $\mathcal{A}(F)$ is far from trivial. This problem has been considered before for the case where the function graphs are planes. Mulmuley [36] described a randomized incremental construction 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 runs in $O(nt^2\log(n/t))$ expected time.[1] 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, whose level in $\mathcal{A}(F)$ is known, and 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 extend 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 of a set F of n bivariate functions of constant description complexity with linear envelope complexity. Our algorithm runs in expected $O(nt\lambda_{s+2}(t)\log(n/t)\log n)$ time, where s is a constant that depends on the surfaces and $\lambda_{s+2}(t)$ is the function which bounds the maximum length of a Davenport-Schinzel sequence of order $s+2$ [42].[2] 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 such an arrangement. Agarwal et al. proved that this complexity is $O(nt^{2+\varepsilon})$, for any $\varepsilon > 0$, using a fairly complicated charging scheme, and we improve this to $O(nt\lambda_{s+2}(t))$, using a simpler argument, where s is some constant that depends on the algebraic complexity of the functions of F.

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

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

[2] As is well known [42], the function $\lambda_{s+2}(t)$ is "almost" linear, i.e., $\lambda_{s+2}(t) = t\beta_{s+2}(t)$ for some extremely slow-growing function $\beta_{s+2}(t)$ inverse-Ackermann type.

Once we have available 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 doing so, we re-examine Chan's data structure, for the case of planes, and revise it into a structure that (in our opinion) is easier to understand an at the same time is slightly faster. Our variant follows a standard route: we begin with a static data structure, then extend it to support insertions, using a variant of the well-known Bentley-Saxe binary counter technique [7], and finally show how to perform deletions via re-insertions of planes, using a so-called *deletion lookahead mechanism*, the major innovation in Chan's work. We believe that this new perspective sheds additional light on the inner workings of Chan's structure. Furthermore, we were able to improve the amortized cost of a deletion, by a logarithmic factor, to $O(\log^5 n)$. Deletions are the costliest operations in Chan's technique, and are therefore the bottleneck in most applications.

We finally combine our shallow cutting construction with our improved version of Chan's data structure, to obtain a dynamic data structure for vertical ray shooting into the lower envelope of a dynamically changing set of bivariate functions of the type that we consider. Our (worst-case, deterministic) query time is $O(\log^2 n)$, the (amortized, expected) time for an insertion is $O(\log^5 n\lambda_{s+2}(\log n))$, and the (amortized, expected) time for a deletion is $O(\log^9 n\lambda_{s+2}(\log n))$.

As mentioned, plugging our new bounds into the applications described in Agarwal et al. [3] and in Chan [13] immediately improves numerous running time bounds by a factor of n^ε. Some prominent examples are shown in following table, details can be found in Section 9.

Problem	Old Bound	New Bound
Dynamic bichromatic closest pair in general planar metric	n^ε update [3]	$\log^{10} n\lambda_{s+2}(\log n)$ insertion, $\log^{11} n\lambda_{s+2}(\log n)$ deletion
Minimum Euclidean planar bichromatic matching	$n^{2+\varepsilon}$ [3]	$n^2 \log^{11} n\lambda_{s+2}(\log n)$
Dynamic minimum spanning tree in general planar metric	n^ε update [3]	$\log^{13} n\lambda_{s+2}(\log n)$ update
Dynamic intersection of unit balls in $\mathbb{R}^3$	n^ε update [3] queries in $\log n$ and $\log^4 n$	$\log^5 n\lambda_{s+2}(\log n)$ insertion, $\log^9 n\lambda_{s+2}(\log n)$ deletion, queries in $\log^2 n$ and $\log^5 n$

One 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 $w_p > 0$, $p \in S$. The *disk intersection graph* for S, $D(S)$, has the sites in S as vertices, and there is an edge between two sites pq 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 enjoy an increasing interest in the research community, in particular due to applications in wireless sensor networks [10,14,25,40]. The following table gives an overview of our results.

Problem	Old Bound	New Bound
Shortest path tree in a unit disk graph	$n^{1+\varepsilon}$ [10]	$n \log^{11} n\lambda_{s+2}(\log n)$
Dynamic connectivity in disk intersection graphs	$n^{20/21}$ update $n^{1/7}$ query [14]	$\Psi^2 \log^9 n\lambda_{s+2}(\log n)$ update $\log n/\log\log n$ query
BFS tree in a disk intersection graph	$n^{1+\varepsilon}$ [40]	$n \log^9 n\lambda_{s+2}(\log n)$
$(1+\rho)$-spanner for a disk intersection graph	$n^{4/3+\varepsilon}\rho^{-4/3} \log^{2/3} \Psi$ [25]	$(n/\rho^2) \log^9 n\lambda_{s+2}(\log n)$

Two applications concern finding shortest path trees in unit disk graphs and BFS-trees in disk intersection graphs. Here, our new structures give improved bounds almost in a black-box fashion using the techniques of Cabello and Jejčič [10], and of Roditty and Segal [40]. The other two applications are a bit more involved. The first application consists of a data structure for the dynamic maintenance of the connected components in a disk intersection graph, as disks are inserted or deleted. We assume that all disks that are inserted or deleted have weights that lie in the interval $[1, \Psi]$. Then, we can leverage our data structure in a novel 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 [14] is only slightly sublinear, but independent of Ψ. Finally, we give an algorithm for computing a $(1+\rho)$-spanner in a disk intersection graph, for $\rho > 0$. 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 $(1+\varepsilon)$. The previous construction by Fürer and Kasiviswanathan [25] 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 level k of $\mathcal{A}(F)$ by random sampling, via the notion of *relative* (p, ε)-*approximations* (see [28] and later in this paper). In Section 5, we define a vertical shallow cutting based on our level approximation and show how to compute it via a randomized incremental construction of the shallowest t levels in $\mathcal{A}(F)$. In Section 6, we describe the randomized incremental construction and analyze it. Section 7 gives our variant of Chan's structure. 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 (with linear lower envelope complexity). Finally, in Section 9, we describe a few known applications, for which we obtain better bounds, and our new applications for disk graphs.

3 Preliminaries

Let $\mathcal{F}$ be a family of bivariate functions in $\mathbb{R}^3$, 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*, formally meaning that the graph of each function

is a semialgebraic set, defined by a constant number of polynomial equalities and inequalities of constant maximum degree. The *lower envelope* $\mathcal{E}_F$ of F is the graph of the pointwise minimum of the functions of F. The xy-projection of $\mathcal{E}_F$ is 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) structure (see, e.g., [8]). Each of its faces corresponds to (and is labeled by) the function in F that attains $\mathcal{E}_F$ over that face.

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*, and we assume this to be the case for the families $\mathcal{F}$ that we are going to consider. In particular, this holds when $\mathcal{F}$ is the family of all nonvertical planes, and when $\mathcal{F}$ is a family of distance functions under some metric (or so-called convex distance function), each of which measures the distance of a point in the xy-plane to some given site (cf. Section 9).

For simplicity, we also assume that F is in *general position*, i.e., no more than three function graphs meet at a common point, no more than two function graphs meet in a one-dimensional curve, and no pair of graphs are tangent to each other. (This holds if, say, the coefficients of the polynomials defining the functions in F are algebraically independent over the reals [42].) Furthermore, we assume that the coordinate frame is generic, so that the xy-projections of the intersection curves of pairs of the function graphs 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 point of a triple of function graphs, computing the intersection curve of a pair of graphs, 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., [6,41]), and actual packages (such as the one described in [9]), that perform such operations exactly in constant time. (Technically, these methods and packages determine exactly the truth value of any Boolean predicate of constant description complexity. That is, they are not expected to provide exact 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 of n bivariate functions $F \subseteq \mathcal{F}$ 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 whose graphs pass strictly below q. For $k \in \{0, \ldots, n-1\}$, the *k-level* $L_k(F)$ of $\mathcal{A}(F)$ is the closure of the set of points at level k that lie on the union of the graphs of the functions in F. We denote by $L_{\leq k}(F)$ the union of the first k levels of $\mathcal{A}(F)$. For given parameters k, $r < n$, 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 these regions covers $L_{\leq k}(F)$, and so that the interior of each region $\tau \in \Lambda$ is intersected by at most n/r graphs of functions of F. The *size* of Λ is the number of regions in Λ.

A *vertical k-shallow $(1/r)$-cutting* in $\mathcal{A}(F)$ is a collection Λ of pairwise openly disjoint vertical semi-unbounded *pseudo-prisms*, a notion to be defined momentarily, so that, as above, the union of these pseudo-prisms covers $L_{\leq k}(F)$, and so that the interior of each pseudo-prism $\tau \in \Lambda$ is intersected by at most n/r graphs of functions of F. Note that, for both conditions to hold simultaneously, we must have $k \leq n/r$. In our setting, we will always have $r = \Theta(n/k)$, which is

the case most relevant for applications.

A pseudo-prism τ consists of all points that lie vertically below some portion $\overline{\tau}$ of a graph of a function in F, so that $\overline{\tau}$ has constant description complexity. In our application, $\overline{\tau}$ will be a *pseudo-trapezoid*, defined as the portion of a function graph consisting of points (x, y, z) satisfying $x^- \leq x \leq x^+$, $\psi^-(x) \leq y \leq \psi^+(x)$, for real numbers $x^- < x^+$ and for (semi-)algebraic functions ψ^-, ψ^+ of constant description complexity. In the case of planes, $\overline{\tau}$ will simply be a triangle, and we do not insist that $\overline{\tau}$ be contained in one of the input planes.

In his seminal paper on reporting points in halfspaces [35], Matoušek proved the existence of a k-shallow $(1/r)$-cutting, for n hyperplanes in $\mathbb{R}^d$, whose size is $O(q^{\lceil d/2 \rceil} r^{\lfloor d/2 \rfloor})$, where $q = k(r/n)+1$. For the interesting special case where $k = \Theta(n/r)$, we have $q = O(1)$ and the size of the cutting is $O\left(r^{\lfloor d/2 \rfloor}\right)$, a significant improvement over the general bound $O(r^d)$ for a cutting that covers the whole arrangement (rather than just $L_{\leq k}(F)$) [16]. For example, in three dimensions, we get (for arrangements of planes) $O(r)$ simplices, instead of $O(r^3)$ simplices for the whole arrangement. This has led to improved solutions of many range searching and related problems (see, e.g., [15] and the references therein).

Matoušek [35] presented a deterministic algorithm to construct a shallow cutting in polynomial time; the running time improves to $O(n \log r)$, for $r < n^\delta$ for a sufficiently small constant δ (depending on the dimension d). Later, Ramos [39] 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 [15] provided 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, their analysis 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 show the existence of vertical shallow cuttings for planes in three dimensions. Such a cutting is associated with a polyhedral triangulated xy-monotone terrain, which lies entirely above the k-level of the arrangement, so that each triangle $\overline{\tau}$ of the terrain generates a semi-unbounded triangular prism with $\overline{\tau}$ as its top face. Such shallow cuttings have many applications, in particular in Chan's dynamic lower envelope data structure [13]. The deterministic construction of Chan and Tsakalidis [15] constructs vertical shallow cuttings. Recently, Har-Peled, Kaplan, and Sharir [27] gave an alternative construction with some additional favorable properties.[3]

Things become technically more involved when we consider cuttings in arrangements of algebraic functions in dimension three and higher. Decomposing cells of the arrangement into subcells of constant description complexity is easy for hyperplanes (where the subcells are simplices), using, e.g., the *bottom-vertex triangulation* [17,20]. For general curves or surfaces, the only known general-purpose cell decomposition technique is *vertical decomposition* [18,42]. In the plane, the complexity of such a decomposition is proportional to the complexity of the undecomposed arrangement, and in three and four dimensions, near (but not quite) optimal upper bounds are known [18,32], but in dimension five and higher, the known upper bounds are significantly larger [18]. 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.

[3] One significant difference is that the "top terrain" in [27] approximates the corresponding level k up to any specified accuracy, whereas the structure in [15] does not.

4 Approximate k-levels

In this section and the next, we show how to obtain shallow cuttings for surfaces via random sampling; we address the issue of how to efficiently compute the cuttings and their conflict lists in a later section.

Let $\mathcal{F}$ be a family of continuous, totally-defined bivariate algebraic functions of constant description complexity, and let F be a collection of n functions from $\mathcal{F}$. In what follows we will generally make no distinction between a function $f \in F$ and its graph $z = f(x, y)$ in $\mathbb{R}^3$, which is a continuous (xy-monotone) surface, called a *terrain*, with constant description complexity. Recall that we also assume that the maximum lower envelope complexity $\psi(m)$ of any m functions of $\mathcal{F}$ is $O(m)$.

Agarwal et al. [3] provide a shallow cutting for this setup. For general values of k and r, the bound in [3] on the size of a k-shallow $(1/r)$-cutting for $\mathcal{A}(F)$ is $O(q^{3+\varepsilon}\psi(r/q))$, where $q = kr/n+1$, which is slightly sub-optimal when q is large. However, we are interested in the special case $r \approx n/k$ and $\psi(r) = O(r)$, so $q = O(1)$ and the bound becomes $O(r)$, which is optimal. Nonetheless, the cutting in [3] is not "vertical", that is, it is not composed of vertical pseudo-prisms, and is therefore useless for our approach to the construction of level approximations and for the dynamic maintenance of lower envelopes.

Techniques for computing vertical shallow cuttings for planes, and the conflict lists associated with their prisms [15,27] heavily rely on the fact that if a plane intersects a semi-unbounded prism τ it must intersect a vertical edge of τ. This does not necessarily hold for general functions, and we therefore need to use a somewhat different approach, that results in cuttings of slightly suboptimal size, but only by (small) polylogarithmic factors. It is an interesting challenge to tighten the bound, bringing it down to optimal. For the time being, though, we are not aware of any alternative to the cuttings constructed here, for the specific applications in this paper.

Let $0 < \varepsilon \leq 1/2$ be a specified error parameter.[4] We next present a technique for approximating a level $L_k(F)$ of $\mathcal{A}(F)$ by a terrain $\overline{T}_k$ (which will actually be a level in an arrangement of some subsample of F), with the following properties.

1. $\overline{T}_k$ fully lies above $L_k(F)$ and below $L_{(1+\varepsilon)k}(F)$ of $\mathcal{A}(F)$.
2. The complexity $|\overline{T}_k|$ (that is, number of vertices, edges, and faces) of $\overline{T}_k$ is $O\left(\frac{n \log^2 n}{\varepsilon^5 k}\right)$.

Relative (p, ε)-approximation. We construct $\overline{T}_k$ via the notion of *relative (p, ε)-approximation* (see Har-Peled and Sharir [28] for more details): For a range space $(X, \mathcal{R})$ of finite VC-dimension, and for given parameters $0 < p, \varepsilon < 1$, a set $A \subseteq X$ is called a *relative (p, ε)-approximation*, if, for each range $R \in \mathcal{R}$, we have

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

[4] If the ε that we use is not too close to 0 (say, $\varepsilon = 1/2$), the dependence of the bounds derived below on ε can be suppressed. We include it, though, in the interest of precision, and in the hope that these bounds might be useful for future applications that would require closer level approximations.

As shown in [28] (following Li et al. [34], see also Har-Peled's book [26]), a random sample of size

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

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

We apply this general machinery to the range space $(\mathcal{F},\mathcal{R})$ defined as follows. Each range $R\in\mathcal{R}$ is the set of functions of $\mathcal{F}$ whose graphs intersect an object o, which is either a straight line, segment, or ray, or an edge in the arrangement of the graphs of a constant number of functions of $\mathcal{F}$, or a face in such an arrangement, or 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. The fact that $(\mathcal{F},\mathcal{R})$ has finite VC-dimension, for a collection $\mathcal{F}$ of bivariate algebraic functions of constant description complexity, follows by standard arguments (see, e.g., [42]).

Fix a parameter $k<n$, and let $S_k\subseteq F$ be a random sample of size

$$r_k=|S_k|=\frac{c_0 n}{\varepsilon^2 k}\left(\log\frac{n}{k}+\log\frac{1}{q}\right),$$

where c_0 is a suitable constant (that depends on the VC-dimension of $(\mathcal{F},\mathcal{R})$, but is independent of ε), and $q=1/n^b$, for some sufficiently large constant exponent b. By what has just been noted, with an appropriate choice of c, S_k is a relative $\left(\frac{k}{2n},\frac{\varepsilon}{3}\right)$-approximation for $(\mathcal{F},\mathcal{R})$, with probability at least $1-q$. With the above choice of q, we can simplify the expression, and put

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

for another suitable absolute constant $c>0$. Note that for this choice of r_k to make sense, k has to be $\Omega\left(\frac{1}{\varepsilon^2}\log n\right)$. The case of smaller k is simpler, and will be treated below.

Set $\overline{T}_k$ to be a *random* level in $\mathcal{A}(S_k)$, of index t chosen uniformly at random in the range $\left[(1+\frac{\varepsilon}{3})\lambda,\ (1+\frac{\varepsilon}{2})\lambda\right]$, for $\lambda=\frac{c}{\varepsilon^2}\log n$. We refer to $\overline{T}_k$ as an *ε-approximation* to level $L_k(F)$ in $\mathcal{A}(F)$, and justify this terminology in the following lemmas.

In the lemmas, we assume, without stating it explicitly, that the random sample S_k is indeed a relative $\left(\frac{k}{2n},\frac{\varepsilon}{3}\right)$-approximation. Nevertheless, the bounds in Lemmas 4.2 and 4.3 also hold without this assumption, since S_k fails to be a relative $\left(\frac{k}{2n},\frac{\varepsilon}{3}\right)$-approximation with probability at most $1/n^b$, which is polynomially small in n.

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

Proof. Let p be a point of level k of $\mathcal{A}(F)$, and let $R^{(p)}$ denote the range of those functions that pass below p, i.e., that cross the downward-directed vertical ray emanating from p. By assumption, S_k is a relative $\left(\frac{k}{2n},\frac{\varepsilon}{3}\right)$-approximation, for a range space that includes $R^{(p)}$. Since

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

we can conclude (using the first case in (1)) that

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

That is, at most $\left(1+\frac{\varepsilon}{3}\right)\lambda$ functions of S_k lie or on below p, or, in other words, p must lie on or below $\overline{T}_k$. Similarly, let p be a point of level $(1+\varepsilon)k$ of $\mathcal{A}(F)$. By a symmetric argument, using the fact that $\varepsilon \leq 1/2$, at least $\left(1-\frac{\varepsilon}{3}\right)(1+\varepsilon)\frac{k}{n}r_k \geq \left(1+\frac{\varepsilon}{2}\right)\lambda$ planes of S_k cross the vertical ray emanating downwards from p. Hence p must lie on or above $\overline{T}_k$, and the lemma then follows. □

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

Proof. As argued in the proof of Lemma 4.1, since S_k is a relative approximation, only vertices p of $\mathcal{A}(F)$ whose level at this arrangement lies between k and $(1+\varepsilon)k$ can satisfy $\left(1+\frac{\varepsilon}{3}\right)\lambda \leq \ell_{S_k}(p) \leq \left(1+\frac{\varepsilon}{2}\right)\lambda$. The probability of any vertex of $\mathcal{A}(F)$ to show up in $\mathcal{A}(S_k)$ is[5]

$$\frac{\binom{n-3}{r_k-3}}{\binom{n}{r_k}} \approx \left(\frac{r_k}{n}\right)^3 = O\left(\frac{1}{\varepsilon^6 k^3}\log^3 n\right).$$

As shown in Clarkson and Shor [21] and noted in the introduction, the number of vertices of $\mathcal{A}(F)$ at level at most $(1+\varepsilon)k$ is $O(n((1+\varepsilon)k)^2) = O(nk^2)$. (This bound holds because $\psi(m) = O(m)$; in general, the bound is $O(k^3\psi(n/k))$.) Hence, the expected number of vertices p of $\mathcal{A}(S_k)$ with $\left(1+\frac{\varepsilon}{2}\right)\lambda \leq \ell_{S_k}(p) \leq \left(1+\frac{\varepsilon}{2}\right)\lambda$ is at most

$$O\left(\frac{n}{\varepsilon^6 k}\log^3 n\right), \tag{2}$$

as claimed. □

Lemma 4.3. *The expected complexity of $\overline{T}_k$ is*

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

Proof. The bound in (2) is proportional to the sum of the complexities of all the j-levels of $\mathcal{A}(S_k)$, for $j \in \left[\left(1+\frac{\varepsilon}{3}\right)\lambda,\ \left(1+\frac{\varepsilon}{2}\right)\lambda\right]$. Hence, the expected complexity of a random level t among them is

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

as claimed. □

Finally, we show how to handle the case when k is small. If $k < (1/\varepsilon^2)c\log n$, we pick an integer t randomly in the interval $[k,(1+\varepsilon)k]$. Let $\overline{T}_k$ be the t-level in the arrangement $\mathcal{A}(F)$ of all surfaces. By definition $\overline{T}_k$ approximates the k-level in $\mathcal{A}(F)$. Furthermore, the same Clarkson-Shor based analysis as used in Lemma 4.2 and Lemma 4.3 shows that $\overline{T}_k$ has expected complexity $O(\frac{nk}{\varepsilon}) = O(\frac{n\log^2 n}{\varepsilon^5 k})$, for k small enough.

[5]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

Informally, we would like to turn the approximate level $\overline{T}_k$ into a shallow cutting of the first k levels of $\mathcal{A}(F)$, so that each face $\overline{\varphi}$ of $\overline{T}_k$ becomes a semi-unbounded vertical pseudo-prism φ, consisting of all the points that lie vertically below $\overline{\varphi}$. In what follows, we simplify the notation, and refer to these pseudo-prisms simply as *prisms*. We denote by T_k the collection of prisms obtained in this way. The only technical obstacle is that the faces $\overline{\varphi}$ need not have constant complexity, which might cause the corresponding prisms to be crossed by too many function graphs.

To overcome this issue, we decompose each such face $\overline{\varphi}$ into sub-faces of constant complexity, using two-dimensional vertical decomposition. That is, we project each face $\overline{\varphi}$ onto the xy-plane, and decompose the resulting projection $\overline{\varphi}^*$ into y-vertical pseudo-trapezoids by erecting y-vertical segments (or rays) from each vertex of $\overline{\varphi}^*$ and from each point of vertical tangency on its boundary, extending them until they hit another edge of $\overline{\varphi}^*$ (or all the way to $\pm\infty$ if they don't). The number of pseudo-trapezoids is proportional to the complexity of $\overline{\varphi}$. We then lift each resulting pseudo-trapezoid τ^* into a vertical semi-unbounded pseudo-prism τ (called, as above, a *prism*), consisting of all the points that lie vertically below $\overline{\varphi}$ and project to points in τ^*.[6] Our cutting Λ_k is the collection of all these prisms τ. We also denote by $\overline{\Lambda}_k$ the terrain formed by the ceilings $\overline{\tau}$ of the prisms $\tau \in \Lambda_k$. Note that $\overline{\Lambda}_k$ is a refinement of $\overline{T}_k$. As we will shortly show, Λ_k is indeed a shallow cutting of the first k levels of $\mathcal{A}(F)$.

For each prism $\varphi \in T_k$ and $\tau \in \Lambda_k$, its *conflict list*, denoted as $\mathrm{CL}(\varphi)$ and $\mathrm{CL}(\tau)$, respectively, is the set of functions of F whose graphs intersect φ or 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 τ be a prism of Λ_k. By Lemma 4.1, for each point p on the ceiling $\overline{\tau}$ of τ, the level $\ell_F(p)$ of p in $\mathcal{A}(F)$ is in the range $[k,(1+\varepsilon)k]$. In particular, the level of each vertex of $\overline{\tau}$ is in $[k,(1+\varepsilon)k]$. Furthermore, since $\overline{\tau}$ does not intersect the graph of any function of S_k (which is a relative $\left(\frac{k}{2n},\frac{\varepsilon}{3}\right)$-approximation for $(\mathcal{F},\mathcal{R})$, and $\overline{\tau}$ induces a range in $\mathcal{R}$), it cannot intersect more than

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

graphs of functions of F (here the second bound in (1) applies). For any function $f \in F$ whose graph crosses τ, the graph either passes below all vertices of τ (and there are at most $(1+\varepsilon)k$ such functions), or crosses $\overline{\tau}$. Altogether,

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

The construction of Λ_k ensures that the number of its prisms is proportional to the complexity of Λ_k, so, using Lemma 4.3, it satisfies (in expectation) the bound asserted in the lemma. □

[6] A significant difference between the machinery used here and that for the case of planes, as in [27], 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.

6 Randomized Incremental Construction of the $\leq t$ Level

In this section, we present a randomized incremental construction of the first t levels in an arrangement of n xy-monotone surfaces of constant description complexity in $\mathbb{R}^3$, for which the complexity of the lower envelope of any m of them is $O(m)$. The expected running time of the algorithm is $O(nt\lambda_{s+2}(t)\log(n/t)\log n)$, and the expected storage is $O(nt\lambda_{s+2}(t))$, where s is a constant that depends on the complexity of the surfaces and $\lambda_{s+2}(t)$ is the familiar Davenport-Schinzel bound [42]. We will later use it to compute the shallow cutting as described in Section 5 together with its conflict lists.

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 $\mathcal{F}$. Our goal is to construct the first t levels of the arrangement $\mathcal{A}(F)$. For doing this, we follow the standard technique of *randomized incremental construction* (RIC in short), in which we insert the surfaces of F one at a time, in a random order, and maintain, after each insertion, the first t levels in the arrangement of the functions inserted so far (t is kept fixed during the process). Enumerate the functions of F in the random insertion order as $f_1, f_2, \ldots, f_n$, and let $F_i = \{f_1, \ldots, f_i\}$, for $i = 1, \ldots, n$.

As is standard in this approach, the algorithm maintains a decomposition of $L_{\leq t}(F_i)$ into cells of constant description complexity (see below for details on the decomposition that we use), and maintains, for each cell τ, its *conflict list*, which is the set of all functions not yet inserted that cross τ. When the next function f_{i+1} is inserted, it retrieves right away, using the conflict lists, all the cells that it crosses. These cells are "destroyed" by f_{i+1} (that is, they are no longer valid cells of the new decomposition), and are partitioned by f_{i+1} into subcells. These subcells, though, are also not necessarily valid cells of the new decomposition of $L_{\leq t}(F_{i+1})$, and need to be merged and refined into the correct new cells. In addition, one has to construct the conflict lists of the new cells, which are composed from the elements in the conflict lists of the destroyed cells.

6.1 Computing the First t Levels

We take a random permutation of F, which we assume to be $f_1, f_2, \ldots, f_n$, and denote by F_i the prefix of the first i elements, for $i = 0, 1, \ldots, n$.

The structure that we maintain after each insertion is the vertical decomposition of $L_{\leq t} = L_{\leq t}(F_i)$ of the first t levels of $\mathcal{A}(F_i)$. This vertical decomposition is denoted by $\mathrm{VD}_{\leq t}$ and is defined in the following standard manner.

We obtain $\mathrm{VD}_{\leq t}$ in two decomposition stages. In the first stage, we erect within each cell C of $L_{\leq t}$ a vertical curtain up and/or down from each edge (an intersection edge of a pair of surfaces) of $L_{\leq t}$. Each such wall consists of maximal vertical segments contained in (the closure of) C and passing through the points of the edge. The collection of these walls partitions C into subcells, each having the property that it has a unique "top" facet (referred to as its *ceiling*) and a unique "bottom" facet (referred to as its *floor*); one or both of these facets may be undefined when the subcell is unbounded, and all other facets of the subcell lie on the vertical walls. However, the complexity of each subcell may still be arbitrarily large. Thus, in the second decomposition stage, we take each subcell C', project it onto the xy-plane, and apply to the projection a similar but two-dimensional vertical decomposition: we erect a y-vertical segment from each vertex of the projected subcell and from each point of local x-extrema on its edges. This yields a collection of trapezoidal-like subcells, and we then lift each of them vertically to 3-space; formally, we take each trapezoid τ and form the intersection $(\tau \times \mathbb{R}) \cap C'$. This yields a decomposition of C' into prism-like subcells,

each having "constant description complexity," meaning that each of them is a semialgebraic set defined by a constant number of polynomials of constant maximum degree (which depends on the maximum degree of the surfaces in $\mathcal{F}$). Repeating this second stage for all subcells C' produced in the first stage, and all the cells C of $L_{\leq t}$, we obtain the desired vertical decomposition of $L_{\leq t}$. More details can be found in [18, 42].

As is well known, the complexity of $\mathrm{VD}_{\leq t}$ is proportional to the number of pairs (e, e') of edges of $L_{\leq t}$, such that (i) the xy-projections of e and e' cross one another, at some point q, and (ii) the z-vertical line through q meets e and e' at two respective points w and w', such that the segment ww' does not cross any other surface of F_i. We refer to such a pair (e, e') as a *vertically visible* pair of edges, and assume that the pair is ordered, and that e lies above e' (i.e., w lies above w').

The following crucial lemma improves an earlier bound of $O(nt^{2+\varepsilon})$ by Agarwal et al. [3]. The parameter s in the lemma is defined as follows. For any quadruple f_1, f_2, f_3, f_4 of functions of F, we let $s(f_1, f_2, f_3, f_4)$ denote the number of co-vertical pairs of points $q \in f_1 \cap f_2$, $q' \in f_3 \cap f_4$. We define s to be the maximum value of $s(f_1, f_2, f_3, f_4)$, over all quadruples f_1, f_2, f_3, f_4 of functions of F. By our assumptions on $\mathcal{F}$ (including the one on general position), we have $s = O(1)$. The function $\lambda_{s+2}(t)$ in the lemma is the familiar bound on the maximum length of a Davenport-Schinzel sequence of order $s + 2$ [42].

Lemma 6.1. *Let F be a set of n functions of $\mathcal{F}$, and let $t \leq n$ be a parameter. The complexity of $\mathrm{VD}_{\leq t}(F)$ is $O(nt\lambda_{s+2}(t))$.*

Proof. Let (e, e') be a vertically visible pair of edges of $\mathcal{A}(F)$ within $L_{\leq t}(F)$. By definition, e is not crossed by any function of F, and the overall number of functions whose graphs appear below e is at most t. (Clearly, if a function f appears below some point of e, it must appear below every point of e.) Hence, the complexity of the upper envelope of these functions, clipped to the vertical curtain V_e erected downward from e, is at most $\lambda_s(t)$. Indeed, using a suitable parametrization of e, the cross-sections of these functions within the curtain are totally defined univariate continuous functions, each pair of which intersect at most s times. The latter property follows from the definition of s, by noting that each vertex of the envelope is an intersection point of some edge e' of $L_{\leq t}(F)$ with V_e, that forms with e a vertically visible pair (e, e'), and vice versa.

A standard application of the Clarkson-Shor technique implies that the number of edges of $L_{\leq t}(F)$ is $O(nt^2)$. This follows by charging the edges to their endpoints, and by using the fact that the number of vertices in the lower envelope of any m functions of F is $O(m)$. The analysis so far already gives the (weak) bound of $O(nt^2\lambda_s(t)) \approx nt^3$ on the complexity of $\mathrm{VD}_{\leq t}(F)$.

The arguments so far repeat more or less those given in the initial part of the analysis of [3], and we are now going to replace the rest of the analysis in [3] by the following sharper one.

Fix a pair of functions $f, f' \in F$ and let γ denote their intersection curve. We first cut γ at each of its singular points and locally x-extremal points, into $O(1)$ connected pieces, each of which is an x-monotone Jordan arc. (In addition to our assumption of general position of the surfaces in F, we also assume that the xy-frame is generic, to ensure that none of the curves γ contains any arc that lies within some yz-parallel plane.)

We cut these arcs further at their intersections with $L_t(F)$, and keep the portions of these arcs 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, as mentioned above, the overall complexity of $L_{\leq 2t}(F)$ is $O(nt^2)$, the average complexity of a level between t and $2t$ is indeed $O(nt)$, so there exists a level t' with the above properties. In the rest

of the proof, we will establish the asserted upper bound for $\mathrm{VD}_{\leq t'}(F)$, which will clearly serve as an upper bound for the quantity we are after. To keep the notation simple, we continue to denote the top level t' as t.

Fix a Jordan subarc of some intersection curve that lies in $L_{\leq t}(F)$ (now with the new, potentially larger index t). Any such subarc, if not fully contained strictly below $L_t(F)$, ends in at least one vertex of $L_t(F)$, implying that the number of subarcs that reach the t-level is $O(nt)$. Any other arc is charged either to one of its endpoints, or, if it is unbounded and has no endpoints, to its intersection with the plane at infinity, say $x = +\infty$.

Any arc that reaches the plane at infinity appears there as a vertex of the first t levels of the cross-sections of the functions of F with that plane. An application of the Clarkson-Shor technique to this planar arrangement implies that the number of these vertices is $O(nt)$, so this also bounds the number of arcs of this particular kind.

Finally, the number of arcs with an endpoint that is either singular or locally x-extremal is bounded by charging each such arc to its respective (singular or x-extremal) endpoint, and by bounding the number of these points (all lying in $L_{\leq t}(F)$) 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 $O(nt)$ on their number.

To recap, the overall number of Jordan arcs that we have constructed is $O(nt)$. Let Γ denote their collection.

Fix an arc $\gamma \in \Gamma$. In general, γ will consist of more than one edge of $\mathcal{A}(F)$, and we denote their number by $\mu(\gamma)$. We decompose γ into $\xi(\gamma) := \lceil \mu(\gamma)/t \rceil$ pieces, each consisting of at most t consecutive edges. Again, assuming general position, if e_1 and e_2 are consecutive edges along (a piece of) γ, the set of functions of F whose graphs appear below e, and the set of those whose graphs appear below e' differ by exactly one function (the one incident to the common vertex of e and e'), which appears below e but not below e', or vice versa. This implies that, for a piece δ of γ, the overall number of functions whose graphs appear below δ is at most $2t$. Note that some of these functions are now only partially defined. Arguing as above, the number of vertically visible pairs of edges whose top edge is part of δ is at most $\lambda_{s+2}(2t) = O(\lambda_{s+2}(t))$. Hence, the overall number of vertically visible pairs of edges in $L_{\leq t}(F)$ is

$$\left(\sum_{\gamma\in\Gamma} \xi(\gamma)\right) \cdot O(\lambda_{s+2}(t)) \leq \left(\sum_{\gamma\in\Gamma} \left(\frac{\mu(\gamma)}{t} + 1\right)\right) \cdot O(\lambda_{s+2}(t)) = \left(\frac{1}{t}\sum_{\gamma\in\Gamma} \mu(\gamma) + |\Gamma|\right) \cdot O(\lambda_{s+2}(t)).$$

We have already argued that $|\Gamma| = O(nt)$. The sum $\sum_{\gamma\in\Gamma} \mu(\gamma)$ is simply the number of edges in $L_{\leq t}(F)$ which, as already argued, is $O(nt^2)$. In other words, the number of vertically visible pairs of edges in $L_{\leq t}(F)$ is $O\left(nt\lambda_{s+2}(t)\right)$, as asserted. □

Remark. The same analysis yields a bound on the complexity of $\mathrm{VD}_{\leq t}(F)$ also in cases where the complexity of the lower envelope is not necessarily linear.

Each three-dimensional cell of $\mathrm{VD}_{\leq t}(F_i)$ is a *pseudo-prism*, with up to six faces. Concretely, we have already used "floor" and "ceiling" to refer, respectively, to the bottom and top faces of τ. The *y-forward* (resp., *y-backward*) faces are the z-vertical "curtains" that bound τ in the respective positive and negative y-directions, and are each erected from an edge of τ which is a portion of an intersection curve between the surface supporting the floor or ceiling of τ with another surface. We will sometimes refer to these curtains shortly as the forward and backward faces of τ. The last

two faces of τ are portions of planes orthogonal to the x-axis; they are the *left* and *right* faces of τ. When we do not care whether a face of the second kind is forward or backward, we will refer to it as a *y-face*. Similarly, we will refer to the left and right faces collectively as *x-faces*; see Figure 2 for an illustration. The notation also carries over to edges of τ. There are three kinds of edges: (i) A *y-edge*, which is the common edge of the floor or ceiling and a y-face of τ. It is either (a portion of) a "real" intersection curve, or a *shadow edge*, which lies vertically below or above a real intersection edge on the other (floor or ceiling) side of τ. (ii) An *x-edge*, which is the common edge of the floor or ceiling and an x-face of τ. (iii) A straight z-parallel segment, which is a common edge of an x-face and a y-face.

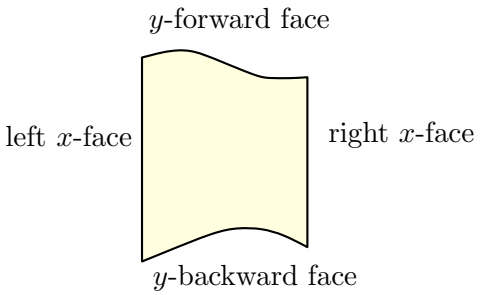

Figure 2: A top view of a prism τ and its y-faces and x-faces.

6.1.1 Inserting a Surface

Consider the step when we insert 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 of its prisms τ 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 perform this task in two steps. First we obtain the vertical decomposition of the part of the arrangement $\mathcal{A}(F_{i+1})$ which lies below the t level of $\mathcal{A}(F_i)$. We denote this arrangement by $\mathcal{A}^{+f}_{\leq t}(F_i)$. This vertical decomposition contains some prisms which are above $\mathcal{A}_{\leq t}(F_{i+1})$, and we discard them in the second step.

When f is inserted, we retrieve all the "old" prisms of $\mathrm{VD}_{\leq t}(F_i)$ that f crosses. Note that any old prism τ that f does not cross remains a valid prism in the vertical decomposition of $\mathcal{A}^{+f}_{\leq t}(F_i)$, but, in case f passes fully below it, its level in the new arrangement (compared to its level $\mathcal{A}(F_i)$) is increased by 1. Hence, it may find itself above the t-level, in which case we will discard it in the second stage. On the other hand, any old prism τ that f does cross is destroyed. It is split by f into fragments, some of which may climb up to level $t+1$ (and have to be discarded), while others stay within the first t levels, but in general they need not be valid prisms of $\mathcal{A}^{+f}_{\leq t}(F_i)$.

For the subsequent steps of the algorithm, we take these fragments, and construct their vertical decomposition within τ. This takes $O(1)$ time. In addition, we compute the conflict list of each of the refined prisms of this decomposition, in brute force, by inspecting each function $g \in \mathrm{CL}(\tau)$ and selecting those that intersect the sub-prism.

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), or a y-edge (where f intersects the floor or ceiling of τ at a boundary edge), 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 edge on f, which defines an x-face of τ; see Figure 3).

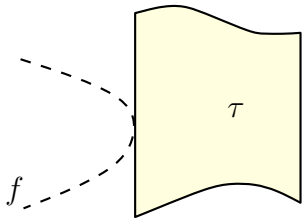

Figure 3: An x-face of a prism τ (seen here as the left side of the view) formed by a locally x-extremal point of some intersection edge with f (top view).

New prisms with a face on f. To construct these prisms, we begin by constructing and tracing the y-edges of $\mathrm{VD}(\mathcal{A}^{+f}_{\leq t}(F_i))$ along f. More precisely, we collect the edges along f of the first stage of the vertical decomposition of $\mathcal{A}^{+f}_{\leq t}(F_i)$. Following the terminology just introduced, each of these edges is either a real intersection edge between f and an older surface, or a shadow edge, namely the vertical projection of the portions of some real intersection edge that are vertically visible from f. The real intersection edges are drawn, naturally, on both sides of f, but each shadow edge is drawn only on one side (top or bottom) of f. Therefore, we obtain two different maps on f, M^t_f and M^b_f, corresponding to the top and bottom sides of f, respectively.

We start by identifying the real intersection edges along f. For each of the old prisms τ that f crosses (which we have already retrieved), we check whether f intersects the floor and/or the ceiling of τ. Each such intersection consists of $O(1)$ connected subarcs, which are portions of the real edges that we are after. When such an edge e leaves a prism τ (at an endpoint of some intersection subarc), it can do so either through an x-edge or through a y-edge. In the former case (crossing an x-edge), we have to glue e to a suitable portion of its continuation into the appropriate old prism adjacent to τ, whereas in the latter case (crossing a y-edge) the crossing point v is either a real vertex of $\mathcal{A}^{+f}_{\leq t}(F_i)$ on e, or part of a vertically visible pair in $\mathcal{A}^{+f}_{\leq t}(F_i)$ consisting of e and the real intersection edge of τ on its other side (floor or ceiling). In the former case, v delimits two subedges of e, one within τ (locally near v) and one entering an adjacent prism, and it is a feature of both M^t_f and M^b_f. In the latter case v is a feature only in one of the maps M^t_f or M^b_f. See Figure 4 for an illustration.

When this process terminates, over all old prisms that f crosses, we have constructed the edges of the first stage of the new vertical decomposition along f that are contained in real intersection edges (between f and older surfaces).

In the next stage, we construct the shadow edges along f. Recall that these are vertical projections onto f of portions of intersection edges (between two surfaces in F_i) that lie above or below f and are vertically visible from f (at the point where their projections cross). Shadow edges that are obtained by projecting real edges onto f from above are drawn on M^t_f and shadow edges that are obtained by projecting real edges onto f from below are drawn on M^b_f.

For an illustration, when an intersection edge e between two other surfaces f' and f'' crosses

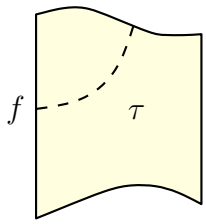

Figure 4: Tracing an intersection curve involving f along the floor or ceiling of a prism τ (view from above). At the left exit point of the curve we glue it to its continuation within an adjacent prism, and the top exit point is a real feature (vertex or part of a vertically visible pair) of the new decomposition.

f, at some 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 on the top side of f, and that of e^- only on the bottom side. Both projections are arcs emanating from v. See, e.g., Figure 5, where the intersection curve between the surfaces a and b intersects f in a vertex v.

As another illustration, if w is a real vertex of $\mathcal{A}^{+f}_{\leq t}(F_i)$, incident to three surfaces f_1, f_2, f_3, which is vertically visible from f and lies, say, above f, then w is incident to three intersection edges of pairs of these functions. Each of these edges is split at w into two portions, one visible from f and one invisible (hidden by the third function), so we draw on the top side of f three respective projected arcs, all emanating from the projection of w. See Figure 5 for an illustration of these structures, where the real vertices 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.

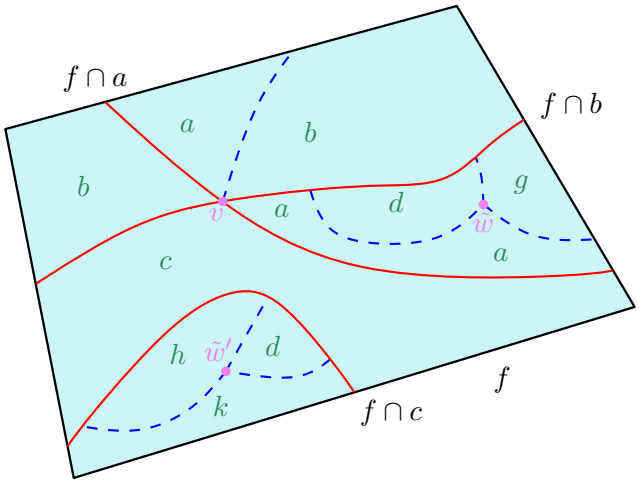

Figure 5: The first stage of the vertical decomposition on (the top side of) the newly inserted surface f. Red (solid) arcs depict real intersection edges and blue (dashed) arcs depict shadow arcs. The label of each face denotes the surface that appears vertically above f over that face.

The shadow edges are also fairly easy to collect: they are composed of projections onto f of

portions of y-edges of the prisms that f crosses, which are portions of real intersection edges that are vertically visible from f. These edges are collected from all the prisms τ that f crosses, based on the local information available at each such prism. For example, if such an edge e lies on the top side of the prism τ, we take the y-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 on (the top side of) f over e' as e' itself; see Figure 6. The other portions of e are not handled within τ, since the information we have at τ does not let us know whether these pieces are at all visible (in their entirety) from f; these pieces will be handled within other nearby prisms that f crosses and have e (or an edge overlapping e) as an edge. Similarly, real intersection edges on the bottom side of τ will be handled in a fully symmetric manner, and their relevant portions will be drawn as shadow edges on the bottom side of f. Once we have all these pieces of shadow edges, we glue them together into full shadow edges. Each such edge ends either when (i) it reaches a real intersection edge γ on f (the original intersection edge e is no longer visible from f on the other side of γ, as the other surface forming γ, rises above f on the other side and hides e), or when (ii) it reaches the projection onto f of a real vertex (an endpoint of e), or (iii) at a real vertex on f where e crosses f (the continuation of e past this vertex has to be drawn on the other side of f). Again, see Figure 5 and also Figure 6 for an illustration of all these situations.

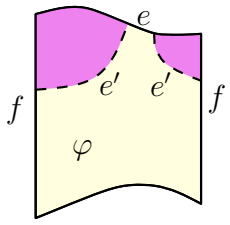

Figure 6: Creating shadow edges on the top side of f within a vertical curtain φ (a side view).

Note that endpoints of type (i) correspond, in a one-to-one manner, to new vertically visible pairs of edges (each of which necessarily involves f as one of its four defining functions) Note also that there might be shadow edges that do not cross 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 might also be closed or unbounded Jordan curves.

The time to construct all the real edges and shadow edges, as well as (a DCEL representation of) the planar map M_f^t (resp., M_f^b) that they form along the top (resp., bottom) side of f, is linear in the number of prisms that f crosses.

Each face of M_f^t is the lower face of some cell of the first stage of the vertical decomposition of $\mathcal{A}_{\leq t}^{+f}(F_i)$, namely, the decomposition of $\mathcal{A}_{\leq t}^{+f}(F_i)$ obtained by adding the z-vertical curtains from all the edges of $\mathcal{A}_{\leq t}^{+f}(F_i)$. Recall that each such 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 $\overline{H}$ the collection of these cells that have f on their floor. Each cell of $\overline{H}$ is the union of fragments of the prisms of $\mathrm{VD}_{\leq t}(F_i)$ (obtained by decomposing prisms of $\mathrm{VD}_{\leq t}(F_i)$ that are intersected by f into smaller "local" prisms, as mentioned above). Each cell of $\overline{H}$ can be obtained by traversing these smaller prisms and by "gluing" them along suitable common x-faces.

In the next step, we perform the second stage of the vertical decomposition within each cell B

of $\overline{H}$, by computing the x-faces that partition it into our standard, constant-complexity prisms. 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-parallel edges of $\mathrm{VD}(B_f)$, when lifted to three dimensions and intersected with B, define the x-faces of the desired vertical decomposition of B, which we denote as $\mathrm{VD}(B)$.

Constructing the Conflict Lists. Finally, we construct the conflict lists of the prisms $\tau \in \mathrm{VD}(B)$, in three stages. In the first stage we distribute functions g to conflict lists of prisms τ, such that the connected component of $g \cap B$ that contains $g \cap \tau$ also contains the intersection of g with a y-edge of B. In the second stage we distribute functions g to conflict lists of prisms τ, such that the connected component of $g \cap B$ that contains $g \cap \tau$ intersects only the top or bottom faces of B. In the third stage we find the functions that penetrate τ through its z-parallel edges. Clearly, these are the only (not necessarily mutually exclusive) ways in which g can intersect a prism $\tau \in \mathrm{VD}(B)$.

We start by computing the conflict lists of the y-edges of B. Each y-edge e of B is the concatenation of y-edges of (refined) prisms that comprise B (namely, prisms which are fragments of prisms intersected by f). The conflict list $\mathrm{CL}(e)$ of e is the union of the conflict lists of these y-edges (recall that the conflict lists of the refined prisms have already been constructed, as described earlier). We then sort the intersections of the functions in $\mathrm{CL}(e)$ with e, along e. Finally, we traverse this sorted list of intersections and distribute the functions in $\mathrm{CL}(e)$ to the prisms of $\mathrm{VD}(B)$ that contain a segment of e as a y-edge. When we put a function g in $\mathrm{CL}(\tau)$ for some $\tau \in \mathrm{VD}(B)$ we check which of the prisms τ' adjacent to τ intersects g and add g to $\mathrm{CL}(\tau')$ if $\tau' \cap g \neq \emptyset$. If we added g to $\mathrm{CL}(\tau')$ we continue this search to prisms adjacent to τ' and so on. The search stops at each prism that does not intersect g. This completes the first stage of the construction of the conflict lists

For the second stage, we merge the conflict lists of all the (refined) prisms that comprise B into one list, which we denote by $\mathrm{CL}(B)$. We construct a planar point location data structure over $\mathrm{VD}(B_f)$. For each function $g \in \mathrm{CL}(B)$, we compute the intersections $g \cap f$ and $g \cap h$, where h is the function containing the ceiling of B. In each connected component of $g \cap f$, we pick an arbitrary point p, and find, using the point location data structure if $p \in B_f$ and if so we also find the trapezoid $T \in \mathrm{VD}(B_f)$ that contains p. If p is out of B_f we stop. Otherwise, we add g to $\mathrm{CL}(\tau_T)$, where τ_T is the lifted image of T, that is, the prism whose top and bottom faces project to T. Then, for each trapezoid T' adjacent to T, we check whether the lifted image of T' to B_f also intersects g. If so, we add g to $\mathrm{CL}(\tau_{T'})$, and continue the search through $\mathrm{VD}(B_f)$ to the neighbors of T'. The search stops at trapezoids whose lifted images to B_f do not intersect g. We repeat this procedure, in a fully symmetric fashion, to the ceiling B_h of B, over the connected components of $g \cap h$. This yields all the new prisms into which g penetrates through their floor or ceiling. Repeating this for all functions $g \in \mathrm{CL}(B)$, and for all regions B, completes the second stage of the construction. Note that if the connected component of $g \cap f$ or $g \cap h$ intersects a y-edge of B, then we have already added g to all the conflicts list of prisms that intersect this connected component at the first stage.

We now describe the execution of the third stage. For each z-parallel edge ζ of some prism $\tau \in \mathrm{VD}(B)$, the bottom (resp., top) endpoint ζ^b (resp., ζ^t) of ζ is a vertex on some y-edge of B. We process all the z-parallel edges adjacent to each y-edge e of B in turn. We pick an arbitrary initial z-parallel edge ζ_0 intersecting e which is also a z-parallel edge of some (refined) old prism, so we have the conflict list of ζ_0 ready. (It is easy to see that there must be such a z-parallel edge Let ζ be a z-parallel edge adjacent to ζ_0 along e. An easy but crucial observation is that any

function g that intersects ζ must also cross either ζ_0 or one of the top and bottom y-subedges of e that connect ζ_0^t to ζ^t or ζ_0^b to ζ^b. We can therefore obtain $\mathrm{CL}(\zeta)$ by inspecting all the functions in $\mathrm{CL}(\zeta_0)$ and in the conflict lists of this pair of y-edges. We continue this traversal in a similar manner and compute the conflict lists of all new z-parallel edges that intersect e.

We obtain the final conflict list of each prism $\Delta \in \mathrm{VD}(B)$ by uniting its conflict lists from the first and second stages and the conflict lists of its z-parallel edges, as computed in the third stage.

Running time. To compute the conflict lists of the first stage, we perform $O(1)$ point location queries for each element of $\mathrm{CL}(B)$. Following each point location query, we traverse a connected subset of prisms (separated by x-faces) to locate all prisms containing a particular connected component of $g \cap f$ or $g \cap h$ for each $g \in \mathrm{CL}(B)$. This scan performs $O(1)$ operations per item in a conflict list of a new trapezoid. The sorting of the intersections of items in $\mathrm{CL}(e)$ along each y-edge of B takes logarithmic time per item in these conflict lists. It is also easy to verify that the work in the second stage of the vertical decomposition is proportional to the total size of the conflict lists of the destroyed and the new prisms. Summing up we get that the time it takes to compute the conflict lists of the prisms is proportional to their size times a logarithmic factor.

We produce the new prisms whose top face lies on f and their conflict lists analogously, using M_f^b rather than M_f^t.

Prisms for which f defines a y-edge or an x-face (that passes through a vertex on f). Consider the new prisms of this kind whose bottom faces lie on the graph of g, for 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 top faces are on f, which we have already computed. On the other side of e, we draw the projections of the y-faces of the fragments of old prisms that were cut by f and intersect e.

Each cell which we obtain on g is the projection on g of a cell (with a bottom face on g) of the first stage of the vertical decomposition of the arrangement of the intersection of f with the first t levels of $\mathcal{A}(F_i)$. We compute the vertical decomposition of each such cell and the conflict lists of its prisms as we did for cells with a bottom or a top face on f.

We repeat this process for each side of each function intersected by f. See Figure 7 for an illustration.

Removal of prisms which are above $\mathcal{A}_{\leq t}(F_{i+1})$. Consider an old prism τ that lay at the top t-level (that is, its ceiling was part of the t-level) before f was inserted. If f passes fully below τ, the level of τ goes up by 1, to $t+1$, and τ has to be removed from the structure. Similarly, some new prisms are now at level $t+1$, and we need to remove them too.

If we could explicitly record in the structure the level of each prism that it stores, and update these counters after each insertion, the removal of these "overflowing" prisms would be trivial—simply remove them from the structure. This however is expensive to do. The main difficulty is that there might be many prisms that lie fully above f, as discussed above, and broadcasting to all of them that their level has increased by 1 will in general be too expensive.

Instead, we note that any connected patch of prisms that has to be removed is surrounded by prisms that have f as their floor, so that this portion of f lies at the t-level. Such prisms are easy to identify: any old prism τ_0 that overlaps such a prism τ had its ceiling at level t before f was

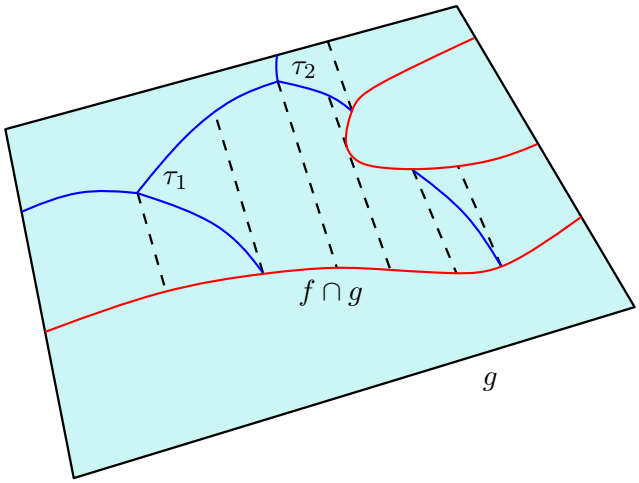

Figure 7: Collecting prisms to which f contributes only a y-edge or a vertex. The figure depicts floors of such prisms on an older surface g. The red arcs depict $f \cap g$, and the blue arcs depict (portions of) older real or shadow arcs. For the prisms τ_1 and τ_2, f only contributes a vertex. For all the other prisms, f contributes a y-edge. f passes below g in the region decomposed into prisms, and above g in the complementary regions.

inserted, and τ_0 "was aware" of this fact simply because there were no prisms above the ceiling, a property that is easy to detect by locally inspecting the neighbors of τ_0.

We therefore identify these "fringe" prisms, and then carry out a transversal of the adjacent prisms, collecting in this manner all the prisms that have to be removed. To better understand this process, we note that each cell C of the first stage of the vertical decomposition is such that all its prisms are at the same level of $\mathcal{A}^{+f}_{\leq t}(F_i)$, and they are adjacent to each other only through common x-faces. Hence, if one of these prisms lie at the fringe, they all are.

When we cross a y-face of a prism, we pass into another region of the first-stage decomposition. The level does not change in this case, but such a y-face may overlap several y-faces from the other side, because the second-stage decompositions within each first-stage region are not compatible.

When we cross the floor or the ceiling of the current prism, the level increases or decreases, so we will not do this. We do face, however, the situation where we reach a y-edge e of a prism in which its floor and ceiling meet. When we cross e (technically, we cross the ceiling upwards, move over e, and then cross the floor downwards), we reach a new cell of $\mathcal{A}^{+f}_{\leq t}(F_i)$ that also lies at level $t+1$ (note that, locally near e, the floor and ceiling surfaces are swapped). Here too, there may be several prisms that touch e on its other side, in the above sense, all part of a single first-stage decomposition cell, and they are adjacent to one another, in the order in which they meet e, via common x-faces.

This traversal finds all prisms that are now at level $t+1$ and need to be discarded. The traversal terminates in situations when we reach a y-edge e at which the floor and ceiling of a prism meet, and there is "nothing" on the other side of e. This would be the case when f is the floor of the current prism. On the other side of e, f should become the ceiling, lying at level $t+1$, but this part of f would not have been processed, as there were no older prisms at this level for f to cross.

6.2 Analysis

The 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, times a logarithmic factor. Let Π denote the set of all possible pseudo-prisms. That is, we consider all possible subsets F_0 of up to six functions of F, and for each such F_0, we construct the vertical decomposition of the entire arrangement $\mathcal{A}(F_0)$, and add all resulting prisms to Π.

We associate two *weights* with each prism $\tau \in \Pi$. The first weight, denoted $w_0(\tau)$, is equal to the size of its conflict list. The second weight, denoted $w^-(\tau)$, is equal to the number of surfaces that pass fully below τ. For simplicity, we focus below on prisms that are defined by exactly six functions; the treatment of prisms defined by fewer functions is done in a fully analogous manner.

Following one of the standard approaches to the analysis of RICs, we proceed in two steps. First, we estimate the probability that a prism with given weights ever appears in the first t levels of $\mathcal{A}(F_i)$, for some $i \leq n$, during the incremental process. Then we estimate the number of prisms with weights $\leq a$, $\leq b$, using the Clarkson-Shor technique and several other considerations, and then combine the bounds to get the desired bound on the expected running time and storage of the algorithm.

Estimating the probability of a prism to appear. For the first step, let τ be a prism in Π with weights $w^-(\tau) = a$, $w_0(\tau) = b$, and with six defining functions. We refer to the a surfaces counted in $w^-(\tau)$ as the *lower surfaces* of τ, and to the b surfaces counted in $w_0(\tau)$ as the *crossing surfaces* of τ.

Then τ appears as a prism in some $\mathrm{VD}_{\leq t}(F_i)$ if and only if (i) the last of the six defining functions, call it f_6, is inserted before any of the b crossing surfaces; and (ii) at most $t' = t - 1 - \xi$ of the a lower surfaces are inserted before f_6, where ξ is the number of defining functions of τ that pass below τ (the "1" accounts for the floor of τ)

An explicit calculation of this probability, denoted p_τ, goes 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 defining τ). To get a restricted permutation that satisfies (i) and (ii), we first choose which of the six defining functions is the last one f_6, then we choose some $j \leq \min\{a, t'\}$ of the a lower surfaces to precede f_6, then mix these j surfaces with the other five defining ones, and finally place the remaining $a - j$ lower surfaces and all b crossing surfaces after f_6. We thus get

$$p_\tau = \sum_{j=0}^{\min\{a,t'\}} \frac{6\binom{a}{j}(j+5)!(a+b-j)!}{(a+b+6)!}. \tag{4}$$

We rewrite and upper bound each term in the sum 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 the bound we obtained on the jth item of the sum in (4).

Note that with a, b fixed, $\varphi_{a,b}(x)$ peaks at $x = \frac{5(a+b)}{b} - 5 = \frac{5a}{b}$ (the zero of the derivative, satisfying $5(x+5)^4 - \frac{b}{a+b}(x+5)^5 = 0$).

We estimate p_τ by replacing the sum by an integral. That is, we have

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

to justify bounding the sum by the integral in the third inequality above, it suffices to note that, for $x \in [j, j+1]$,

$$\frac{\varphi_{a,b}(x)}{\varphi_{a,b}(j)} = \left(\frac{x+5}{j+5}\right)^5 e^{-b(x-j)/(a+b)} \ge \frac{1}{e},$$

for every $j \ge 0$, from which the argument follows.

To estimate the integral, we apply the substitution $y = xb/(a+b)$, replace the upper limit by

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

and get

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

The integral is at most

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

which is some absolute constant. Thus,[7] $p_\tau = O(1/b^6)$. For large c we cannot improve on this bound but if c is sufficiently small, the integration up to c may accumulate only a fraction of the total (constant) mass underneath the integrand. For $a \leq t$ we will not refine the bound and use $p_\tau = O(1/b^6)$. Consider now the case where $a > t$ so $c = \min\{a, t'\}\frac{b}{a+b} = \frac{bt'}{a+b}$.

As it is easily checked, the integrand peaks at $y = 5a/(a+b)$, so when

$$c = \cdot\frac{bt'}{a+b} \leq \frac{5a}{a+b}, \qquad \text{or} \qquad t' \leq \frac{5a}{b}\ ,$$

we tighten the bound and upper bound the integral by c times the value of the integrand at $y = c$. We get[8]

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

To recap, we have an upper bound for p_τ in terms of a and b. Denoting this bound by $p(a,b)$, we have

$$p(a,b) = \begin{cases} O\left(\frac{1}{b^6}\right) & \text{for } a \leq bt'/5 \text{ or } a \leq t \\ O\left(\frac{t^6}{a^6}\right) & \text{for } a > bt'/5 \text{ and } a > t. \end{cases} \tag{5}$$

(Unless b is very small, the constraint $a \leq t$ 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 τ with $w^-(\tau) \leq a$ and $w_0(\tau) \leq b$.

Lemma 6.2. *The number of prisms τ with $w^-(\tau) \leq a$ and $w_0(\tau) \leq b$ is $O(nb^5)$ for $a \leq b$, and $O(nab^4\lambda_{s+2}(a/b))$ for $a > b$.*

Proof. Pick a random sample R of F where each function is chosen with some probability p that we will fix later. The expected size of R is np. Let τ be a prism in Π, defined by six functions, with $w^-(\tau) = i$ and $w_0(\tau) = j$, with $i \leq a$, $j \leq b$. The probability q_τ that τ appears in the vertical decomposition of the first ξ levels of $\mathcal{A}(R)$ (ξ is another parameter that we will fix shortly) is the probability of the event that (i) the six defining functions of τ are chosen in R; (ii) none of the j crossing functions is chosen; and (iii) at most ξ of the i lower functions are chosen.

To proceed, we distinguish, as in the statement of the lemma, between the cases $a \leq b$ and $a > b$. In the former case, we replace the third constraint by the constraint (iii') none of the i lower functions are chosen. This only lowers the estimate for q_τ, and we thus get (note that in this part of the analysis ξ is irrelevant).

$$q_\tau \geq p^6(1-p)^j(1-p)^i \geq p^6(1-p)^{a+b} \geq p^6(1-p)^{2b}.$$

[7] 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.

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

In this case, τ becomes a prism in the vertical decomposition of the first two levels in $\mathcal{A}(R)$. By Lemma 6.1, the number of such prisms is $O(|R|)$, so its expected value is $O(np)$. Choosing $p = 1/b$, this yields, as in Clarkson and Shor, $N_{\leq a,\leq b} = O(nb^5)$.

Consider next the case $a > b$. The probability of (i) and (ii) is, as above, $p^6(1-p)^j \geq p^6(1-p)^b$. The event (iii), which is independent of (i) and (ii), is to have at most ξ successful trials in a Bernoulli process with i trials, each with success probability p. We take $p = 1/b$ and $\xi = 2a/b$. Since the mean of the process is $i/b < a/b$, Chernoff's bound implies that the failure probability is at most $e^{-a/(3b)} \leq e^{-1/3}$. Hence, with this choice of ξ, we have

$$q_\tau \geq \frac{1 - e^{-1/3}}{b^6} \left(1 - \frac{1}{b}\right)^b = \Omega(1/b^6).$$

To complete the Clarkson-Shor analysis, we need an upper bound on the (expected) number of prisms in the vertical decomposition of the first ξ levels of $\mathcal{A}(R)$. By Lemma 6.1, this number is $O(|R|\xi\lambda_{s+2}(\xi))$. The analysis thus yields

$$N_{\leq a,\leq b} = O\left(b^6 np \cdot \xi\lambda_{s+2}(\xi)\right) = O\left(b^6(n/b)(a/b)\lambda_{s+2}(a/b)\right) = O\left(nab^4\lambda_{s+2}(a/b)\right),$$

as asserted. □

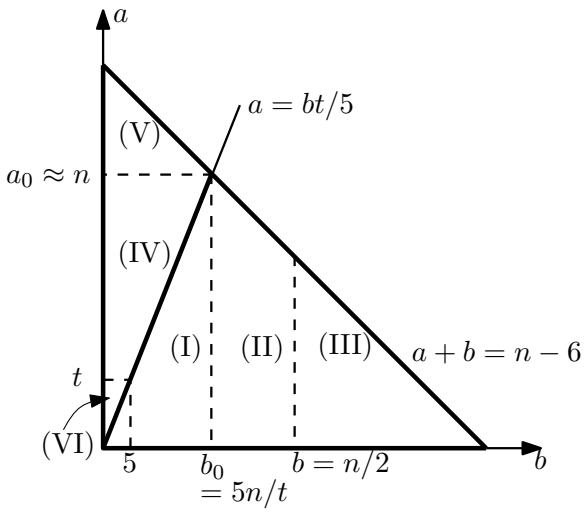

Figure 8: The decomposition of the (a, b)-range into subranges.

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

$$\sum_{\tau \in \Pi} p_\tau = \sum_a \sum_b p(a, b) N_{a,b}. \tag{6}$$

Similarly, the expected overall size of the conflict lists is

$$\sum_{\tau\in\Pi} w_0(\tau)p_\tau = \sum_a \sum_b bp(a,b)N_{a,b}. \tag{7}$$

We bound these sums separately for pairs (a,b) within each of the six regions depicted in Figure 8. Together, these regions cover the entire range $a,b \geq 0$, $a+b \leq n-6$. Note that 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 $j = 1,2,\ldots$, where $b_j = 5\cdot 2^j$. 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 6.2, is

$$O\left(n(b_j t/5)b_j^4 \lambda_{s+2}(t/5)\right) = O\left(nb_j^5 t\lambda_{s+2}(t)\right) = O\left(2^{5j} nt\lambda_{s+2}(t)\right).$$

Hence, the contribution of S_j to (6) is at most

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

and, summing this over j, we get that the contribution of region (I) to (6) is $O(nt\lambda_{s+2}(t))$.

Similarly, the contribution of S_j to (7) is at most

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

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 (7) is $O(nt\lambda_{s+2}(t)\log(n/t))$.

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 $j = 1,2,\ldots$, where $b'_j = (5n/t)\cdot 2^j$. 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 $N_{\leq n-b'_{j-1}-6, \leq b'_j}$ which, by Lemma 6.2, is (upper bounding $n - b'_{j-1} - 6$ simply by n)

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

Hence, the contribution of S'_j to (6) is at most

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

and, summing this over j, we get that the contribution of region (II) to (6) is $O(t^2\lambda_{s+2}(t)) = O(nt\lambda_{s+2}(t))$.

Similarly, the contribution of S'_j to (7) is at most

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

and, summing this over j, we get that the contribution of region (II) to (7) is $O(nt\lambda_{s+2}(t))$.

Region (III). In this region, $n/2 \leq b \leq n$ and $0 \leq a \leq n-6-b$. We treat this region as a single entity. The maximum value of p_τ here is $O(1/n^6)$, and we bound $\sum_{(a,b)\in(III)} N_{a,b}$ by the overall number of prisms, which is $O(n^6)$, getting a negligible contribution to (6) of only $O(1)$. A similar argument shows that the contribution of this region to (7) is $O(n)$, again negligible compared with the other regions.

Region (IV). In this region, $t \leq a \leq a_0 \approx n$ and $0 \leq b \leq 5a/t$. We cover the region by horizontal slabs of the form $S''_j := \{(a,b) \mid a_{j-1} \leq a \leq a_j\}$, for $j = 1, 2, \ldots$, where $a_j = t \cdot 2^j$. 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 $N_{\leq a_j, \leq 5a_j/t}$ which, by Lemma 6.2, is

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

Hence, the contribution of S''_j to (6) is at most

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

and, summing this over j, we get that the contribution of region (IV) to (6) is $O(nt\lambda_{s+2}(t))$.

Similarly, the contribution of S''_j to (7) is at most

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

Here too, as in the case of region (I), the number of slabs is $O(\log(n/t))$, making the contribution of region (IV) to (7) is $O(nt\lambda_{s+2}(t)\log(n/t))$.

Region (V). In this region, $a_0 \leq a \leq n$ and $0 \leq b \leq n-a-6$. 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 $\sum_{(a,b)\in(V)} N_{a,b}$ by

$$N_{\leq n, \leq 5n/t} = O\left(n^2(5n/t)^4\lambda_{s+2}(t/5)\right) = O(n^6\lambda_{s+2}(t)/t^4).$$

Hence, the contribution of region (V) to (6) is at most

$$O\left(t^2\lambda_{s+2}(t)\right) = O(nt\lambda_{s+2}(t)).$$

For the contribution to (7), we multiply this bound by $O(n/t)$, an upper bound on b in this region, and get

$$O\left((n/t)\cdot t^2\lambda_{s+2}(t)\right) = O(nt\lambda_{s+2}(t)).$$

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 $\sum_{(a,b)\in(VI)} N_{a,b}$ by $N_{\leq t, \leq 5}$, which is $O(nt\lambda_{s+2}(t))$. Hence, the contribution of region (VI) to (6) is at most $O(nt\lambda_{s+2}(t))$. Since b is bounded by a constant in this region, the same expression also bounds the contribution of region (VI) to (7).

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

Theorem 6.3. *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 $O(nt\lambda_{s+2}(t)\log(n/t)\log n)$, and whose expected storage is $O(nt\lambda_{s+2}(t))$.*

7 Improved Dynamic Lower Envelopes for Planes

In this section, we present our own 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 of planes, or dually, dynamically maintaining convex hulls in $\mathbb{R}^3$. In the next section, we will see how to combine this structure with the results from the previous sections to obtain a data structure that works for general surfaces and that achieves polylogarithmic update and query time.

We present our version of the structure in three phases: We begin with a static structure, then develop a variant of a simple standard technique for extending it so as to allow insertions, and finally describe how to perform deletions. With the help of a simple counting argument in the construction of the static structure, we also manage to improve Chan's bound for the deletion time by a logarithmic factor, the first improvement on this problem in ten years.

7.1 A static structure

Let H be a fixed set of n non-vertical planes in $\mathbb{R}^3$. We fix a constant level k_0, and we consider the sequence of levels $k_j = 2^j k_0$, for $j = 0, 1, \ldots, m$, where $m = \lfloor \log(n/k_0) \rfloor$. (Level k_m lies somewhere between levels $n/2$ and n.)

We construct a sequence of shallow cuttings, one for each level k_j in the following iterative manner. The initial cutting is a vertical k_m-shallow $(\alpha k_m/n_m)$-cutting Λ_m, of $\mathcal{A}(H)$ for some fixed constant $\alpha > 1$, as in [15, 27]. During this process, we keep pruning away planes from H (these are not real deletions, but are only made to ensure efficient performance of the data structure). So, when we reach level k_j, we have a subset H_j of $n_j \leq n$ surviving planes and we construct a vertical k_j-shallow $(\alpha k_j/n_j)$-cutting, Λ_j, of $\mathcal{A}(H_j)$.[9] That is, Λ_j consists of $O(n_j/k_j)$ semi-unbounded vertical prisms, where each prism τ consists of all the points that lie vertically below some triangle $\overline{\tau}$ (the 'ceiling' of τ), so that these triangles form the faces of a polyhedral terrain $\overline{\Lambda}_j$. The number of planes intersecting each prism τ is at most αk_j and its ceiling $\overline{\tau}$ lies fully above level k_j.

With each prism τ of Λ_j, we maintain its *conflict list* C_τ, containing the at most αk_j planes of H_j that cross τ. However, before storing these lists, we prune away all the planes $h \in H_j$ that belong to more than $c \log n$ conflict lists of prisms in *all* the cuttings constructed so far (including the new Λ_j);[10] here c is some sufficiently large constant that we will specify shortly. We collect all the pruned planes in a remainder set $H^{(2)}$, and we let H_{j-1} denote the set of surviving planes.[11] A subtle yet important property of the construction is that when we prune away a plane, we do *not* delete it from the conflict lists of prisms that have been constructed for higher levels. This pruning mechanism ensures that each plane of H is stored in at most $c \log n$ conflict lists, a property crucial for the efficiency of the algorithm.[12]

We keep iterating in this manner, until we reach the k_0-level, and then stop; note that planes can still be pruned away at this step. The conflict lists of the prisms of this lowest level is the only

[9] In particular $H_m = H$.

[10] 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 level individually. Our approach ensures that each plane appears in $O(\log n)$ conflict lists in a static structure, 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.

[11] We use the same remainder set $H^{(2)}$ for all levels j of the construction. For consistency, we also put $H^{(1)} := H$.

[12] 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).

output that we use for answering queries. (To support deletions, as will be described below, we will also need the conflict lists of the prisms of all the other cuttings in the hierarchy, not for answering queries but for controlling the deletion mechanism.) We denote by $\mathcal{D}^{(1)}$ the structure consisting of the resulting sequence of shallow cuttings and the conflict lists of their prisms.

We associate with $\mathcal{D}^{(1)}$ the set of planes $H \setminus H^{(2)}$ that survive the construction of $\mathcal{D}^{(1)}$ (without being pruned away at any stage), and denote it by $H(\mathcal{D}^{(1)})$. The following lemma bounds the number of planes that are pruned and collected in $H^{(2)}$ during the construction.

Lemma 7.1. *For a suitable choice of c, the overall number of planes that get pruned, over all levels k_j, is at most βn, for a constant $\beta < 1$ inversely proportional to c. That is, $|H(\mathcal{D}^{(1)})| \geq (1-\beta)n$.*

Proof. Define a potential function Φ to be equal to the sum of the sizes of the conflict lists up to some level k_j in the construction, where the *size* of a conflict list C_τ is defined to be the number of planes of H_{j-1} that cross τ (i.e. planes of H_j that were not pruned away at this stage).

Since Λ_j consists of $O(n_j/k_j)$ prisms, and the conflict list of each prism contains at most αk_j planes, the overall size of the conflict lists of the prisms of Λ_j is $O(n_j) = O(n)$. Hence, generating Λ_j and the conflict lists of its prisms increases Φ by at most γn, for some fixed constant γ.

Each plane h that we prune away is contained in at least $c \log n$ conflict lists. Therefore the pruning of h reduces Φ by at least $c \log n$. Since Φ is initially 0 and never negative, and since we increase it by at most $\gamma n \log n$ units, it follows that we discard at most βn, planes for $\beta = \gamma/c$. $\square$

We now repeat the whole process, and apply it to the set $H^{(2)}$, obtaining an analogous structure $\mathcal{D}^{(2)}$, and a remainder set $H^{(3)}$ of at most $\beta^2 n$ planes that got pruned away at some level. Proceeding in this manner for at most $\log_{1/\beta} n$ steps, we obtain the complete structure, that we denote as $\mathcal{D}$, which is simply the sequence of substructures $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \ldots$. Note that the sets $H(\mathcal{D}^{(i)})$ are disjoint with union H. Furthermore, Lemma 7.1 implies that $|H(\mathcal{D}^{(i)})| \geq 2|H(\mathcal{D}^{(i+1)})|$, since we can choose c to obtain $\beta < 1/2$.

For each i, the overall size of $\mathcal{D}^{(i)}$, including the conflict lists, is

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

(Note that the number of levels m depends on $|H^{(i)}|$ (it is $\lfloor \log(|H^{(i)}|/k_0) \rfloor$), and decreases as i increases; for simplicity, we use the original value of m in the above upper bound, for each i.) Since $|H^{(i)}| \leq \beta^{i-1} n$, the overall size of $\mathcal{D}$ is $O(n \log n)$.[13]

We can construct each cutting Λ_j, in each of the substructures $\mathcal{D}^{(i)}$, using the algorithm of Chan and Tsakalidis [15], 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, using the fact that $|H^{(i)}|$ decreases geometrically with i, we get a total of $O(n \log^2 n)$ running time.

Answering a query is easy: Given a point q in the xy-plane, we iterate over each substructure $\mathcal{D}^{(i)}$, and find the prism τ of the corresponding lowest cutting Λ_0 (of the first k_0 levels) whose xy-projection contains q. This is done using a suitable point-location structure constructed for the xy-projection of Λ_0. We then access the conflict list C_τ of τ, and search in it, in brute force, for the lowest plane over q. We repeat this search over all substructures $\mathcal{D}^{(i)}$, and return the plane that is lowest over q among all $O(\log n)$ candidate outputs. The cost of a query is thus $O(\log^2 n)$ time.

[13] Later we will show how to reduce the storage to linear, by representing the conflict lists in an implicit manner.

7.2 Handling insertions

We use a variant of a standard technique, originally due to Bentley and Saxe [7] and later refined in Overmars and van Leeuwen [38] (see also Erickson's notes [24]).

Specifically, we maintain a sequence $\mathcal{I} = (\overline{\mathcal{D}}_{i_1}, \overline{\mathcal{D}}_{i_2}, \ldots, \overline{\mathcal{D}}_{i_k})$ of structures, where $0 \leq i_1 < i_2 < \cdots < i_k$. (These indices are not fixed, and in general will vary after each insertion. Lemma 7.2 below shows that the length of $\mathcal{I}$ is $O(\log n)$.) Each $\overline{\mathcal{D}}_{i_j}$ is a substructure $\mathcal{D}^{(u)}$ of some static structure $\mathcal{D}$, as constructed above, over some subset $\overline{H}_{i_j}$ of H. We maintain the following invariants.

1. The set $\overline{H}_{i_j}$, which is used to construct $\overline{\mathcal{D}}_{i_j}$, has size at most 2^{i_j}, and $|H(\overline{\mathcal{D}}_{i_j})| > 2^{i_j-1}$.
2. The sets $H(\overline{\mathcal{D}}_{i_j})$ are disjoint and their union is the set of planes currently in the data structure.

We refer to the structure $\overline{\mathcal{D}}_{i_j}$ as the structure at *location* i_j. For each plane h, we refer to the structure $\overline{\mathcal{D}}_{i_j}$ such that $h \in H(\overline{\mathcal{D}}_{i_j})$ as the structure that *stores* h. If h is stored at $\overline{\mathcal{D}}_{i_j}$ we also say that h is stored at location i_j.[14]

When a plane h is inserted, we look for the smallest non-negative integer j that does not belong to the sequence $(i_1, i_2, \ldots, i_k)$. If $j = 0$, we just set $\overline{H}_0 := \{h\}$, and construct over it a trivial structure $\overline{\mathcal{D}}_0$ with only one plane.

Otherwise, we set $\overline{H}_j := \left(\bigcup_{i=0}^{j-1} H(\overline{\mathcal{D}}_{i_j})\right) \cup \{h\}$. Assuming that Invariant (1) holds prior to the insertion of h, we have

$$2^{j-1} \leq 2 + \sum_{i=1}^{j-1} 2^{i-1} \leq |\overline{H}_j| = 1 + \sum_{i=0}^{j-1} |H(\overline{\mathcal{D}}_{i_j})| \leq 1 + \sum_{i=0}^{j-1} |\overline{H}_i| \leq 1 + \sum_{i=0}^{j-1} 2^i = 2^j. \tag{8}$$

We construct over $\overline{H}_j$ a static structure $\mathcal{D} = \mathcal{D}(\overline{H}_j)$ as in Section 7.1, and recall that $\mathcal{D}$ is a sequence of a logarithmic number of substructures, $\mathcal{D}^{(1)}, \mathcal{D}^{(2)}, \ldots, D^{(s)}$, where $s \leq \log |\overline{H}_j| \leq j$. We remove $\overline{\mathcal{D}}_{i_0}, \ldots \overline{\mathcal{D}}_{i_{j-1}}$ (which are in fact $\overline{\mathcal{D}}_0, \overline{\mathcal{D}}_1, \ldots, \overline{\mathcal{D}}_{j-1}$) from $\mathcal{I}$. Then, for each structure $\mathcal{D}^{(u)}$, we set

$$\overline{\mathcal{D}}_i := \mathcal{D}^{(u)} \text{ for } i = \lfloor \log |H(\mathcal{D}^{(u)})| \rfloor + 1, \tag{9}$$

and add it to $\mathcal{I}$. Note that by Equation (8) and Lemma 7.1, if $\beta < 1/2$, $\mathcal{D}^{(1)}$ is set to be either $\overline{\mathcal{D}}_j$ or $\overline{\mathcal{D}}_{j-1}$. It also follows from Lemma 7.1 that each structure $\mathcal{D}^{(u)}$ is associated with a different index $i \leq j$. It is clear from the definition of the algorithm that Invariants (1) and (2) hold.

The number of planes in our structure changes as we do insertions (and later deletions). We denote this number by n and treat it as a static quantity. That is, we keep it fixed (equal to some power of 2), even though the actual number of planes changes due to insertions and deletions. To justify this, we add a global rebuilding mechanism to the structure that rebuilds it entirely from scratch whenever the number of elements changes (increases or decreases) by a factor of 2. When this happens, we double or half our n, as appropriate.

The following lemma gives an upper bound on the number of structures in $\mathcal{I}$.

Lemma 7.2. *The number of structures in $\mathcal{I}$ is at most $\lfloor \log n \rfloor + 3$.*

Proof. The actual number of planes currently in the structure is at most $2n$, where n is the static size of the structure, as defined above. It follows by Invariant (1) that the largest index j for which $\overline{\mathcal{D}}_j$ exists is $\lfloor \log(2n) \rfloor + 1 = \lfloor \log n \rfloor + 2$. □

[14] Note that a plane h may also appear in conflict lists of substructures $\overline{\mathcal{D}}_{i_j}$ that do not store it, as is the case with the static structure too.

Lemma 7.3. *The deterministic amortized cost of an insertion is $O(\log^3 n)$, and the deterministic worst-case cost of a query is $O(\log^2 n)$, where n is the number of planes in the data structure when we perform the operation.*

Proof. The claim for the cost of a query is obvious: In each of the $O(\log n)$ structures $\overline{\mathcal{D}}_j$ of $\mathcal{I}$ we find the prism of the corresponding lowest level Λ_0 whose xy-projection contains the query point q, and search over its at most k_0 planes for the lowest one over q. This takes $O(\log n)$ time per structure, by augmenting $\overline{\mathcal{D}}_j$ with a planar point location data structure over the minimization diagram of Λ_0 (as in the static structure).

Concerning insertions, we recall that an insertion of a plane h destroys a prefix of length j of the subsets in $\mathcal{I}$ and moves their planes, including h, to a subset $\overline{H}_j$, computes a new static structure for $\overline{H}_j$, and spreads its substructures $\mathcal{D}^{(u)}$ according to Equation (9) from j downwards along the prefix of $\mathcal{I}$. The real cost of such an insertion is $a|\overline{H}_j| \log^2 |\overline{H}_j|$, for some absolute constant a.

To pay for this cost, we use the following charging argument. We maintain the invariant that 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 location of h, and $w := \lfloor \log n \rfloor + 4$ is the maximum length of $\mathcal{I}$ plus one. Here b is some absolute constant that we will fix shortly.

When a reconstruction takes place at some location j of $\mathcal{I}$, as prescribed above, its cost can be covered by at most $t = |\overline{H}_j|$ credits.

We first allocate bw credits to the newly inserted plane h. Then we show that the total credit of the planes in the data structure decreases by at least t, covering the cost of the reconstruction. This implies that the total amortized cost of the insertion is the allocation of the bw new credits to h, which amount to $ba \log^3 n$ units.

We use the following observations regarding the planes in $\overline{H}_j$ to lower bound the total credit decrease following the reconstruction.

- As in Equation (8), $|\overline{H}_j| \leq 2^j$.
- First, by the lower bound in Invariant (1), at least $\sum_{i=1}^{j-2} 2^{i-1} + 1 = 2^{j-2}$ of the planes in $\overline{H}_j$ were stored, before the reconstruction, in structures $\overline{\mathcal{D}}_i$, for $i \leq j-2$. insertion, in this sum). That is, at least a quarter of the planes in $\overline{H}_j$ are from a location $\leq j-2$.
- By Lemma 7.1, after the reconstruction, at least $(1-\beta)$ of the planes in $\overline{H}_j$ are stored in the first substructure $\mathcal{D}^{(1)}$ of the resulting static structure, which we place at location j or $j-1$.

Let $f \leq \beta|\overline{H}_j| \leq \beta t$ be the number of planes that were at location $j-1$ before the reconstruction, and were pruned away during the construction of $\mathcal{D}^{(1)}$. It follows from the observations above that at least $\left(\frac{1}{4} - \beta\right) t + f$ planes that were stored at locations $\leq j-2$ before the reconstruction have not been pruned away during the construction of $\mathcal{D}^{(1)}$. (Indeed, we had at least $t/4$ planes stored at at locations $\leq j-2$, and we have pruned away a total of at most βt planes, of which f came from location $j-1$.) Since we place $\mathcal{D}^{(1)}$ either at location j or at location $j-1$, all these planes are stored either at location j or at location $j-1$ following the reconstruction. Hence, the credit of each of these planes has decreased by at least b, so in total they release at least $\left(\frac{1}{4} - \beta\right) bt + bf$ credits.

At most f planes from location $j-1$ are moved down to location $j-2$, thereby requiring a total of bf additional credits. By Lemma 7.1, the number of planes that end up at position $j-3$

is at most $\beta^2 t$, and, in general the number of planes that end up at position $j-i$ is at most $\beta^{i-1}t$. These planes require at most

$$\beta^2 t \cdot 2b + \beta^3 t \cdot 3b + \beta^4 t \cdot 4b + \cdots \leq 2\beta^2 bt \left(1 + \frac{3}{2}\beta + \frac{4}{2}\beta^2 + \ldots\right) \leq 2\beta^2 bt \sum_{i=0}^{\infty} (2\beta)^i \leq \frac{2\beta^2}{1-2\beta} bt$$

additional credits.

Summing up the credit changes of all planes we get that the total credit decreases by at least

$$\left(\frac{1}{4} - \beta\right) bt + bf - \left(bf + \frac{2\beta^2}{1-2\beta} bt\right) = \frac{1-6\beta}{4(1-2\beta)} bt$$

which is larger than t for $b > \frac{4(1-2\beta)}{1-6\beta}$. That is, we have shown that the number of credits has decreased by at least t, which suffices to pay for the reconstruction. □

7.3 Handling deletions

We support deletions by adding a *deletion lookahead mechanism* to each structure $\overline{\mathcal{D}}_j$. This is implemented as follows. When we delete a plane h, we go over each substructure $\overline{\mathcal{D}}_j$ that has a conflict list containing h, and remove h from each of the $O(\log n)$ conflict lists of $\overline{\mathcal{D}}_j$ that it belongs to. In total, we remove h from at most $O(\log^2 n)$ conflict lists within the entire $\mathcal{I}$. When the size of such a list C_τ becomes too small, the surviving planes in C_τ might show up on the new lower envelope of H, and thus might be the answer to some query, even though they might not necessarily belong to the conflict list of any prism at the lowest k_0-level, in which case our mechanism might fail to report them. To avoid this situation, we delete *all* the surviving planes from C_τ. So as not to lose the planes involved in these additional artificial deletions, we re-insert them into the structure, using the insertion mechanism described above. The precise rule for "emptying out" a conflict list C_τ is to do so as soon as its size becomes $\leq (1 - 1/2\alpha)$ its original size, where $\alpha > 1$ is our cutting parameter (so that each prism of Λ_k intersects at most αk planes), as in the static construction. This description pertains to all the prisms, including those at the k_0-level.

We say that h was *lookahead deleted* (by the deletion lookahead mechanism) from a structure $\overline{\mathcal{D}}_j$ if it belonged to some annihilated conflict list of $\overline{\mathcal{D}}_j$.[15] A plane h that was lookahead deleted from $\overline{\mathcal{D}}_j$ may still appear in other conflict lists of $\overline{\mathcal{D}}_j$. Therefore h may be lookahead deleted from $\overline{\mathcal{D}}_j$ many times, once per conflict list of $\overline{\mathcal{D}}_j$ containing it. Note also that the conflict lists of $\overline{\mathcal{D}}_j$ may contain planes that are not *stored* in $\overline{\mathcal{D}}_j$ (that is planes that are not in $H(\overline{\mathcal{D}}_j)$), which may be lookahead deleted as well. To economize, we do not re-insert these planes into the structure, as they are already stored, by construction, in another structure $\overline{\mathcal{D}}_i$, for some $i < j$. Furthermore, we can mark a plane $h \in H(\overline{\mathcal{D}}_j)$ when it is lookahead deleted for the first time from a conflict list of $\overline{\mathcal{D}}_j$. Then, when we purge additional conflict lists of $\overline{\mathcal{D}}_j$ containing h, we also refrain from re-inserting h into the data structure.

In spite of these refinements, a plane h may be stored (contained in $H(\overline{\mathcal{D}}_j)$) at multiple locations j. (This happens when we lookahead delete a plane $h \in H(\overline{\mathcal{D}}_j)$ (for the first time) from $\overline{\mathcal{D}}_j$. The reinsertion of h will place it at another $h \in H(\overline{\mathcal{D}}_i)$ for some $i < j$.) We maintain the invariant that

[15] To avoid confusion, we emphasize that there are three kinds of removals of planes that occur in the algorithm: *pruning* in the static construction, *lookahead deletions*, as described now, and real *deletions*, of planes from H, to which we refer as deletions *by the adversary*.

h is not lookahead deleted from the structure $\overline{\mathcal{D}}_j$ of lowest index j containing it, and observe that h always belongs to the corresponding set $H(\overline{\mathcal{D}}_j)$.

Recall that an insertion (or reinsertion) of a plane h takes a contiguous prefix $\mathcal{I}'$ of $\mathcal{I}$ of length j, discards the existing structures in $\mathcal{I}'$, sets $\overline{H}_j := \left(\bigcup_{i=0}^{j-1} H(\overline{\mathcal{D}}_i)\right) \cup \{h\}$, constructs a new static structure for $\overline{H}_j$, and spreads its components along $\mathcal{I}'$.

The correctness of the data structure is a consequence of the following Invariant.

Invariant 7.4. *Let h be a plane that is stored in the current $\mathcal{I}$, and that shows up on the lower envelope of the entire current set H over some point q in the xy-plane. Let $\overline{\mathcal{D}}_j$ be a substructure from which h was not deleted by the lookahead deletion mechanism. Then h belongs to the conflict list C_τ of the prism τ of the lowest cutting Λ_0 of $\overline{\mathcal{D}}_j$, whose xy-projection contains q.*

Proof. Assume to the contrary that h does not belong to the conflict list C_τ of the corresponding prism τ of the lowest level of $\overline{\mathcal{D}}_j$. Let q^+ denote the point on h that is co-vertical with q (by assumption, q^+ lies on the lower envelope of H). By assumption, 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 $\overline{\mathcal{D}}_j$.

Let τ' be the prism of Λ_t such that q^+ lies above its ceiling $\overline{\tau}'$. At the time τ' was constructed, each of the vertices of $\overline{\tau}'$ was at some level larger than k_t and smaller than αk_t in the arrangement $\mathcal{A}(H_t)$ of some subset H_t of H. At this time, at least k_t planes of H_t passed below q^+. Indeed, Λ_t was a shallow cutting of $\mathcal{A}_{\leq k_t}(H_t)$ at the time of its construction, so $\overline{\Lambda}_t$ passed fully above the k_t-level of $\mathcal{A}(H_t)$. This subset of H_t, denoted as C_{q^+}, is not necessarily a subset of the actual conflict list $C_{\tau'}$, as some of the planes C_{q^+} may have been deleted by the pruning mechanism of the static structure, as it processed level k_t, and moved to the next substructure of the relevant static structure. Nevertheless, since q^+ now belongs to the lower envelope of H, all the (at least k_t) planes of C_{q^+} must have been deleted (by the adversary) from H.

Consider now the prism τ'' of Λ_{t+1} that contains q^+ (letting τ'' be the entire $\mathbb{R}^3$ if $t = m$). Since C_{q^+} is contained in H_t, none of its planes was pruned away (by the static mechanism) during the preceding processing of Λ_{t+1}, and consequently all of them are actually stored at $C_{\tau''}$ (all of them certainly cross τ''). (For the case $t = m$, this simply means that all the planes are present at the beginning of the process.) The original size of $C_{\tau''}$ is at most $\alpha k_{t+1} \leq 2\alpha k_t$, and, by the time q^+ has reached the lower envelope of H, at least

$$|C_{q^+}| \geq k_t \geq \frac{|C_{\tau''}|}{2\alpha}$$

of them have been deleted (by the adversary). By the deletion lookahead mechanism, we must have then lookahead deleted h before this has occurred, which is a contradiction. □

The following lemma analyzes the performance of the data structure.

Lemma 7.5. *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 $O(\log^2 n)$, where n is the size of H at the time the operation takes place.*

Proof. The bound on the cost of the query is proved as in the proof of Lemma 7.3.

Consider insertions. We have to modify the analysis in Lemma 7.3 since $\overline{H}_j := \left(\bigcup_{i=0}^{j-1} H(\overline{\mathcal{D}}_i)\right) \cup \{h\}$ does not contain duplicates and therefore the lower bound on its size, given in Equation (8), may not hold.

Consider the size of $\overline{H}_j$ before and after removing duplicate planes. We split the analysis into two cases: If, following the removal of duplicate planes, the size of $\overline{H}_j$ does not shrink substantially, say it only shrinks by a factor of at most $1/8$, then the analysis in Lemma 7.3 stays valid with a minor adaptation of the constants. If at least $1/8$ the planes of $\overline{H}_j$ are duplicate planes, then these planes have at least $\frac{1}{8}b|\overline{H}_j| = \frac{1}{8}bt$ credits which are now freed and, for b sufficiently large, suffice to pay for the reconstruction and to allocate sufficient credit to the remaining planes that move to substructures of lower index as a result of the reconstruction.

Finally, we analyze the amortized cost of a deletion. When we delete (that is, the adversary deletes) a plane h, we remove it from at most $O(\log^2 n)$ conflict lists, as already argued above. Each removal of h from such a conflict list may trigger the annihilation of the entire list, accompanied with the re-insertion of each of its survivors back into $\mathcal{I}$. For the analysis of the amortized cost of a deletion, each time a plane h is removed from some conflict list, without causing the list to be purged, we let h deposit in the list $\Theta(\log^3 n)$ units. In total, h deposits $\Theta(\log^5 n)$ units. When a conflict list C_τ gets purged (by the lookahead mechanism) at the deletion of a plane h, at least $\frac{1}{2\alpha}$ of its original elements have already been deleted (by the adversary), so it has accumulated $\Omega\left(\frac{1}{2\alpha}|C_\tau|\log^3 n\right)$ units, where here $|C_\tau|$ refers to the original size of the list. With a suitable choice of the constant of proportionality, these units suffice to pay for the (amortized) cost of re-insertion of the remaining elements of C_τ. □

Storage. The entire structure, as described so far, requires $O(n \log n)$ storage. Indeed, $\mathcal{I}$ consists of $O(\log n)$ substructures, where the structure $\overline{\mathcal{D}}_j$ at index j, if nonempty, is a static substructure that has originally been constructed for some set $\overline{H}_j$ of at most 2^j planes. Such a substructure is a hierarchy of cuttings, each approximating some level in a geometric sequence of levels. Put $n_j := |\overline{H}_j| \leq 2^j$. The number of prisms in the cutting for level k is $O(n_j/k)$, and the size of the conflict list of each of its prisms is $O(k)$, so the total storage for each level of $\overline{\mathcal{D}}_j$ is $O(n_j)$, for a total storage of $O(n_j \log n_j)$. Summing over j, we get that the total storage used by $\mathcal{I}$ is $O(n \log n)$. (Note that a similar analysis shows that the total storage required by $\mathcal{I}$, excluding the conflict lists, is only $O(n)$.)

Following an idea of Chan [13], we can improve the storage to linear, if we do not store explicitly the conflict list of each prism, but only its size (except for the lowest-level prisms, but the size of each list at that level is only $O(1)$). It is easily seen (along the comment just made) that with this approach the total storage for $\overline{\mathcal{D}}_j$ is $O(n_j)$, making the overall storage $O(n)$.

To make this reduction effective, we need additional mechanisms that will compensate for the missing conflict lists. Specifically, when we delete a plane h, we need to find the prisms at which it would have been stored, and decrement the counter (size of the conflict list) of each such prism. Proceeding naively, within a substructure $\overline{\mathcal{D}}_j$, 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 of these prisms. However, h might lie below a vertex v and not belong to the conflict lists of the incident prisms, because it has been pruned away while processing a previous (higher) level. To handle this issue, we augment $\overline{\mathcal{D}}_j$ with a halfspace range reporting data structure for the set of vertices of each of its cuttings separately. We use the recent algorithm of Afshani and Chan [1] (which can be made deterministic by using the shallow cutting construction of [15]), 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 query plane h can be reported in $O(\log|V| + t)$ time, where t is the output size. The cost of augmenting $\overline{\mathcal{D}}_j$ with these reporting structures is

subsumed by the cost of building $\overline{\mathcal{D}}_j$ itself.

Now, when deleting a plane h, we access each substructure $\overline{\mathcal{D}}_j$ of $\mathcal{I}$ for which h belongs to some conflict list of $\overline{\mathcal{D}}_j$. For this, each plane h stores pointers to all these structures. Since the overall size of the sets $\overline{H}_j$ is $O(n)$, the overall number of pointers is also $O(n)$. For each substructure $\overline{\mathcal{D}}_j$ that contains h, we find the prisms that contain h in their conflict lists. To ensure correctness of this step, h also stores a second pointer, for each $\overline{\mathcal{D}}_j$ 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 from the level at which h was pruned and higher, and retrieves from each of these structures the prisms that contain it in their conflict lists. For each such prism τ, we decrement its counter by 1. If the counter becomes too small, according to the criterion given above, we purge the entire conflict list, and reinsert its surviving members into $\mathcal{I}$ (this step of course also requires an alternative 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 $\overline{\mathcal{D}}_j$, 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, the step that purges conflict lists that have become too small also requires a mechanism to compute the conflict lists to be purged. Specifically, when we purge the conflict list of a prism τ in the structure $\overline{\mathcal{D}}_j$, we need to retrieve the planes of C_τ that have to be re-inserted. To do so, we preprocess the planes of $H(\overline{\mathcal{D}}_j)$ into a (dual version of a) halfspace reporting data structure that we keep with $\overline{\mathcal{D}}_j$. We query this structure with each of the four vertices of τ, to obtain, in an output-sensitive manner, all the planes of $H(\overline{\mathcal{D}}_j)$ that cross τ. This structure takes space linear in $|H(\overline{\mathcal{D}}_j)|$, $O(|H(\overline{\mathcal{D}}_j)| \log |H(\overline{\mathcal{D}}_j)|)$ time to build, and can answer a query in $O(\log |H(\overline{\mathcal{D}}_j)| + t)$ time, where t is the output size. The cost of answering such a query is subsumed by the cost of reinserting the planes of C_τ, and the cost of constructing this reporting structure is subsumed by the cost of constructing $\overline{\mathcal{D}}_j$.

As we mentioned before, on top of the maintenance mechanism described so far, we rebuild the entire structure when the number of planes in H increases or decreases by a factor of 2. The cost of this rebuilding, namely $O(n \log^2 n)$, is subsumed by the cost of the insertions and/or deletions that have been performed since the last global rebuilding.

We thus obtain the following main summary result of this section.

Theorem 7.6. *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 generalize the data structure from the previous section for general surfaces. As mentioned in the introduction, the key observation is that Chan's technique, as well as our improvement thereof, 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 by our algebraic

model of computation—see the introduction for details), the rest of the algorithm is a purely symbolic organization and manipulation of the given surfaces into standard data structures. Indeed, the whole geometry needed for the deletion lookahead mechanism is encapsulated in the proof of Invariant 7.4, which relies only on the properties of conflict lists in a vertical shallow cutting.

We begin by showing how to find a *vertical* shallow cutting *with* conflict lists as needed for Chan's construction.

Theorem 8.1. *Let F be a set of n continuous totally defined algebraic functions of constant description complexity for which the complexity of the lower envelope of any m functions is $O(m)$. Furthermore, let $k \in \{1, \ldots, n\}$. Then, there is a vertical shallow cutting Λ_k for the first k levels of $\mathcal{A}(F)$ with the following properties:*

1. *The number of cells in Λ_k is $O((n/k)\log^2 n)$.*
2. *Each prism in Λ_k intersects at least k and at most $2k$ graphs of functions in F.*
3. *We can find Λ_k and the conflict lists for its prisms in expected time $O(n\log^3 n\lambda_{s+2}(\log n))$ using expected space $O(n\log n\lambda_{s+2}(\log n))$.*

Proof. We combine the techniques from Sections 4, 5 and 6. First, set $\varepsilon = 1/2$ and $\lambda = 4c\log n$, for a suitable constant c as in Section 4. Pick t randomly in $\left[\frac{7}{6}, \frac{5}{4}\right]\lambda$, 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 $[k, 2k]$. 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 6.3, this step takes expected time $O(nt\lambda_{s+2}(t)\log(n/t)\log n) = O(n\log^3 n\lambda_{s+2}(\log n))$ and expected space $O(nt\lambda_{s+2}(t)) = O(n\log n\lambda_{s+2}(\log n))$. As a result, we get the vertical decomposition $\mathrm{VD}_{\leq t}(S_k)$ of the $(\leq t)$-level of $\mathcal{A}(S_k)$ 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 lie on the t level. If the complexity of $\overline{T}_k$ exceeds its expectation by a constant factor that is too large, 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. If τ is incident to $\overline{T}_k$, we 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 very complex). The total time for this step is proportional to the total size of the conflict lists of $\mathrm{VD}_{\leq t}(S_k)$, which is $O(n\log n\lambda_{s+2}(\log n))$ in expectation, by Theorem 6.3.

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 performing a vertical sweep on the xy-projection of $\overline{T}_k$. By Lemma 5.1, the downward vertical extension of $\overline{\Lambda}_k$, Λ_k, is a shallow cutting for the first k levels of $\mathcal{A}(F)$, with high probability. To find the conflict lists 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 and a walk in $\overline{\Lambda}_k$ to find all trapezoids of $\overline{\Lambda}_k$ that are intersected by $f \cap \overline{T}_k$. Starting from these trapezoids, we perform another

walk in $\overline{\Lambda}_k$ to find all trapezoids of $\overline{\Lambda}_k$ that lie above f. For all these trapezoids, f is in the conflict list of the corresponding vertical prism, so 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 location overhead. Thus, 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, 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. □

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 we construct are of slightly sub-optimal size ($O((n/k)\log^2 n)$ instead of $O(n/k)$) and that we need more time to construct them ($O(n\log^3 n\lambda_{s+2}(\log n))$ instead of $O(n\log n)$). Specifically, the following adjustments are necessary: 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 $O(n\log^4 n\lambda_{s+2}(\log n))$ (in expectation). The query time remains $O(\log^2 n)$, since $O(\log n)$ time point location is also possible in general minimization diagrams. Concerning insertions, the increased construction time implies that in Lemma 7.3, we need to allocate $\Theta(\log^4 n\lambda_{s+2}(\log n))$ units for one credit. Then, the remaining analysis in the proof of Lemma 7.3 remains valid, and we have an amortized insertion cost of $O(\log^5 n\lambda_{s+2}(\log n))$. Finally, we need to assess the effect on the deletion cost: since now a deleted element can appear in $O(\log^4 n)$ conflict lists, and since a reinsertion now requires $O(\log^5 n\lambda_{s+2}(\log n))$ units, we must equip each deleted element with $\Theta(\log^9 n\lambda_{s+2}(\log n))$ units to pay for the lookahead deletions. Since the storage for the dynamic structure is proportional to the storage for the static structure, our structure needs $O(n\log^3 n)$ space overall.

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

Theorem 8.2. *The lower envelope of a set of n bivariate functions of constant description complexity in three dimensions can be maintained dynamically, so as to support insertions, deletions, and queries, so that each insertion takes $O(\log^5 n\lambda_{s+2}(\log n))$ amortized expected time, each deletion takes $O(\log^9 n\lambda_{s+2}(\log n))$ amortized expected time, and each query takes $O(\log^2 n)$ worst-case deterministic time, where n is the maximum size of the set of functions during the process. The data structure requires $O(n\log^3 n)$ storage in expectation.*

9 Applications

Let $S \subset \mathbb{R}^2$ be a finite set of parwise disjoint sites, each being a simply-shaped convex region in the planhe, e.g., points, line segments, disks, etc. Finding for a point $q \in \mathbb{R}^2$ its nearest neighbor in S under any norm or convex distance function δ [19] translates to ray shooting in the lower envelope of the graphs of the functions $f_s(x) = \delta(x, s)$ for $s \in S$. Let F denote the set of these functions. Thus, if the lower envelope of F has linear complexity, Theorem 8.2 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, 22].

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 $p \in [1, \infty]$. Then we define for $(x_1, y_1), (x_2, y_2) \in \mathbb{R}^2$ the L_p metric $\delta_p((x_1, y_1), (x_2, y_2)) = (|x_1 - x_2|^p + |y_1 - y_2|^p)^{1/p}$. It is well known that δ_p is a metric, and thus, like any metric, it induces lower envelopes of linear complexity for any set of sites as above [33].
- **Additively Weighted Euclidean Metric**: Let $S \subset \mathbb{R}^2$ be a set of 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].

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, 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 defined as 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 [23] with the dynamic lower envelope structure from Theorem 8.2.

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 $O(\log^{10} n \lambda_{s+2}(\log n))$ amortized expected time per insertion and $O(\log^{11} n \lambda_{s+2}(\log n))$ amortized expected time per deletion.*

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 $O(n^{2+\varepsilon})$, building on a trick by Vaidya [43]. The essence of the algorithm lies in a dynamic bichromatic closest pair data structure for the additively weighted Euclidean metric. The algorithm makes $O(n^2)$ updates to this structure. Thus, using Theorem 9.1, we get the following improvement:

Theorem 9.2. *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 $O(n^2 \log^{11} n \lambda_{s+2}(\log n))$ expected time.*

Dynamic Minimum Spanning Trees. Following Eppstein [23], Theorem 9.1 immediately gives a data structure for dynamic maintenance of minimum spanning trees. We thus get the following improved version of Theorem 6.9 in Agarwal et al. [3].

Theorem 9.3. *A minimum spanning tree of a set of at most n points in the plane, under any L_p-metric, can be maintained in $O(\log^{13} n\lambda_{s+2}(\log n))$ per update, using $O(n\log^5 n)$ space.*

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]

Theorem 9.4. *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 or deletion takes $O(\log^5 n\lambda_{s+2}(\log n))$ or $O(\log^9 n\lambda_{s+2}(\log n))$ 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 if $p\in B^\cap$, 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 $C\subseteq\mathcal{C}$ such that at any point 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]:

Theorem 9.5. *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 $O(\log^5 n\lambda_{s+2}(\log n))$ amortized expected time and a deletion takes $O(\log^9 n\lambda_{s+2}(\log n))$ 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 from Agarwal et al. [3, Theorem 6.8]. By plugging in our improved Theorem 9.1, we get the following result.

Theorem 9.6. *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 $O(n\log^{11} n\lambda_{s+2}(\log n))$.*

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 s to t if and only if $|st|\leq w_s+w_t$. We assume that all weights lie in the interval $[1,\Psi]$, for some $\Psi\geq 1$, and we call Ψ the *radius ratio*. We show how to dynamically maintain $D(S)$ under insertions and deletions of vertices, 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 update time will be $O(\Psi^2\log^8 n)$ for insertions and $O(\Psi^2\log^{12} n)$

for deletions, and the 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 [14].

Our approach is as follows: let $\mathcal{G}$ be a planar grid whose cells are 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 $\sigma \cap S$ induce a clique in $D(S)$. For $S \subset \mathbb{R}^2$, we define an abstract graph G whose vertices are the *non empty* cells $\sigma \in \mathcal{G}$, i.e., the cells with $\sigma \cap S \neq \emptyset$. The *neighborhood* $\mathrm{N}(\sigma)$ of a cell $\sigma \in \mathcal{G}$ is the $\left(\lceil 4\sqrt{2}\Psi\rceil + 3\right) \times \left(\lceil 4\sqrt{2}\Psi\rceil + 3\right)$ block of cells in $\mathcal{G}$ with σ in the center. We call two cells *neighboring* if they are in each other's neighborhood. Thus, the endpoints of any edge in $D(S)$ must lie in neighboring cells. We pick the following edges for G: consider any two 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 cells in G:

Lemma 9.7. *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 a 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 with respect to *edge* updates [29].

Theorem 9.8 (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 in 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 9.8 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 non empty grid cells changes by a factor of 2. When a site s is inserted into or deleted from S, only $O(\Psi^2)$ 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 time $O(\Psi^2 \log^2 n)$, by Theorem 9.8. 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 [23]. This is defined as follows: let $R \subseteq S$ and $B \subseteq S$ be two sets of sites. A MBM M between R and B is a maximal set of edges in $(R \times B) \cap D(S)$ that form a matching. Using our dynamic lower envelope structure from Theorem 8.2, we can easily maintain MBMs.

Lemma 9.9. *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 $D(R \cup B)$ such that we can insert or delete sites in expected amortized time $O(\log^9 n \lambda_{s+2}(\log n))$.*

Proof. We have two dynamic lower envelope structures, one for R and one for B, as in Theorem 8.2, with the weighted distance function $\delta(p, s) = |ps| - w_s$, that allow us to perform nearest neighbor search (i.e., vertical ray shooting at the lower envelope) with respect to δ. 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, the insertion procedure maintains a 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. The procedures for updating B are analogous.

Since inserting and deleting sites requires $O(1)$ insert, delete or query operations in NN_R or NN_B, the lemma follows. □

We create a data structure as in Lemma 9.9 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 cells in $\mathrm{N}(\sigma)$. 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 obtain the following theorem:

Theorem 9.10. *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 $O(\Psi^2 \log^9 n \lambda_{s+2}(\log n))$ and (ii) we can determine for any pair of sites s, t whether they are connected by a path in $D(S)$, in time determistic worst-case time $O(\log n / \log\log n)$.*

As stated above, prior to this study, polylogarithmic bounds were 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 $O(\log^{10} n)$ [14]. An extension of our method leads to significantly improved bounds for this case, too. Namely, for this case, we can obtain an amortized update time $O(\log^2 n)$ with worst-case query time $O(\log n / \log\log n)$ and amortized update $O(\log n \log\log n)$ with worst-case query time $O(\log n)$ [31].

9.3 Breadth-First-Search in Disk Graphs

As observed by Roditty and Segal [40] 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 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 of the BFS-algorithm, the dynamic nearest neighbor data structure contains all sites that are not yet part of the BFS-tree. To find all 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 $D(S)$. The successful queries can be charged to the edges of the BFS-tree, the last unsuccessful query can be charged to p. Thus, the total number of operations on the data structure is $O(n)$. We get the following theorem:

Theorem 9.11. *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 $O(n \log^9 n \lambda_{s+2}(\log n))$.*

9.4 Spanners for Disk Graphs

Finally, we discuss how to use our data structure in order to compute efficiently a *spanner* for the disk graph of a given weighted point set. We only sketch our approach and leave further details for the full version. 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 [25] show that a simple construction based on the *Yao graph* [44] yields a $(1+\varepsilon)$-spanner for $D(S)$ with $O(n/\varepsilon)$ edges: let $\mathcal{C}$ be a set of $k = O(1/\varepsilon)$ cones 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 [25, 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 show how to 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 weights 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* [30]. 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 for t in C. Again it is not clear how to obtain this spanner efficiently. To solve this problem, Kaplan et al. [30] proceed as Fürer and Kasiviswanathan and describe several 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 $O(n\log^5 n)$ is as follows: we compute a *compressed quadtree* T for S [26]. 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 9. Kaplan et al. show that this augmentation requires adding $O(n)$ additional nodes to T that 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.

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.

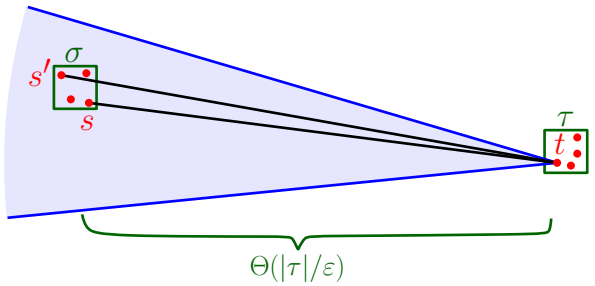

Figure 9: A cone C (blue) and the shortest edge st in this cone. Any edge $s't$ with $s' \in \sigma$ has approximately the same length as st.

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 is relevant for 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. [30]. They prove that the running time is dominated by the time needed for $O(n \log n)$ insertions and $O(n/\varepsilon^2)$ deletions in the dynamic nearest neighbor structure.

Now, a similar strategy works for the case of disk graphs. We sketch how to modify the approach above. 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 in NN_τ yet. 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 $w_s \leq w_t \leq w_{s'}$). When we have finished processing W_σ, we proceed with the next cell σ'. To ensure that our nearest neighbor queries still returns only smaller disks, we need to delete all sites in NN_τ whose weight is larger than the smallest weight in W'_σ. This can be done by deleting all sites in W_τ from NN_τ. 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. [30] 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 8.2, we thus obtain the following result:

Theorem 9.12. *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 $O((n/\varepsilon^2) \log^9 n \lambda_{s+2}(\log n))$.*

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 Journal on Computing*, 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 Journal on Computing*, 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, 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, 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, third edition, 2008.

[9] Jean-Daniel Boissonnat and Monique Teillaud. *Effective computational geometry for curves and surfaces*. Mathematics and Visualization. Springer-Verlag, 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 Journal on Computing*, 30(2):561–575, 2000.

[12] Timothy M. Chan. Low-dimensional linear programming with violations. *SIAM Journal on Computing*, 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):Art. 16, 15, 2010.

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

[15] Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. In *Proc. 31st Int. Sympos. Comput. Geom. (SoCG)*, pages 719–732, 2015.

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

[17] Bernard Chazelle. *The discrepancy method*. Cambridge University Press, 2000.

[18] 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.

[19] 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.

[20] Kenneth L. Clarkson. A randomized algorithm for closest-point queries. *SIAM Journal on Computing*, 17(4):830–847, 1988.

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

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

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

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

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

[26] Sariel Har-Peled. *Geometric approximation algorithms*, volume 173 of *Mathematical Surveys and Monographs*. American Mathematical Society, 2011.

[27] Sariel Har-Peled, Haim Kaplan, and Micha Sharir. Approximating the k-level in three-dimensional plane arrangements. In *Proc. 27th Annu. ACM-SIAM Sympos. Discrete Algorithms (SODA)*, 2016.

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

[29] 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.

[30] 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.

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

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

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

[34] 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.

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

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

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

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

[39] 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.

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

[41] 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.

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

[43] Pravin M. Vaidya. Geometry helps in matching. *SIAM Journal on Computing*, 18(6):1201–1225, 1989.

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