Sampling Hyperplanes and Revealing Disks*

Haim Kaplan[1], Alexander Kauer[†2], Wolfgang Mulzer[‡2], and Liam Roditty[3]

1 **School of Computer Science, Tel Aviv University**
haimk@tau.ac.il
2 **Institut für Informatik, Freie Universität Berlin, Berlin, Germany**
[akauer, mulzer]@inf.fu-berlin.de
3 **Department of Computer Science, Bar Ilan University**
liamr@macs.biu.ac.il

Abstract

We present a data structure which allows detecting when disks of a set B are no longer intersected by disks of set of disk A when deleting its disks. Preprocessing A, B and deleting n disks of A with detecting all newly revealed disks of B requires $\mathcal{O}(|B|\log^4|A| + |A|\log^5(|A|)\lambda_6(\log|A|) + n\log^9(|A|)\lambda_6(\log|A|))$ expected time, where $\lambda_6(\cdot)$ is the Davenport-Schinzel bound of order 6.

To construct this data structure, we extend known dynamic lower envelope data structures for hyperplanes and bivariate functions of constant description complexity with a linear size lower envelope in $\mathbb{R}^3$, such that they allow sampling of a random element not above a given point in $\mathcal{O}(\log^3 n)$ expected time.

1 From Graph Connectivity to Disk Sampling

Graph connectivity plays a fundamental role in algorithms and data structures. The dynamic variant where edges can be inserted or deleted is reasonably well understood [6–8,11,14], with data structures supporting updates and queries determining the connectivity of two vertices in polylogarithmic time. Updating vertices seems significantly harder, as a single update can have a large impact on the overall structure. Chan et al. [4, Theorem 1] presented a data structure allowing vertex updates in $\widetilde{\mathcal{O}}(m^{2/3})$ amortized time and queries in $\widetilde{\mathcal{O}}(m^{1/3})$ time, where m is the number of possible edges of the graph that need to be known in advance.

The vertices of geometric intersection graphs correspond to geometric objects and its edges to intersections. As they restrict possible graphs, faster solutions may be within reach. However, work is required to find edges affected by an update. Chan et al. [4, Theorem 5] gave a general method for various objects with sub-linear update and query times.

In this work we present a data structure, which allows detecting when a disk is no longer intersected by any disk of a set after deletions. We call such a disk *revealed*. This data structure can be used as a component in maintaining connectivity information in deletion-only disk intersection graphs [9].

To construct it, we first describe data structures for randomly sampling a hyperplane or a continuous function of constant description complexity not above a given point in $\mathbb{R}^3$, which might be of interest of its own.

* Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF).
† Supported in part by grant 1367/2016 from the German-Israeli Science Foundation (GIF) and by the German Research Foundation within the collaborative DACH project *Arrangements and Drawings* as DFG Project MU 3501/3-1.
‡ Supported in part by ERC StG 757609.

37th European Workshop on Computational Geometry, St. Petersburg, Russia, April 7–9, 2021.
This is an extended abstract of a presentation given at EuroCG'21. It has been made public for the benefit of the community and should be considered a preprint rather than a formally reviewed paper. Thus, this work is expected to appear eventually in more final form at a conference with formal proceedings and/or in a journal.

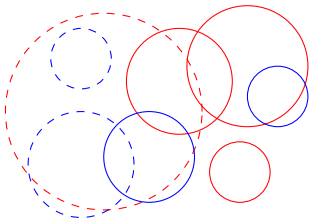

Figure 1 When removing a red disk, we want to obtain all blue disks intersecting this red disk but no other red disk. After removing the dashed red disk, the dashed blue disks need to be obtained.

2 Preliminaries

Let A, B be two sets of disks in $\mathbb{R}^3$, such that for each disk $b \in B$ exists at least one disk $a \in A$ it intersects. We now want to remove a disk $a \in A$ and obtain all disks $b \in B$ which no longer have any disk from A they intersect. We call such disks *revealed*. See Figure 1.

A data structure for detecting such revealed disks is constructed over the course of Sections 4 and 5, which concludes in Theorem 5.3. Its central idea is representing intersections sparsely via assigning each $b \in B$ repeatedly to one random $a \in A$ it intersects until there is none left and b is revealed.

Obtaining such a random disk of a given semi-dynamic set which intersects a query disk is the main problem. We will build upon the dynamic lower envelope data structures by Kaplan et al. Those maintain a dynamic set of hyperplanes [10, Section 7] or continuous bivariate functions of constant description complexity [10, Section 8] in $\mathbb{R}^3$ under insertions and deletions. Also, they support *vertical ray shooting queries* into their lower envelopes. The second variant is an extension of the first, and the first is a slight extension of a data structure by Chan [3]. We will briefly describe the first variant in Section 3.

The data structures use vertical k-shallow $(1/r)$-cuttings as their most integral part. Let $\mathcal{A}(H)$ be the arrangement of hyperplanes H in $\mathbb{R}^3$. The *k-level* L_k of $\mathcal{A}(H)$ is the closure of all points of $\bigcup H$ with k hyperplanes of H strictly below. Then, $L_{\leq k}$ is the union of the levels until L_k. A *vertical k-shallow $(1/r)$-cutting* is a set Λ of pairwise openly disjoint prisms, such that the union of Λ covers $L_{\leq k}$, the interior of each $\tau \in \Lambda$ is intersected by at most $|H|/r$ hyperplanes of H, and each prism is vertical (i.e. it consists of a triangle and all points below it). Some or all points of a prism's top may lie at infinity. The number of prisms is the *size* of the cutting. Using the algorithm by Chan and Tsakalidis a vertical $\Theta(|H|/r)$-shallow $1/r$-cutting of size $\mathcal{O}(r)$ can be created in $O(|H| \log r)$ time [5]. The *conflict list* $\mathrm{CL}(\tau)$ of a prism τ is the set of all hyperplanes crossing its interior.

This notion can also be extended to bivariate functions. Then the regions are vertical pseudo-prisms, where the top is limited by a pseudo-trapezoid part of a function [10, Section 3].

3 Vertical Ray Shooting Queries for Hyperplanes

First, we will construct a data structure for the simpler problem of sampling a random disk containing a given point from a dynamic set of disks. Using linearization [1, 15] we can transform this problem in $\mathbb{R}^2$ into the problem of sampling a hyperplane not above a given point in $\mathbb{R}^3$. This allows us to build upon the data structure for hyperplanes by Kaplan et al.

Their data structure consists of $\mathcal{O}(\log n)$ static substructures of exponentially decaying size, where n is the current number of hyperplanes. Substructures are periodically rebuilt similar to the Bentley-Saxe technique [2] and the whole data structure after $\Theta(n)$ updates. Each

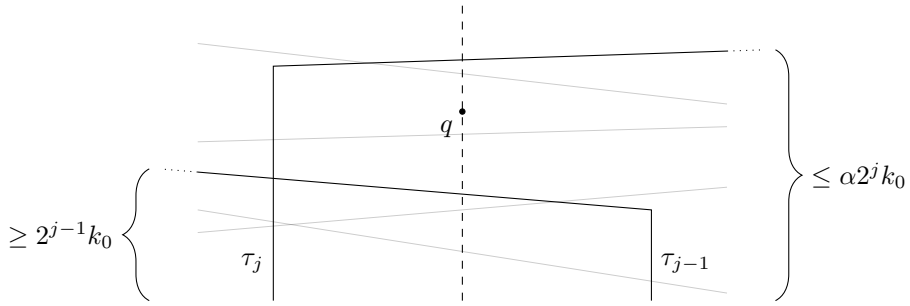

Figure 2 The situation after walking up for q and the number of hyperplanes of $\mathrm{CL}(\tau_j)$ intersecting.

substructure of n' elements consists of a hierarchy of vertical shallow cuttings $\{\Lambda_j\}_{0\leq j\leq m+1}$ with $m \in \mathcal{O}(\log n')$. Let $k_j = 2^j k_0$, n_j be the number of planes in Λ_{j+1}, and k_0, α be constants. Λ_{m+1} consists of a single prism covering all of $\mathbb{R}^3$. Λ_j for $j < m+1$ consists of a vertical k_j-shallow $(\alpha k_j/n_j)$-cutting of the hyperplanes of Λ_{j+1}, with hyperplanes intersecting too many prisms removed. Thus, the prisms of Λ_j cover $L_{\leq k_j}$ of the planes of Λ_{j+1} and each conflict list contains at most αk_j hyperplanes. The removed hyperplanes are reinserted into other substructures, such that each hyperplane is in a substructure's Λ_0. A substructure requires $\mathcal{O}(n' \log n')$ space and building it requires $\mathcal{O}(n' \log^2 n')$ time.

Vertical ray shooting queries are done via searching the Λ_0 of all $\mathcal{O}(\log n)$ substructures for the prism intersecting the vertical line containing the given position in $\mathcal{O}(\log n)$ time each. Then, all obtained prisms are searched for the lowest plane along the line in $\mathcal{O}(\log n)$ time.

Insertions are handled with a Bentley-Saxe approach via rebuilding multiple smaller substructures into new substructures periodically. They require $\mathcal{O}(\log^3 n)$ amortized time.

Deletions are handled differently. In all substructures hyperplanes are only marked as deleted and ignored in queries. As described above, queries per substructure are done in their Λ_0 only. Thus, the lowest non-deleted hyperplane along a given vertical line might not be contained in the prism of Λ_0 intersecting said line. It may require the corresponding prism of Λ_1 or an even higher Λ_j, as prisms from higher cuttings can intersect more hyperplanes.

To handle this, when deleting a hyperplane in a substructure all prisms containing it are identified. For each such prism an individual counter is incremented. If it reaches a fraction of $f = \frac{1}{2\alpha}$ of the size of its conflict list, the prism and its hyperplanes are marked as purged in the substructure. When hyperplanes are first marked as purged, they are also reinserted into the data structure anew. Hyperplanes marked as purged are skipped in queries as well.

Consider prisms $\tau_1, \ldots, \tau_{m+1}$ from $\Lambda_1, \ldots, \Lambda_{m+1}$ intersecting a vertical line. The idea behind the purging is that $|\mathrm{CL}(\tau_j)| \leq \alpha k_j$ and Λ_{j-1} covers $L_{\leq k_{j-1}}$ of the hyperplanes of Λ_j, see Figure 2. Thus, a fraction of at least $\frac{k_{j-1}}{\alpha k_j} = f$ of $\mathrm{CL}(\tau_j)$ is intersected by τ_{j-1}. A plane first appearing in $\mathrm{CL}(\tau_j)$ then can only be the lowest along the vertical line if all from $\mathrm{CL}(\tau_{j-1})$ have been deleted and thus a fraction of f of $\mathrm{CL}(\tau_j)$ as well. Then, τ_j would have been purged and its hyperplanes reinserted into other substructures. Also, each hyperplane is contained in at most one substructure in Λ_0 without being marked as purged or deleted.

Deletions require $\mathcal{O}(\log^5 n)$ amortized time and overall $\mathcal{O}(n \log n)$ space is needed.

4 Sampling Hyperplanes

When at least a fraction of all hyperplanes in $\mathrm{CL}(\tau_j)$ is intersected by any τ_{j-1} and is not marked as deleted, we can sample inside $\mathrm{CL}(\tau_j)$ to get a non-deleted hyperplane intersecting

τ_{j-1} with a minimum probability. For this, we lower the purging threshold to $f' = \frac{1}{4\alpha}$.

▶ **Lemma 4.1.** *The Kaplan et al. data structure for hyperplanes [10, Section 7] with $f' = \frac{1}{4\alpha}$ as purging threshold works as desired with the same asymptotic time and space bounds.*

The correctness of the argument in Section 3 (and the proof by Kaplan et al. [10, Lemma 7.6]) is unchanged, as prisms are just purged earlier. The run time analysis [10, Lemma 7.7] requires an adjustment of constants only[1], and the asymptotic space bound is unaffected as well.

▶ **Lemma 4.2.** *Given prisms τ_j of Λ_j and τ_{j-1} of Λ_{j-1} with $j \geq 1$ from the same substructure and intersected by the same vertical line. If τ_j not has been purged with threshold $f' = \frac{1}{4\alpha}$, at least $\frac{1}{4\alpha}|CL(\tau_j)|$ of its hyperplanes are intersected by τ_{j-1} and are not marked as deleted.*

Proof. The prism τ_{j-1} intersects at least $k_{j-1} = 2^{j-1}k_0$ hyperplanes of Λ_j in its interior, as τ_{j-1} is from a vertical k_{j-1}-shallow cutting of the hyperplanes of Λ_j. Due to the vertical line these hyperplanes must all be contained in $\mathrm{CL}(\tau_j)$ as well. See Figure 2. Also, the prism τ_j intersects at most $\alpha k_j = \alpha 2^j k_0$ hyperplanes, as it is built from a $(\alpha k_j/n_j)$-cutting.

Thus, if τ_j has not been purged due to deletions, it must contain a fraction of

$$> \frac{2^{j-1}k_0 - \frac{1}{4\alpha} \cdot \alpha 2^j k_0}{\alpha 2^j k_0} = \frac{2^{j-2}k_0}{\alpha 2^j k_0} = \frac{1}{4\alpha} \tag{1}$$

non-deleted hyperplanes intersecting τ_{j-1}. ◀

Kaplan et al. [10] already observed the following. For each individual substructure, the ceilings of the prisms of Λ_j form a polyhedral terrain $\overline{\Lambda}_j$. Due to the removal of planes between steps during creation the terrain $\overline{\Lambda}_j$ does not necessarily lie below $\overline{\Lambda}_{j+1}$. Nevertheless, the number of hyperplanes in Λ_j below any point is less or equal than the number in Λ_{j+1}. This is in particular valid for those points in or above $\overline{\Lambda}_j$.

To sample a hyperplane not above a given point we walk the Λ_j from $j = 0$ upwards. At each step we locate the vertical prism which contains the point or has it above in $\mathcal{O}(\log n)$ time, as it is done in Λ_0 in the original data structure. In case the point is located *inside* the prism we stop, otherwise we continue. This allows us to apply Lemma 4.2. See Figure 2.

▶ **Theorem 4.3.** *The lower envelope of hyperplanes in $\mathbb{R}^3$ can be maintained dynamically, where each insertion takes $\mathcal{O}(\log^3 n)$ amortized time, each deletion takes $\mathcal{O}(\log^5 n)$ amortized time, vertical ray shooting queries take $\mathcal{O}(\log^2 n)$ time, and sampling a random hyperplane not above a given point takes $\mathcal{O}(\log^3 n)$ expected time, where n is the number of hyperplanes when the operation is performed. The data structure requires $\mathcal{O}(n \log n)$ storage.*

Proof. We construct the data structure for hyperplanes by Kaplan et al. [10, Section 7] with $f' = \frac{1}{4\alpha}$ as the purging threshold and without their memory optimization (which we omitted in Section 3). According to Lemma 4.1 the correctness and asymptotic bounds are unchanged. We construct for all substructures for all Λ_j, $j \geq 1$ point location data structures. This requires $\mathcal{O}(n \log n)$ storage and the run time can be subsumed in the respective creation of the vertical shallow cutting.

Sampling a hyperplane not above a query point q then can be done as follows. For each substructure the first Λ_j and corresponding τ_j are obtained where q is inside τ_j, as described

[1] In the original proof b' has to be chosen larger (e.g. $b' \geq 8\alpha b''$), as purging a prism τ releases $\geq (\frac{b'}{4\alpha} - b'')|\mathrm{CL}(\tau)| \log N$ credits now.

above. In case the prism was purged or the prism is τ_0 and no non-deleted hyperplane lies not above q, this substructure is skipped. This required $\mathcal{O}(\log^3 n)$ time altogether.

All hyperplanes of a substructure's Λ_0 not above q are contained in $\mathrm{CL}(\tau_j)$. Hence, if all substructures were skipped there is no hyperplane not above q. Otherwise, hyperplanes are sampled from all prisms obtained simultaneously, until the result is not marked as deleted, is not marked as purged in its substructure, and is contained in the substructure's Λ_0 (i.e. was not removed during creation).

If we stopped at $j \geq 1$, the top of τ_{j-1} lies not above q, and we can apply Lemma 4.2. Thus, each non-skipped conflict list contains a fraction of at least $\min(f', 1/k_0)$ non-deleted hyperplanes not above q. Recall that each non-deleted hyperplane is contained in exactly one substructure in Λ_0 without being marked as purged. Hence, each non-deleted hyperplane is sampled with equal probability and each sampling returns with a probability of at least $\min(f', 1/k_0) \cdot 1/\mathcal{O}(\log n)$ a valid hyperplane. Thus, we expect $\mathcal{O}(\log n)$ samplings. ◀

▶ **Corollary 4.4.** *Sampling a random disk in $\mathbb{R}^2$ containing a given point from a dynamic set can be implemented with the bounds of Theorem 4.3.*

5 Sampling Disks

Sampling a random disk intersecting a given disk requires another approach, as linearization results in hyperplanes in $\mathbb{R}^4$. Instead, we sample via the distance functions the disks imply.

In addition to the hyperplane lower envelope data structure, Kaplan et al. describe how to create shallow cuttings of totally defined continuous functions $\mathbb{R}^2 \to \mathbb{R}$ of constant description complexity [10, Theorem 8.1, Theorem 8.2]. Afterwards, they plug it as a black box into their hyperplane data structure [10, Theorem 8.3]. Its analysis changes in values only. We adjust the purging threshold and sample as before, while keeping bounds and correctness intact. $\lambda_s(n)$ is the maximum length of a Davenport-Schinzel sequence of order s on n symbols [13].

▶ **Theorem 5.1.** *The lower envelope of totally defined continuous bivariate functions of constant description complexity in $\mathbb{R}^3$, where the lower envelope of any subset has linear complexity, can be maintained dynamically, where each insertion takes $\mathcal{O}(\log^5(n)\lambda_s(\log n))$ amortized expected time, each deletion takes $\mathcal{O}(\log^9(n)\lambda_s(\log n))$ amortized expected time, vertical ray shooting queries take $\mathcal{O}(\log^2 n)$ time, and sampling a random function not above a given point takes $\mathcal{O}(\log^3 n)$ expected time, where n is the number of functions when the operation is performed. The data structure requires $\mathcal{O}(n \log^3 n)$ expected space.*

Using this theorem we can finally sample disks intersecting a given disk and construct a data structure for finding newly revealed disks after deletions.

▶ **Corollary 5.2.** *A set of disks can be maintained dynamically and a random disk sampled intersecting a given disk with the bounds of Theorem 5.1 with $s = 6$.*

Proof. Let d be a disk with center c_d and radius r_d. Then we can represent the distance of any point $p \in \mathbb{R}^2$ from this disk as additively weighted Euclidean metric with $\delta(p, d) = |pc_d| - r_d$, where $|\cdot|$ is the Euclidean distance. The distance functions of multiple disks form a lower envelope of linear complexity [12] and have $s = 6$ [10, Section 9]. Hence, we can build the data structure of Theorem 5.1 with $s = 6$ to maintain the distance functions $\delta(p, d)$ of all disks d. A disk intersecting a given disk q then can be found via sampling a random function not above the point $((c_q)_x, (c_q)_y, r_q)^T$, as every function not above has $\delta(c_q, d) = |c_q c_d| - r_d \leq r_q$. ◀

▶ **Theorem 5.3.** *Let A and B be sets of disks in $\mathbb{R}^2$. We can preprocess A and B, such that elements can be inserted into or deleted from B and elements can be deleted from A while detecting all newly revealed disks of B after each operation. Preprocessing A and B and deleting n disks of A with detecting all newly revealed disks of B requires $\mathcal{O}(m \log^4 |A| + |A| \log^5(|A|)\lambda_6(\log |A|) + n \log^9(|A|)\lambda_6(\log |A|))$ expected time and $\mathcal{O}(|A| \log^3 |A| + m)$ expected space, where m is maximum size of B. Updating B requires $\mathcal{O}(\log^3 |A|)$ expected time.*

Proof. We repeatedly assign each $b \in B$ randomly to an $a \in A$ it intersects. New assignments are made both initially and each time the previous assigned $a \in A$ gets deleted, unless b got revealed. Fix the deletion order $a_{|A|}, \dots a_1$. Each $b \in B$ is reassigned after deleting $a_i \in A$ with probability $1/i$, assuming it intersects all a_i. This results in an expected number of $\leq H_{|A|} \in \mathcal{O}(\log(|A|))$ reassignments per $b \in B$.

We manage A with the data structure of Corollary 5.2. Building it and assigning each $b \in B$ to an $a \in A$ it intersects requires $\mathcal{O}(|A| \log^5(|A|)\lambda_6(\log |A|) + |B| \log^3 |A|)$ expected amortized time. Each deletion in A requires $\mathcal{O}(\log^9(|A|)\lambda_6(\log |A|))$ amortized time plus the time for reassignments. These sum up to $\mathcal{O}(m \log^4 |A|)$ expected time. Updating B needs $\mathcal{O}(\log^3 |A|)$ expected time. The space bound follows from Corollary 5.2. ◀

References

1. P. K. Agarwal and J. Matousek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11(4):393–418, 1994. doi:10.1007/BF02574015.
2. Jon Louis Bentley and James B Saxe. Decomposable searching problems 1. static-to-dynamic transformation. *J. Algorithms*, 1:58, 1980.
3. Timothy M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. *J. ACM*, 57(3):16:1–16:15, 2010. doi:10.1145/1706591.1706596.
4. Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011.
5. Timothy M. Chan and Konstantinos Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. *Discrete Comput. Geom.*, 56(4):866–881, 2016. doi:10.1007/s00454-016-9784-4.
6. David Eppstein, Giuseppe F. Italiano, Roberto Tamassia, Robert Endre Tarjan, Jeffery Westbrook, and Moti Yung. Maintenance of a minimum spanning forest in a dynamic plane graph. *J. Algorithms*, 13(1):33–54, 1992.
7. Monika Rauch Henzinger and Valerie King. Randomized fully dynamic graph algorithms with polylogarithmic time per operation. *J. ACM*, 46:502–516, 1999.
8. 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.
9. Haim Kaplan, Katharina Klost, Kristin Knorr, Wolfgang Mulzer, and Liam Roditty. Deletion only dynamic connectivity for disk graphs. In *Proceedings of the 37th European Workshop on Computational Geometry (EuroCG)*, page 58:1 ff., 2021. Extended abstract.
10. Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. *Discrete Comput. Geom.*, 64(3):838–904, 2020. doi:10.1007/s00454-020-00243-7.
11. Mihai Pătraşcu and Mikkel Thorup. Planning for fast connectivity updates. In *Proc. 48th Annu. IEEE Sympos. Found. Comput. Sci. (FOCS)*, pages 263–271, 2007.
12. Micha Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM Journal on Computing*, 14(2):448–468, 1985.
13. Micha Sharir and Pankaj K. Agarwal. *Davenport-Schinzel sequences and their geometric applications.* Cambridge University Press, 1995.

14 Mikkel Thorup. Near-optimal fully-dynamic graph connectivity. In *Proc. 32nd Annu. ACM Sympos. Theory Comput. (STOC)*, pages 343–350, 2000.

15 Andrew C. Yao and F. Frances Yao. A general approach to D-dimensional geometric queries. In *Proc. 17th Annu. ACM Sympos. Theory Comput. (STOC)*, pages 163–168, 1985. doi:10.1145/22145.22163.