Dynamic Maximum Depth of Geometric Objects

Subhash Suri
Department of Computer Science, University of California, Santa Barbara, CA, USA

Jie Xue
New York University Shanghai, China

Xiongxin Yang
New York University Shanghai, China

Jiumu Zhu
New York University Shanghai, China

Abstract

Given a set of geometric objects in the plane (rectangles, squares, disks etc.), its maximum *depth* (or *geometric clique*) is the largest number of objects with a common intersection. In this paper, we present data structures for dynamically maintaining the maximum depth under insertions and deletions of geometric objects, with sublinear update time. We achieve the following results:

- a $\frac{1}{k}$-approximate dynamic maximum-depth data structure for (axis-parallel) rectangles with $O(n^{1/(k+1)} \log n)$ amortized update time, for any fixed $k \in \mathbb{Z}^+$. In particular, when $k = 1$, this gives an exact data structure for rectangles with $O(\sqrt{n} \log n)$ amortized update time, almost matching the best known bound for the offline version of the problem.
- a $(\frac{1}{2} - \varepsilon)$-approximate dynamic maximum-depth data structure for disks with $n^{2/3} \log^{O(1)} n$ amortized update time, for any constant $\varepsilon > 0$. Having exact data structures for disks with sublinear update time is unlikely, since the static maximum-depth problem for disks is 3SUM-hard and thus does not admit subquadratic-time algorithms.

2012 ACM Subject Classification Theory of computation → Computational geometry; Theory of computation → Design and analysis of algorithms

Keywords and phrases dynamic algorithms, maximum depth

Digital Object Identifier 10.4230/LIPIcs.SoCG.2025.77

1 Introduction

We consider the problem of *dynamically* maintaining a set $\mathcal{O}$ of geometric objects (rectangles, squares, disks etc.) under insertions and deletions so that at any instant we can find its maximum depth. The *maximum depth* of $\mathcal{O}$ is the largest depth of any point in the plane, where the depth of a point p, denoted $\mathsf{dep}(p, \mathcal{O})$, is defined as the number of objects in $\mathcal{O}$ containing p. The maximum depth of the set $\mathcal{O}$, written as $\mathsf{dep}(\mathcal{O}) = \max_{p \in \mathbb{R}^d} \mathsf{dep}(p, \mathcal{O})$, is equivalent to its *maximum geometric clique*, namely, the largest subset of objects in $\mathcal{O}$ with a nonempty common intersection. It is easy to see that for axis-parallel rectangles and squares in the plane the geometric cliques correspond to graph-theoretic cliques in the *intersection graph* of $\mathcal{O}$ because the Helly number of rectangles is two. This equivalence, however, does not hold for general geometric objects (even if the objects are convex), where the size of the maximum geometric clique can be smaller than the size of the maximum clique in the intersection graph. Nevertheless, in some cases, the maximum depth is closely related to the maximum clique size in the intersection graph. For example, it is well-known that for disks, the maximum depth is at least $\frac{1}{4}$ of the size of the maximum clique [8].

The problem of computing a maximum geometric clique for simple planar shapes such as rectangles, squares, or disks (and their counterparts in d-dimensions) is well-studied in computational geometry, and several efficient algorithms are known [14, 2, 9, 12]. The primary focus of our work is to *dynamically* maintain the maximum depth as well as a point p^* that realizes this depth, i.e., $\mathsf{dep}(p^*, \mathcal{O}) = \mathsf{dep}(\mathcal{O})$. To the best of our knowledge, the fully dynamic version of the maximum-depth problem has not been studied before, except for the case of 1-dimensional intervals on the line, where it can be easily solved optimally with $O(\log n)$ update time [14].

The maximum-depth problem in two dimensions provides an interesting setting for dynamic data structures: unlike the NP-complete problems such as set cover, hitting set or independent set for rectangles, which have been the focus of recent work in dynamic data structures [13, 10, 1, 7, 11, 6], the *static* maximum depth problem for rectangles is polynomial-time solvable [14]. Yet, in attempting to design a dynamic data structure for the maximum depth, we face challenges similar to those in set cover etc, and sometimes have to settle for *approximation* to achieve fast update time. For example, for the maximum depth of (axis-parallel) rectangles, it is very difficult to achieve an $O(n^c)$ update time for $c < 1/2$, since the best known static algorithm requires $n^{1.5}/\log^{O(1)} n$ time [9]. Similarly, for the maximum depth of disks, a sublinear update time is unlikely even in the insertion-only setting (if we want exact depth), because the static problem is known to be 3SUM-hard [3] and thus does not admit a (truly) subquadratic-time algorithm by conjecture.

Another challenge for dynamic maximum depth is the non-decomposability of the problem. Specifically, even if we can partition the set $\mathcal{O}$ of geometric objects as $\mathcal{O} = \mathcal{O}_1 \cup \mathcal{O}_2$, the value of $\mathsf{dep}(\mathcal{O})$ cannot be computed from $\mathsf{dep}(\mathcal{O}_1)$ and $\mathsf{dep}(\mathcal{O}_2)$. This non-decomposability makes many techniques in data structure designing inapplicable to the maximum-depth problem.

1.1 Our results

In this paper, we study the dynamic maximum-depth problem for (axis-parallel) rectangles and disks, designing exact and approximate dynamic maximum-depth data structures with sublinear update time. Formally, a dynamic maximum-depth data structure maintains a set $\mathcal{O}$ of geometric objects in $\mathbb{R}^2$ (or more generally $\mathbb{R}^d$) under insertions and deletions, and can report a point $p^* \in \mathbb{R}^2$ satisfying $\mathsf{dep}(p^*, \mathcal{O}) = \mathsf{dep}(\mathcal{O})$ together with the value of $\mathsf{dep}(\mathcal{O})$.

Our main result for rectangles is the following theorem.

▶ **Theorem 1.** *For every $k \in \mathbb{Z}^+$, there exists a $\frac{1}{k}$-approximate dynamic maximum-depth data structure for axis-parallel rectangles with $O(n^{1/(k+1)} \log n)$ amortized update time.*

When $k = 1$, the above theorem gives us an exact data structure for dynamic maximum depth of rectangles, with $O(\sqrt{n} \log n)$ update time. Up to logarithmic factors, this bound matches the update time of the best known *offline* dynamic data structure for the problem [9], which is $O((\sqrt{n}/\sqrt{\log n}) \cdot \log^{3/2} \log n)$. Indeed, $\widetilde{O}(\sqrt{n})$ update time may be best one can expect, even in the insertion-only version, because it is a longstanding open problem whether the static maximum-depth problem for rectangles can be solved in $O(n^{1.5-\varepsilon})$ time. The data structure in Theorem 1 can also be used to dynamically maintain the maximum clique of a rectangle graph with the same update time, because the maximum clique size of a rectangle intersection graph equals the maximum depth of the rectangles.

Our main result for disks is the following theorem.

▶ **Theorem 2.** *For any constant $\varepsilon > 0$, there exists a $(\frac{1}{2} - \varepsilon)$-approximate dynamic maximum-depth data structure for disks with $n^{2/3} \log^{O(1)} n$ amortized update time.*

The static maximum-depth problem for disks is known to be 3SUM-hard [3] with a conjectured lower bound of $\widetilde{\Omega}(n^2)$. It is, therefore, unlikely that an exact dynamic maximum-depth data structure for disks with (truly) sublinear update time exists, even in the insertion-only setting.

Theorem 2 also gives a data structure for dynamically maintaining a $(\frac{1}{8}-\varepsilon)$-approximation of the maximum clique size of a disk graph, with the same update time, because the maximum clique size of a disk graph is at most 4 times the maximum depth of the disks [8]. For convenience, we call such a data structure a *dynamic maximum-clique data structure* for disk graphs.

▶ **Corollary 3.** *For any constant $\varepsilon > 0$, there exists a $(\frac{1}{8}-\varepsilon)$-approximate dynamic maximum-clique data structure for disk graphs with $n^{2/3}\log^{O(1)} n$ amortized update time.*

The key idea underlying the data structure of Theorem 2 is to reduce the original problem to dynamic *discrete* maximum depth (for disks), by introducing a $\frac{1}{2}-\alpha$ multiplicative error. In the discrete version of the problem, besides a dynamic set $\mathcal{D}$ of disks, we also have a dynamic set S of points in $\mathbb{R}^2$, and what we want to maintain is $\max_{p\in S} \mathsf{dep}(p,\mathcal{D})$. This kind of reduction works for not only disks, but also general fat objects (where the multiplicative error depends on the fatness of the objects). We believe that this idea is of independent interest and may find applications in other problems as well.

1.2 Related Work

Dynamic data structures have been a focus of research in computational geometry from its very beginning, and there exists a vast literature. Since our work pertains to the dynamic geometric *optimization* problem, we limit our review to the narrow set of papers that have considered related problems. In recent years, a number of papers have addressed the problem of dynamically maintaining such functions as the minimum geometric set cover, minimum geometric hitting set, maximum geometric independent sets, etc., for simple geometric shapes such as rectangles, squares, disks, or their d-dimensional counterpart. In particular, Agarwal et al. [1] were the first to present sub-linear time data structures for maintaining approximate minimum geometric set covers (and hitting sets) of intervals in one dimension and unit squares in two dimensions. The update time bounds were improved and the results generalized to rectangles as well as weighted objects by Chan et al.[10, 11]. Henzinger et al. [13] consider the problem of approximately maintaining the maximum independent set of d-dimensional rectangles. Bhore and Chan [6] present dynamic data structures for a number of other geometric optimization problems such as minimum piercing sets, minimum vertex cover, maximum independent set of rectangles. Yildiz et al. [16] give a data structure to maintain Klee's measure dynamically in a discrete setting.

2 Preliminaries

In this section, we introduce the basic notions used throughout the paper.

Depth. Given a set $\mathcal{O}$ of geometric shapes in $\mathbb{R}^d$, the depth of a point p with respect to $\mathcal{O}$ is the number of objects in $\mathcal{O}$ that contain p. Formally, $\mathsf{dep}(p,\mathcal{O}) = |\{O\in\mathcal{O} : p\in O\}|$. The maximum depth of the set $\mathcal{O}$ is denoted by $\mathsf{dep}(\mathcal{O}) = \max_{p\in\mathbb{R}^d}\mathsf{dep}(p,\mathcal{O})$. We often use p^* to denote a point that maximizes the depth with respect to the set $\mathcal{O}$, i.e., $\mathsf{dep}(p^*,\mathcal{O}) = \mathsf{dep}(\mathcal{O})$, and $\mathcal{O}^*$ to denote a maximum geometric clique for $\mathcal{O}$, i.e., $\mathcal{O}^* = \{O\in\mathcal{O} : p^*\in O\}$.

Disk and fat object. For a disk D, $\mathsf{ctr}(D)$ and $\mathsf{rad}(D)$ denote its center and radius, respectively. This notation also holds for balls. In this paper, the fat object, which generalizes both rectangle and disk, is defined as follows: Let O be a *convex* object in $\mathbb{R}^d$. Let O_{in} be the largest ball that is contained in O. We say O is *α-fat*, for some parameter $\alpha \geq 1$, if O_{out}, the ball that is concentric with O_{in} and has radius $\mathsf{rad}(O_{\text{out}}) = \alpha \cdot \mathsf{rad}(O_{\text{in}})$, contains O. We call the center of O_{in} the *center* of O and $\mathsf{rad}(O_{\text{in}})$ the *in-radius* of O.[1]

3 Dynamic maximum-depth for rectangles

In this section, we consider the dynamic maximum-depth problem for rectangles. We first describe a data structure for maintaining the exact maximum depth with $O(\sqrt{n}\log n)$ amortized update time and $O(n^{1.5}\log n)$ preprocessing time. This proves the base case of Theorem 1, namely, $k = 1$. We then inductively prove the theorem for all $k \in \mathbb{Z}^+$.

Let $\mathcal{Q}$ be a dynamic set of rectangles in $\mathbb{R}^d$, and denote by n the initial size of $\mathcal{Q}$. Our data structure applies the standard reconstruction technique. Specifically, we will reconstruct the data structure periodically: the $(i+1)$-th reconstruction will be done after handling $n_i/2$ operations following the i-th reconstruction, where n_i is the size of $\mathcal{Q}$ at the time of the i-th reconstruction. Thus, during the i-th period, i.e., the period between the i-th reconstruction and the $(i+1)$-th reconstruction, the size of $\mathcal{Q}$ is always $\Theta(n_i)$. Since our data structure will have an $O(n^{1.5}\log n)$ preprocessing time, the time cost for the $(i+1)$-th reconstruction is $O(n_i^{1.5}\log n_i)$, which can be amortized to the $n_i/2$ operations in the i-th period, which is $O(\sqrt{n_i}\log n_i)$ per operation. Because of the reconstruction, we only need to consider the first period, i.e., the first $n/2$ operations.

Let r be a parameter to be determined later. To construct our data structure, we first partition the plane into r horizontal slabs $L_1, \dots, L_r$ using $r-1$ horizontal lines such that each slab contains (in its interior) at most $4n/r$ corners of rectangles in $\mathcal{Q}$. For each slab L_i, we construct a dynamic data structure $\mathcal{D}(L_i)$ that can maintain a point $p_i \in L_i$ that maximizes $\mathsf{dep}(p_i, \mathcal{Q})$, as well as the value of $\mathsf{dep}(p_i, \mathcal{Q})$, under insertions and deletions on $\mathcal{Q}$. We call $\mathcal{D}(L_1), \dots, \mathcal{D}(L_r)$ *slab data structures* for convenience. The design of the slab data structures will be presented shortly. During the update, for each $i \in [r]$ whenever the number of rectangle corners contained in (the interior of) L_i reaches $8n/r$, we split L_i into two horizontal slabs L_i' and L_i'' each of which contains $4n/r$ corners, and replace $\mathcal{D}(L_i)$ with two new slab data structures $\mathcal{D}(L_i')$ and $\mathcal{D}(L_i'')$. In this way, we guarantee that the number of rectangle corners contained in each slab is always $O(n/r)$. We observe that slab splitting can happen at most $O(r)$ times (in the first period).

▶ **Fact 4.** *During the first t operations, the number of slab splittings is at most $2tr/n$.*

Proof. Let $\mathcal{L}$ be the set of all slabs occurred when processing the first t operations. Consider the insertions in the t operations. For each inserted rectangle Q, we charge it to the (at most) two slabs containing the corners of R when Q is inserted. Observe that a slab $L \in \mathcal{L}$ eventually splits only if it gets charged at least n/r times. Indeed, L contains $4n/r$ rectangle corners when it is created and contains $8n/r$ rectangle corners when it splits. Thus, from the time point that L is created to the time point that L splits, there are at least n/r inserted rectangles which have corners in L, which implies that L gets charged at least n/r times. Since there can be at most t inserted rectangles, the total number of charges is bounded by $2t$, which implies that the number of slab splitting is bounded by $2tr/n$. ◀

[1] There are different definitions of fat objects in different papers. One can easily show that our definition is equivalent to the others.

Clearly, using the information stored in the slab data structures, we can compute in $O(r)$ time a point p^* satisfying $\mathsf{dep}(p^*, \mathcal{Q}) = \mathsf{dep}(\mathcal{Q})$, as well as the value of $\mathsf{dep}(\mathcal{Q})$. More precisely, we just consider the points $p_1, \dots, p_r$ stored in the slab data structures (and their depths with respect to $\mathcal{Q}$) and define p^* as the one that maximizes its depth with respect to $\mathcal{Q}$. Therefore, the remaining task now is to design efficient slab data structures.

The slab data structures. Consider a slab L. We want to maintain a point $p \in L$ that maximizes $\mathsf{dep}(p, \mathcal{Q})$, as well as the value of $\mathsf{dep}(p, \mathcal{Q})$. Clearly, we can ignore the rectangles in $\mathcal{Q}$ that are disjoint from L. Thus, in the following we assume that every rectangle in $\mathcal{Q}$ intersects L. We classify the rectangles in $\mathcal{Q}$ into two types: *long* rectangles and *short* rectangles. A rectangle is *long* if its four corners are all outside L and is *short* if at least one corner is inside (the interior of) L. Let $\mathcal{Q}_{\mathsf{long}} \subseteq \mathcal{Q}$ and $\mathcal{Q}_{\mathsf{short}} \subseteq \mathcal{Q}$ be the sets of long and short rectangles in $\mathcal{Q}$, respectively.

To maintain the maximum depth of $\mathcal{Q}$ inside L, our main idea is to reduce the problem to maintaining the maximum depth of *weighted* intervals. Specifically, we shall construct a set $\mathcal{I}$ of weighted intervals from $\mathcal{Q}$ such that the maximum depth of $\mathcal{Q}$ inside L is equal to the maximum (weighted) depth of $\mathcal{I}$. The construction is done as follows. For each long rectangle $Q = [x^-, x^+] \times [y^-, y^+] \in \mathcal{Q}_{\mathsf{long}}$, we include in $\mathcal{I}$ the interval $[x^-, x^+]$ with weight 1. For the set $\mathcal{Q}_{\mathsf{short}}$ of short rectangles, we first define a *depth function* $f : \mathbb{R} \to \mathbb{N}$ as $f(r) = \max_{p \in \ell_r \cap L} \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}})$ where ℓ_r is the vertical line with equation $x = r$. One can easily observe that f is a piecewise-constant function with $O(|\mathcal{Q}_{\mathsf{short}}|)$ pieces, since the value of f only changes at the x-coordinates of the corners of the short rectangles. Therefore, we can partition the real line $\mathbb{R}$ into $s = O(|\mathcal{Q}_{\mathsf{short}}|)$ intervals $I_1, \dots, I_s$ such that f is a constant function when restricted to each I_i. Also, for each I_i, there is a y-coordinate $y_i \in \mathbb{R}$ such that $I_i \times \{y_i\} \subseteq L$ and $f(x) = \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}})$ for any $x \in I_i$ and $p \in I_i \times \{y_i\}$. Define a function $g : \mathbb{R} \to \mathbb{R}$ as $g(x) = y_i$ if $x \in I_i$. We include $I_1, \dots, I_s$ in $\mathcal{I}$ and set the weight of each I_i to be $f(x)$ for an arbitrary $x \in I_i$. (See Figure 1 for an illustration.)

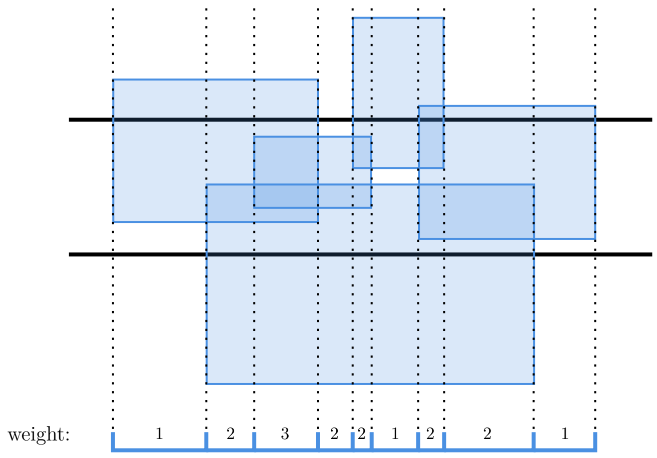

Figure 1 Construct of weighted intervals corresponding to $\mathcal{Q}_{\mathsf{short}}$.

We observe the following simple fact.

▶ **Fact 5.** *If $x^* \in \mathbb{R}$ satisfies that $\mathsf{dep}(x^*, \mathcal{I}) = \mathsf{dep}(\mathcal{I})$, the point $p^* = (x^*, g(x^*)) \in L$ satisfies $\mathsf{dep}(p^*, \mathcal{Q}) = \mathsf{dep}(x^*, \mathcal{I}) = \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$.*

Proof. We first prove that $\mathsf{dep}(p^*, \mathcal{Q}) = \mathsf{dep}(x^*, \mathcal{I})$. By definition, $\mathsf{dep}(p^*, \mathcal{Q}) = \mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{long}}) + \mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{short}})$. Note that a rectangle in $\mathcal{Q}_{\mathsf{long}}$ contains p^* iff its corresponding weight-1 interval in $\mathcal{I}$ contains x^*. So x^* is contained in exactly $\mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{long}})$ weight-1 intervals corresponding to the rectangles in $\mathcal{Q}_{\mathsf{long}}$. Meanwhile, the total weight of the intervals containing x^* which correspond to the rectangles in $\mathcal{Q}_{\mathsf{short}}$ is equal to $f(x^*)$. It follows that $\mathsf{dep}(x^*, \mathcal{I}) = \mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{long}}) + f(x^*)$. As $p^* = (x^*, g(x^*))$, we have $\mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{short}}) = f(x^*)$ by the definition of g. Thus, $\mathsf{dep}(x^*, \mathcal{I}) = \mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{long}}) + \mathsf{dep}(p^*, \mathcal{Q}_{\mathsf{short}}) = \mathsf{dep}(p^*, \mathcal{Q})$.

Since $\mathsf{dep}(p^*, \mathcal{Q}) \leq \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$, it suffices to show that $\mathsf{dep}(x^*, \mathcal{I}) \geq \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$. Consider a point $p \in L$. Again, a rectangle in $\mathcal{Q}_{\mathsf{long}}$ contains p iff its corresponding weight-1 interval in $\mathcal{I}$ contains $x(p)$. Also, $f(x(p)) \geq \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}})$ by the definition of f. Therefore,

$$\mathsf{dep}(x(p), \mathcal{I}) = \mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) + f(x(p)) \geq \mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) + \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}}) = \mathsf{dep}(p, \mathcal{Q}).$$

Since $\mathsf{dep}(x^*, \mathcal{I}) = \mathsf{dep}(\mathcal{I})$, we have $\mathsf{dep}(x^*, \mathcal{I}) \geq \mathsf{dep}(x(p), \mathcal{I}) \geq \mathsf{dep}(p, \mathcal{Q})$. ◀

For our slab data structure $\mathcal{D}(L)$, we can simply use the following result of Imai and Asano [14], which is a byproduct of their (static) algorithm for the maximum-depth problem for weighted rectangles.

▶ **Lemma 6** ([14]). *There exists a dynamic maximum-depth data structure $\mathcal{A}$ for weighted intervals with $O(\log n)$ update time and $O(n \log n)$ preprocessing time.*

Given $\mathcal{Q}$, we first compute the set $\mathcal{I}$ as stated above (as well as the functions f and g), and then build $\mathcal{A}(\mathcal{I})$, i.e., the interval data structure of the above lemma on $\mathcal{I}$). $\mathcal{A}(\mathcal{I})$ maintains $x^* \in \mathbb{R}$ satisfying $\mathsf{dep}(x^*, \mathcal{I}) = \mathsf{dep}(\mathcal{I})$, as well as the value of $\mathsf{dep}(x^*, \mathcal{I})$. The point we maintain is $p^* = (x^*, g(x^*))$, which satisfies $\mathsf{dep}(p^*, \mathcal{Q}) = \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$ by Fact 5. Also, the value of $\mathsf{dep}(p^*, \mathcal{Q})$ is equal to $\mathsf{dep}(x^*, \mathcal{I})$ by Fact 5, and we can maintain the latter from $\mathcal{A}(\mathcal{I})$.

Consider an insertion/deletion on $\mathcal{Q}$, and let Q be the inserted/deleted rectangle. If Q is a long rectangle, we just need to insert/delete its corresponding weight-1 interval on $\mathcal{I}$ and update $\mathcal{A}(\mathcal{I})$. If Q is a short rectangle, the situation is more complicated and what we do is the following. First, we delete from $\mathcal{I}$ all intervals corresponding to the rectangles in the old $\mathcal{Q}_{\mathsf{short}}$. Then we construct the intervals corresponding to the rectangles in the new $\mathcal{Q}_{\mathsf{short}}$ (we shall show how to do this efficiently) and insert them to $\mathcal{I}$. Also, we re-compute the functions f and g corresponding to the new $\mathcal{Q}_{\mathsf{short}}$. For both deletions and insertions on $\mathcal{I}$, we need to update $\mathcal{A}(\mathcal{I})$ accordingly.

Next, we analyze the preprocessing time and update time for our slab data structure. To this end, we first show that one can compute the weighted intervals in $\mathcal{I}$ corresponding to the short rectangles, as well as the functions f and g, in $O(|\mathcal{Q}_{\mathsf{short}}| \log |\mathcal{Q}_{\mathsf{short}}|)$ time. Indeed, we can directly apply the classical (static) rectangle maximum-depth algorithm [14] on $\mathcal{Q}_{\mathsf{short}}$. The algorithm sweeps a vertical line ℓ from left to right, and maintains the maximum depth of $\mathcal{Q}_{\mathsf{short}}$ on ℓ. Therefore, it exactly computes the function f and g, together with their piece intervals $I_1, \dots, I_s$. The running time of the algorithm is $O(|\mathcal{Q}_{\mathsf{short}}| \log |\mathcal{Q}_{\mathsf{short}}|)$. The intervals in $\mathcal{I}$ corresponding to the long rectangles can be constructed directly. Overall, the time for constructing $\mathcal{I}$ is $O(|\mathcal{Q}| \log |\mathcal{Q}|)$, i.e., $O(n \log n)$. By Lemma 6, $\mathcal{A}(\mathcal{I})$ can also be built in $O(n \log n)$ time as $|\mathcal{I}| = O(|\mathcal{Q}|)$. So the preprocessing time of our slab data structure is $O(n \log n)$. For the update time, we need to distinguish long rectangles and short rectangles. Inserting/deleting a long rectangle corresponds to an insertion/deletion on $\mathcal{A}(\mathcal{I})$ and thus can be done in $O(\log n)$ time. For the insertion/deletion of a short rectangle, we need to delete $O(|\mathcal{Q}_{\mathsf{short}}|)$ intervals from $\mathcal{I}$, re-compute the intervals corresponding to $\mathcal{Q}_{\mathsf{short}}$ and the functions f and g, and insert $O(|\mathcal{Q}_{\mathsf{short}}|)$ intervals to $\mathcal{I}$. The $O(|\mathcal{Q}_{\mathsf{short}}|)$ insertions/deletions on $\mathcal{A}(\mathcal{I})$ can be done in $O(|\mathcal{Q}_{\mathsf{short}}| \log n)$ time by Lemma 6, and the re-computation can be done in $O(|\mathcal{Q}_{\mathsf{short}}| \log |\mathcal{Q}_{\mathsf{short}}|)$ time as discussed above. As the number of rectangle corners in a slab is always $O(n/r)$, we have $|\mathcal{Q}_{\mathsf{short}}| = O(n/r)$. Thus, the update time for a short rectangle is $O((n/r) \log n)$.

▶ **Lemma 7.** *One can construct each slab data structure in $O(n \log n)$ time such that for insertions/deletions of long (resp., short) rectangles, the data structure can be updated in $O(\log n)$ time (resp., $O((n/r) \log n)$ time).*

Overall time complexity. Using Lemma 7, we can now analyze the overall preprocessing time and update time of our data structure. Partitioning the plane into slabs can be done in $O(n \log n)$ time by sorting the corners of the rectangles in $\mathcal{Q}$. Since there are r slabs and each slab data structure can be constructed in $O(n \log n)$ time, the preprocessing time is $O(rn \log n)$. To analyze the update time, let Q be the inserted/deleted rectangle. Note that there are at most two slabs in which Q is a short rectangle. So among the updates of the slab data structures, there can be at most two updates for short rectangles and $O(r)$ updates for long rectangles. By Lemma 7, these updates take $O((n/r) \log n + r \log n)$ time. Also, rectangle insertions might cause slab splittings and for each slab splitting we need $O(n \log n)$ time to build the new slab data structures. By Fact 4, the time cost for slab splitting during the first t operations is $O(tr \log n)$, and the amortized time cost is then $O(r \log n)$. Therefore, the overall (amortized) update time of our data structure is $O((n/r) \log n + r \log n)$. To balance the two terms, we set $r = \sqrt{n}$, yielding $O(\sqrt{n} \log n)$ update time. The preprocessing time is then $O(n^{1.5} \log n)$. Based on the above discussion, we have the following conclusion, which is the base case $k = 1$ of Theorem 1.

▶ **Lemma 8.** *There exists a dynamic maximum-depth data structure for axis-parallel rectangles with $O(\sqrt{n} \log n)$ amortized update time, which can be built in $O(n^{1.5} \log n)$ time.*

Approximate data structures. Now we are able to prove Theorem 1 for a general $k \in \mathbb{N}$. As aforementioned, we apply induction on k. Lemma 8 gives us the base case $k = 1$. Suppose there exists a $\frac{1}{k}$-approximate dynamic maximum-depth data structure for rectangles with $O(n^{1/(k+1)} \log n)$ (amortized) update time and $O(n^{1+1/(k+1)} \log n)$ preprocessing time. We want to show the existence of a $\frac{1}{k+1}$-approximate data structure with $O(n^{1/(k+2)} \log n)$ update time and $O(n^{1+1/(k+2)} \log n)$ preprocessing time.

As before, our data structure partitions the plane into r slabs. The only difference occurs in the design of the slab data structures. Consider a slab L. We want our slab data structure for L to maintain a point $p^* \in L$ satisfying that $\mathsf{dep}(p^*, \mathcal{Q}) \geq \frac{1}{k+1} \cdot \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$; this guarantees that our overall data structure achieves the approximation ratio $\frac{1}{k+1}$. Again, let $\mathcal{Q}_{\mathsf{long}}$ and $\mathcal{Q}_{\mathsf{short}}$ denote the set of long and short rectangles in $\mathcal{Q}$ with respect to L, respectively. Unlike in our exact data structure, we now handle $\mathcal{Q}_{\mathsf{long}}$ and $\mathcal{Q}_{\mathsf{short}}$ separately. For long rectangles, we maintain a point $p_1^* \in L$ that maximizes $\mathsf{dep}(p_1^*, \mathcal{Q}_{\mathsf{long}})$. This can be done using exactly the same idea as before: map each long rectangle $Q = [x^-, x^+] \times [y^-, y^+] \in \mathcal{Q}_{\mathsf{long}}$ to the interval $[x^-, x^+]$, and build $\mathcal{A}$ on these intervals. For short rectangles, we maintain a point $p_2^* \in L$ satisfying the condition that $\mathsf{dep}(p_2^*, \mathcal{Q}_{\mathsf{short}}) \geq \frac{1}{k} \cdot \max_{p \in L} \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}})$. To this end, we build a $\frac{1}{k}$-approximate dynamic maximum-depth data structure on $\mathcal{Q}_{\mathsf{short}}$, which can give us the desired point p_2^*. By our induction hypothesis, there exists such a data structure with $O(n^{1/(k+1)} \log n)$ update time and $O(n^{1+1/(k+1)} \log n)$ preprocessing time. Define $p^* = p_1^*$ if $\mathsf{dep}(p_1^*, \mathcal{Q}_{\mathsf{long}}) \geq \mathsf{dep}(p_2^*, \mathcal{Q}_{\mathsf{short}})$ and $p^* = p_2^*$ if $\mathsf{dep}(p_1^*, \mathcal{Q}_{\mathsf{long}}) < \mathsf{dep}(p_2^*, \mathcal{Q}_{\mathsf{short}})$. We observe the following.

▶ **Lemma 9.** $\mathsf{dep}(p^*, \mathcal{Q}) \geq \frac{1}{k+1} \cdot \max_{p \in L} \mathsf{dep}(p, \mathcal{Q})$.

Proof. Let $p \in L$ be a point that maximizes $\mathsf{dep}(p, \mathcal{Q})$. By definition, we have $\mathsf{dep}(p, \mathcal{Q}) = \mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) + \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}})$. We consider two cases: $\mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) \geq \frac{1}{k+1}\mathsf{dep}(p, \mathcal{Q})$ and $\mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) < \frac{1}{k+1}\mathsf{dep}(p, \mathcal{Q})$. If $\mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) \geq \frac{1}{k+1}\mathsf{dep}(p, \mathcal{Q})$, we have

$$\mathsf{dep}(p^*, \mathcal{Q}) \geq \mathsf{dep}(p_1^*, \mathcal{Q}) \geq \mathsf{dep}(p_1^*, \mathcal{Q}_{\mathsf{long}}) \geq \mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) \geq \frac{1}{k+1}\mathsf{dep}(p, \mathcal{Q}).$$

On the other hand, if $\mathsf{dep}(p, \mathcal{Q}_{\mathsf{long}}) < \frac{1}{k+1}\mathsf{dep}(p, \mathcal{Q})$, we must have $\mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}}) > \frac{k}{k+1}\mathsf{dep}(p, \mathcal{Q})$. In this case, we have

$$\mathsf{dep}(p^*, \mathcal{Q}) \geq \mathsf{dep}(p_2^*, \mathcal{Q}) \geq \mathsf{dep}(p_2^*, \mathcal{Q}_{\mathsf{short}}) \geq \frac{1}{k} \cdot \mathsf{dep}(p, \mathcal{Q}_{\mathsf{short}}) > \frac{1}{k+1} \cdot \mathsf{dep}(p, \mathcal{Q}).$$

Therefore, we always have $\mathsf{dep}(p^*, \mathcal{Q}) \geq \frac{1}{k+1} \cdot \mathsf{dep}(p, \mathcal{Q})$. ◀

The data structure for maintaining p_1^* can be constructed in $O(|\mathcal{Q}_{\mathsf{long}}| \log |\mathcal{Q}_{\mathsf{long}}|)$ time and updated in $O(\log |\mathcal{Q}_{\mathsf{long}}|)$ time, by Lemma 6. Furthermore, the data structure for maintaining p_2^* can be constructed in $O(|\mathcal{Q}_{\mathsf{short}}|^{1+1/(k+1)} \log |\mathcal{Q}_{\mathsf{short}}|)$ time and updated in $O(|\mathcal{Q}_{\mathsf{short}}|^{1/(k+1)} \log |\mathcal{Q}_{\mathsf{short}}|)$ time. Since $|\mathcal{Q}_{\mathsf{long}}| = O(n)$ and $|\mathcal{Q}_{\mathsf{short}}| = O(n/r)$, for each slab data structure, the preprocessing time is $O(n \log n + (n/r)^{1+1/(k+1)} \log(n/r))$ and the update time is $O(\log n)$ for long rectangles and $O((n/r)^{1/(k+1)} \log(n/r))$ for short rectangles. This implies that our overall data structure has preprocessing time $O(rn \log n + r(n/r)^{1+1/(k+1)} \log(n/r))$ and update time $O(r \log n + (n/r)^{1/(k+1)} \log(n/r))$. By setting $r = n^{1/(k+2)}$, we have the desired $\frac{1}{k+1}$-approximation data structure with $O(n^{1/(k+2)} \log n)$ amortized update time and $O(n^{1+1/(k+2)} \log n)$ preprocessing time.

▶ **Theorem 1.** *For every $k \in \mathbb{Z}^+$, there exists a $\frac{1}{k}$-approximate dynamic maximum-depth data structure for axis-parallel rectangles with $O(n^{1/(k+1)} \log n)$ amortized update time.*

4 Dynamic maximum-depth for disks

We now describe our dynamic maximum-depth data structure for disks. The main idea underlying our result is to reduce the maximum-depth problem to *discrete* maximum depth.

Suppose we want to design a $(\frac{1}{2} - \varepsilon)$-approximate maximum-depth data structure for disks, where $\varepsilon > 0$ is a constant. Without loss of generality, we assume that ε is sufficiently small, say, $\varepsilon < 0.1$. Let δ be a sufficiently large integer such that $\frac{1}{\delta} \leq 1 - \cos \varepsilon\pi$. For a disk D in $\mathbb{R}^2$ with center (x, y) and radius r, we define

$$\varGamma_\delta(D) = \{(x - 3r + ir/\delta, y - 3r + jr/\delta) \in \mathbb{R}^2 : i, j \in [6\delta]\},$$

and call the points in $\varGamma_\delta(D)$ the *companion points* of D. For a set $\mathcal{D}$ of disks, write $\varGamma_\delta(\mathcal{D}) = \bigcup_{D \in \mathcal{D}} \varGamma_\delta(D)$. The set of companion points is the discrete point set for which our data structure dynamically maintains depth. The following fact is straightforward.

▶ **Fact 10.** *Let D be a disk and S be a set of points in D. There exists a circular sector X of D with angle $(1 - 2\varepsilon)\pi$ such that $|S \cap X| \geq (\frac{1}{2} - \varepsilon) \cdot |S|$.*

Proof. Suppose we randomly sample a circular sector X of D with angle $(1 - 2\varepsilon)\pi$, from the uniform distribution over the space of all such sectors. The probability that a specific point $p \in S$ is contained in X is 1 if p is the center of D and is $\frac{1}{2} - \varepsilon$ otherwise. Therefore, $\mathbb{E}[|S \cap X|] \geq (\frac{1}{2} - \varepsilon) \cdot |S|$, which in turn implies that there exists a sector X of D with angle $(1 - 2\varepsilon)\pi$ for which $|S \cap X| \geq (\frac{1}{2} - \varepsilon) \cdot |S|$. ◀

▶ **Lemma 11.** *Let $\mathcal{D}$ be a set of congruent disks such that $\bigcap_{D\in\mathcal{D}} D \neq \emptyset$. Then for every $D \in \mathcal{D}$, there exists $p \in \Gamma_\delta(D)$ such that* $\mathsf{dep}(p,\mathcal{D}) \geq \left(\frac{1}{2}-\varepsilon\right)\cdot|\mathcal{D}|$.

Proof. Without loss of generality, we assume that the disks in $\mathcal{D}$ are unit disks. Let $p^* \in \bigcap_{D\in\mathcal{D}} D$ and P^* be the unit disk centered at p^*. Define $\mathsf{ctr}(\mathcal{D})$ as the set of the centers of the disks in $\mathcal{D}$. Since $p^* \in \bigcap_{D\in\mathcal{D}} D$, we have $\mathsf{ctr}(\mathcal{D}) \subseteq P^*$. By Fact 10, there is a circular sector X of P^* with angle $(1-2\varepsilon)\pi$ such that $|\mathsf{ctr}(\mathcal{D}) \cap X| \geq (\frac{1}{2}-\varepsilon)\cdot|\mathsf{ctr}(\mathcal{D})| = (\frac{1}{2}-\varepsilon)\cdot|\mathcal{D}|$. Suppose u and v are the two vertices of X on the boundary of P^*. Define o as the middle point of the segment connecting u and v, and D_o as the disk centered at o with radius $1-\cos\varepsilon\pi$. We claim that $\mathsf{dep}(p,\mathcal{D}) \geq \left(\frac{1}{2}-\varepsilon\right)\cdot|\mathcal{D}|$ for any point $p \in D_o$. Consider an arbitrary point $p \in D_o$. Since the angle of X is $(1-2\varepsilon)\pi$, the distance between o and p^* is $\sin\varepsilon\pi$. Also, the distance between o and u (resp., v) is $\cos\varepsilon\pi$. Recall that $\varepsilon < 0.1$ by our assumption, which implies that $\sin\varepsilon\pi < \cos\varepsilon\pi$. Therefore, the distance between p and p^* is at most $\sin\varepsilon\pi + (1-\cos\varepsilon\pi) < 1$, and the distance between p and u (resp., v) is at most $\cos\varepsilon\pi + (1-\cos\varepsilon\pi) = 1$. Therefore, the unit disk centered at p contains points p^*, u, v, which further implies that it contains X. Consequently, the unit disks in $\mathcal{D}$ whose centers lie in X contains p. We have $|\mathsf{ctr}(\mathcal{D}) \cap X| \geq (\frac{1}{2}-\varepsilon)\cdot|\mathcal{D}|$, which then implies that $\mathsf{dep}(p,\mathcal{D}) \geq \left(\frac{1}{2}-\varepsilon\right)\cdot|\mathcal{D}|$. See Figure 2 for an illustration.

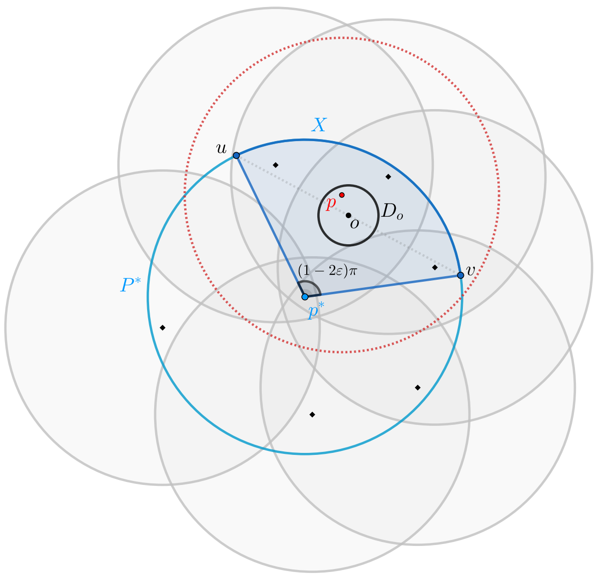

Figure 2 Illustrating the proof of Lemma 11.

Now it suffices to show $\Gamma_\delta(D) \cap D_o \neq \emptyset$ for every $D \in \mathcal{D}$. Note that the distance between o and the center of D is at most 2, since $p^* \in D$ and $o \in P^*$. Furthermore, D_o contains an axis-parallel square centered at o with side-length $1-\cos\varepsilon\pi$, as its radius is $1-\cos\varepsilon\pi$. Since $1-\cos\varepsilon\pi \geq \frac{1}{\delta}$, this square then contains a point in $\Gamma_\delta(D)$, by our construction of $\Gamma_\delta(D)$. Therefore, $\Gamma_\delta(D) \cap D_o \neq \emptyset$. ◀

▶ **Corollary 12.** *For any set $\mathcal{D}$ of disks in $\mathbb{R}^2$, the following holds:*

$$\max_{p\in\Gamma_\delta(\mathcal{D})} \mathsf{dep}(p,\mathcal{D}) \geq \left(\frac{1}{2}-\varepsilon\right)\cdot \mathsf{dep}(\mathcal{D}).$$

Proof. Let $p^* \in \mathbb{R}^2$ such that $\mathsf{dep}(p^*,\mathcal{D}) = \mathsf{dep}(\mathcal{D})$, and $\mathcal{D}^* = \{D \in \mathcal{D} : p^* \in D\}$. Suppose $D_0 \in \mathcal{D}^*$ is the smallest disk in $\mathcal{D}^*$. For each disk $D \in \mathcal{D}^*$, we can find a smaller disk $D' \subseteq D$ with the same radius as D_0 such that $p^* \in D'$. Let $\mathcal{D}'$ be the set of all these disks. Then $\mathcal{D}'$

is a set of congruent disks containing p^*, and $\mathsf{dep}(p, \mathcal{D}') \leq \mathsf{dep}(p, \mathcal{D}^*)$ for any $p \in \mathbb{R}^2$. The former implies that $\bigcap_{D \in \mathcal{D}'} D \neq \emptyset$. Furthermore, we have $D_0 \in \mathcal{D}'$. By Lemma 11, there exists $p \in \Gamma_\delta(D_0)$ such that $\mathsf{dep}(p, \mathcal{D}') \geq (\frac{1}{2} - \varepsilon) \cdot |\mathcal{D}'| = (\frac{1}{2} - \varepsilon) \cdot |\mathcal{D}^*| = (\frac{1}{2} - \varepsilon) \cdot \mathsf{dep}(\mathcal{D})$. This proves the corollary. ◀

Using this corollary, we can reduce the task of maintaining a $(\frac{1}{2} - \varepsilon)$-approximation of the maximum depth of disks to the dynamic *discrete* maximum-depth problem for disks. Let $\mathcal{D}$ be a dynamic set of disks. Corollary 12 implies that $\max_{p \in S} \mathsf{dep}(p, \mathcal{D}) \geq (\frac{1}{2} - \varepsilon) \cdot \mathsf{dep}(\mathcal{D})$, where $S = \Gamma_\delta(\mathcal{D})$. Therefore, it suffices to consider the dynamic discrete maximum-depth instance with point set S and disk set $\mathcal{D}$.

4.1 Dynamic discrete maximum depth for disks

In this section, we design a dynamic discrete maximum-depth data structure for disks with $n^{2/3} \log^{O(1)} n$ amortized update time. Via the standard lifting argument, the dynamic discrete maximum-depth problem for disks can be reduced to the same problem for 3D halfspaces. To construct the discrete maximum-depth data structure, we first need a dynamic *depth-query* data structure for halfspaces. Specifically, such a data structure maintains a dynamic set $\mathcal{H}$ of hyperplanes and can return, for a query point $q \in \mathbb{R}^3$, the value of $\mathsf{dep}(q, \mathcal{H})$.

▶ **Lemma 13.** *There exists an dynamic depth-query data structure for 3D halfspaces with $n^{2/3} \log^{O(1)} n$ query time and $\log^{O(1)} n$ amortized update time. The data structure can be built in $n \log^{O(1)} n$ time.*

Proof. By duality between points and halfspaces, the depth-query problem is equivalent to the range-counting problem: store a dynamic set S of points in $\mathbb{R}^3$ such that for a query halfspace H, one can return the value $|S \cap H|$. A dynamic 3D halfspace range-counting data structure with $n^{2/3} \log^{O(1)} n$ query time and $\log^{O(1)} n$ amortized update time can be directly obtained using the dynamic partition tree given by Matoušek [15]. ◀

Let S be the dynamic set of points and $\mathcal{H}$ be the dynamic set of halfspaces, and we want to maintain $\max_{p \in S} \mathsf{dep}(p, \mathcal{H})$. We build the dynamic depth-query data structure in Lemma 13 for $\mathcal{H}$. We store the points in S in a standard partition tree $\mathcal{T}$ [15]. The tree $\mathcal{T}$ has $|S| \log^{O(1)} |S|$ nodes and can be built in $|S| \log^{O(1)} |S|$ time. The depth of $\mathcal{T}$ is $\log^{O(1)} |S|$ and the leaves of $\mathcal{T}$ one-to-one correspond to the points in S. Each node v of $\mathcal{T}$ corresponds to a subset $S(v) \subseteq S$, which consists of the points on the leaves of the subtree rooted at v. Furthermore, given any halfspace H in $\mathbb{R}^3$, one can find $k = |S|^{2/3} \log^{O(1)} |S|$ nodes $v_1, \dots, v_k$ of $\mathcal{T}$ in $|S|^{2/3} \log^{O(1)} |S|$ time such that $S \cap H = \bigcup_{i=1}^k S(v_i)$ and the sets $S(v_1), \dots, S(v_k)$ are pairwise disjoint; these nodes are called the *canonical nodes* for H. For each node v, we maintain two fields, $\sigma(v)$ and $\mu(v)$. The fields $\sigma(v)$ satisfies the following invariant: for any point $p \in S$, the sum of $\sigma(v)$ over all nodes v satisfying $p \in S(v)$ is equal to $\mathsf{dep}(p, \mathcal{H})$. The field $\mu(v)$ is defined as

$$\mu(v) = \begin{cases} \sigma(v) & \text{if } v \text{ is a leaf,} \\ \sigma(v) + \max_{u \in \mathsf{ch}(v)} \mu(u) & \text{otherwise,} \end{cases}$$

where $\mathsf{ch}(v)$ is the set of children of v. By the invariant for the σ-fields, we see that the μ-field of the root of $\mathcal{T}$ is just equal to $\max_{p \in S} \mathsf{dep}(p, \mathcal{H})$.

Halfspace insertion/deletion. For a halfspace insertion/deletion, we do not need to change the partition tree $\mathcal{T}$ itself; instead, we need to update the σ-fields and μ-fields. Let H be a halfspace to be inserted. We find the canonical nodes $v_1, \dots, v_k$ for H. Then we increase the fields $\sigma(v_1), \dots, \sigma(v_k)$ by 1. We also update the μ-fields of all ancestors of $v_1, \dots, v_k$. Note that if a node is not an ancestor of any of $v_1, \dots, v_k$, then its μ-field is not changed. The deletion of a halfspace can be handled in the same way, with the only difference that we decrease the fields $\sigma(v_1), \dots, \sigma(v_k)$ by 1. The update time is $|S|^{2/3} \log^{O(1)} |S| = n^{2/3} \log^{O(1)} n$. Besides, we also need to update the depth-query data structure for $\mathcal{H}$, which takes $|\mathcal{H}|^{2/3} \log^{O(1)} |\mathcal{H}| = n^{2/3} \log^{O(1)} n$ time as well.

Point insertion/deletion. To delete a point p from S, we can directly remove the leaf v of $\mathcal{T}$ corresponding to p, and update the μ-fields of all ancestors of v. This takes $\log^{O(1)} n$ time. To handle point insertions, we apply the classical logarithmic method [4]. This method, instead of using one partition tree, stores S in $O(\log |S|)$ partition trees each of which corresponds to a subset of S whose size is a power of 2. The only problem we need to solve here is how to efficiently merge two partition trees storing subsets of S with size 2^i into a larger partition tree $\mathcal{T}$, with the σ-fields and μ-fields. The tree $\mathcal{T}$ itself can be built in $2^i \cdot i^{O(1)}$ time. To compute the σ-fields and μ-fields for the nodes of $\mathcal{T}$, what we do is the following. Let $S' \subseteq S$ be the subset $\mathcal{T}$ stores. We use the depth-query data structure to obtain $\mathsf{dep}(p, \mathcal{H})$ for all $p \in S'$. For each leaf v of $\mathcal{T}$, we set $\sigma(v) = \mathsf{dep}(p, \mathcal{H})$, where $p \in S'$ is the point corresponding to v. We set $\sigma(v) = 0$ for all internal nodes v of $\mathcal{T}$. Clearly, the invariant for the σ-fields holds. Using the σ-fields, we then compute the μ-field for every node of $\mathcal{T}$ in a bottom-up order. The most time-consuming part in this procedure is the depth queries, which takes $|S'| \cdot |\mathcal{H}|^{2/3} \log^{O(1)} |\mathcal{H}| = 2^i n^{2/3} \log^{O(1)} n$ time. As such, the amortized update time for a point insertion is $n^{2/3} \log^{O(1)} n$.

▶ **Lemma 14.** *There exists a dynamic discrete maximum-depth data structure for disks with $n^{2/3} \log^{O(1)} n$ amortized update time for insertions/deletions of points and disks; here n is the total number of points and disks in the instance.*

▶ **Theorem 2.** *For any constant $\varepsilon > 0$, there exists a $(\frac{1}{2} - \varepsilon)$-approximate dynamic maximum-depth data structure for disks with $n^{2/3} \log^{O(1)} n$ amortized update time.*

Proof. We just build the data structure of Lemma 14 on the point set $S = \Gamma_\delta(\mathcal{D})$ and the disk set $\mathcal{D}$. The data structure maintains a point $p^* \in S$ such that $\mathsf{dep}(p^*, \mathcal{D}) = \max_{p \in S} \mathsf{dep}(p, \mathcal{D})$ and thus $\mathsf{dep}(p^*, \mathcal{D}) \geq (\frac{1}{2} - \varepsilon) \cdot \mathsf{dep}(\mathcal{D})$ by Corollary 12. Each insertion (resp., deletion) on $\mathcal{D}$ corresponds to $O(1)$ insertions (resp., deletions) on S and one insertion (resp., deletion) on $\mathcal{D}$. Therefore, the update time is $O(n^{2/3} \log^{O(1)} n)$ in total. ◀

We remark that our data structure for disks can be generalized to balls in higher dimensions. Indeed, the proof of Fact 10 holds for balls in any fixed dimension. To solve dynamic discrete maximum depth for balls in $\mathbb{R}^d$, one can reduce to the problem for halfspaces in $\mathbb{R}^{d+1}$ and use partition trees in $\mathbb{R}^{d+1}$. The update time becomes $n^{d/(d+1)} \log^{O(1)} n$.

4.2 Generalization to fat objects

In this section, we discuss how our reduction from the dynamic maximum-depth problem to its discrete counterpart works for general fat objects (where the approximation ratio depends on the fatness of the objects).

For a α-fat object O in $\mathbb{R}^d$ with whose center $(x_1 \cdots, x_d)$ and in-radius r, we define its *companion points* as

$$\Gamma_\alpha(O) = \left\{ (x_1 \pm \sqrt{2} i_1 r, \cdots, x_d \pm \sqrt{2} i_d r) : i_1, \cdots, i_d \in \left[\lceil \sqrt{2}\alpha \rceil \right] \right\}.$$

The key observation of the reduction for general fat objects is the following fact.

▶ **Fact 15.** *A d-dimensional ball with center $(x_1 \cdots, x_d)$ and radius $2\alpha r$ can be covered by balls with centers in $\Gamma_\alpha(O)$ and radii r.*

▶ **Corollary 16.** *For any set $\mathcal{O}$ of α-fat objects in $\mathbb{R}^d$, the following holds:*

$$\max_{p \in \Gamma_\alpha(\mathcal{O})} \mathsf{dep}(p, \mathcal{O}) \geq \frac{1}{\left(2 \lceil \sqrt{2}\alpha \rceil\right)^d} \cdot \mathsf{dep}(\mathcal{O}).$$

Proof. Let $p^* \in \mathbb{R}^d$ such that $\mathsf{dep}(p^*, \mathcal{O}) = \mathsf{dep}(\mathcal{O})$, and $\mathcal{O}^* = \{O \in \mathcal{O} : p^* \in O\}$. Suppose $O_0 \in \mathcal{O}^*$ is the smallest object in $\mathcal{O}^*$, i.e., O_0 has the smallest in-radius, which is assumed to be 1 without loss of generality. For each object $O \in \mathcal{O}^*$, we can find a unit ball $O' \subseteq O$ as following: the unit ball centered on the line segment between p^* and $\mathsf{ctr}(O)$, which is tangent to all tangent hyperplanes of O at p^*. (See Figure 3 for an illustration.)

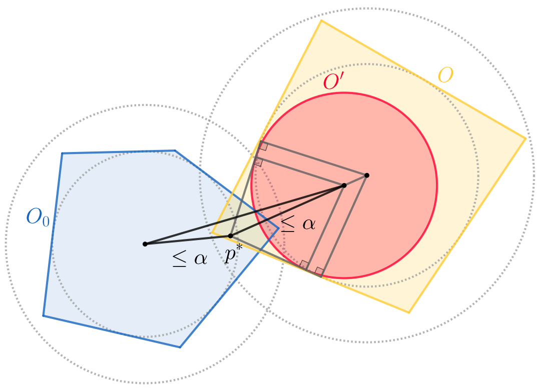

Figure 3 Illustrating the proof of Corollary 16 in $\mathbb{R}^2$.

Since O_0 contains p^*, we have the distance between p^* and $\mathsf{ctr}(O_0)$ is at most α. Moreover, it is clear that the distance between p^* and $\mathsf{ctr}(O')$ is at most α. Therefore, the distance between $\mathsf{ctr}(O')$ and $\mathsf{ctr}(O_0)$ is at most 2α, which means that the centers of these unit balls lie within a ball with center $\mathsf{ctr}(O_0)$ and radius 2α. By Fact 15, we can cover this ball by unit balls with centers in $\Gamma_\alpha(O_0)$. Therefore, by the pigeonhole principle, we have

$$\max_{p \in \Gamma_\alpha(\mathcal{O})} \mathsf{dep}(p, \mathcal{O}) \geq \max_{p \in \Gamma_\alpha(\mathcal{O}_0)} \mathsf{dep}(p, \mathcal{O}) \geq \frac{1}{|\Gamma_\alpha(\mathcal{O}_0)|} \cdot \mathsf{dep}(\mathcal{O}) = \frac{1}{\left(2 \lceil \sqrt{2}\alpha \rceil\right)^d} \cdot \mathsf{dep}(\mathcal{O}),$$

which completes the proof. ◀

As a concluding remark, we note that our reduction for general fat objects relies on the following question: Given a ball of radius 2α in $\mathbb{R}^d$, how many unit balls are needed to cover it? We employ a rather general method in the construction of $\Gamma_\alpha(O)$, to illustrate the main idea, and the constant $\left(2 \lceil \sqrt{2}\alpha \rceil\right)^d$ in Corollary 16 can be improved for specific values of α and d. For example, in the case of disk ($d = 2$ and $\alpha = 1$), as proved in [5], only 7 unit disks are needed, instead of 16 in Corollary 16. This optimization problem for particular values of α and d may have independent interest.

References

1 Pankaj K. Agarwal, Hsien-Chih Chang, Subhash Suri, Allen Xiao, and Jie Xue. Dynamic geometric set cover and hitting set. *ACM Trans. Algorithms*, 18(4):40:1–40:37, 2022. `doi:10.1145/3551639`.

2 Helmut Alt and Ludmila Scharf. Computing the depth of an arrangement of axis-aligned rectangles in parallel. In *26th European Workshop on Computational Geometry*, pages 33–36, 2010.

3 Boris Aronov and Sariel Har-Peled. On approximating the depth and related problems. *SIAM J. Comput.*, 38(3):899–921, 2008. `doi:10.1137/060669474`.

4 Jon Louis Bentley and James B. Saxe. Decomposable searching problems I: static-to-dynamic transformation. *J. Algorithms*, 1(4):301–358, 1980. `doi:10.1016/0196-6774(80)90015-2`.

5 Károly Bezdek. *Körök optimális fedései (Optimal Covering of Circles)*. PhD thesis, Eötvös Lorand University, 1979.

6 Sujoy Bhore and Timothy M. Chan. Fast static and dynamic approximation algorithms for geometric optimization problems: Piercing, independent set, vertex cover, and matching. *CoRR*, abs/2407.20659, 2024. `doi:10.48550/arXiv.2407.20659`.

7 Sujoy Bhore, Guangping Li, and Martin Nöllenburg. An algorithmic study of fully dynamic independent sets for map labeling. *ACM J. Exp. Algorithmics*, 27:1.8:1–1.8:36, 2022. `doi:10.1145/3514240`.

8 Paz Carmi, Matthew J. Katz, and Pat Morin. Stabbing pairwise intersecting disks by four points. *Discret. Comput. Geom.*, 70(4):1751–1784, 2023. `doi:10.1007/S00454-023-00567-0`.

9 Timothy M. Chan. A (slightly) faster algorithm for klee's measure problem. *Comput. Geom.*, 43(3):243–250, 2010. `doi:10.1016/J.COMGEO.2009.01.007`.

10 Timothy M. Chan and Qizheng He. More dynamic data structures for geometric set cover with sublinear update time. *J. Comput. Geom.*, 13(2):90–114, 2021. `doi:10.20382/JOCG.V13I2A6`.

11 Timothy M. Chan, Qizheng He, Subhash Suri, and Jie Xue. Dynamic geometric set cover, revisited. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022, Virtual Conference / Alexandria, VA, USA, January 9 - 12, 2022*, pages 3496–3528. SIAM, 2022. `doi:10.1137/1.9781611977073.139`.

12 Bernard Marie Chazelle and D. T. Lee. On a circle placement problem. *Computing*, 36(1-2):1–16, 1986. `doi:10.1007/BF02238188`.

13 Monika Henzinger, Stefan Neumann, and Andreas Wiese. Dynamic approximate maximum independent set of intervals, hypercubes and hyperrectangles. In Sergio Cabello and Danny Z. Chen, editors, *36th International Symposium on Computational Geometry, SoCG 2020, June 23-26, 2020, Zürich, Switzerland*, volume 164 of *LIPIcs*, pages 51:1–51:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. `doi:10.4230/LIPICS.SOCG.2020.51`.

14 Hiroshi Imai and Takao Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *Journal of Algorithms*, 4(4):310–323, December 1983. `doi:10.1016/0196-6774(83)90012-3`.

15 Jirí Matoušek. Efficient partition trees. *Discret. Comput. Geom.*, 8:315–334, 1992. `doi:10.1007/BF02293051`.

16 Hakan Yildiz, John Hershberger, and Subhash Suri. A discrete and dynamic version of klee's measure problem. In *Proceedings of the 23rd Annual Canadian Conference on Computational Geometry, Toronto, Ontario, Canada, August 10-12, 2011*, 2011. URL: `http://www.cccg.ca/proceedings/2011/papers/paper28.pdf`.