Dynamic Geometric Data Structures via Shallow Cuttings

Timothy M. Chan[*]

May 10, 2020

Abstract

We present new results on a number of fundamental problems about dynamic geometric data structures:

1. We describe the first fully dynamic data structures with sublinear amortized update time for maintaining (i) the number of vertices or the volume of the convex hull of a 3D point set, (ii) the largest empty circle for a 2D point set, (iii) the Hausdorff distance between two 2D point sets, (iv) the discrete 1-center of a 2D point set, (v) the number of maximal (i.e., skyline) points in a 3D point set. The update times are near $n^{11/12}$ for (i) and (ii), $n^{5/6}$ for (iii) and (iv), and $n^{2/3}$ for (v). Previously, sublinear bounds were known only for restricted "semi-online" settings [Chan, SODA 2002].
2. We slightly improve previous fully dynamic data structures for answering extreme point queries for the convex hull of a 3D point set and nearest neighbor search for a 2D point set. The query time is $O(\log^2 n)$, and the amortized update time is $O(\log^4 n)$ instead of $O(\log^5 n)$ [Chan, SODA 2006; Kaplan et al., SODA 2017].
3. We also improve previous fully dynamic data structures for maintaining the bichromatic closest pair between two 2D point sets and the diameter of a 2D point set. The amortized update time is $O(\log^4 n)$ instead of $O(\log^7 n)$ [Eppstein 1995; Chan, SODA 2006; Kaplan et al., SODA 2017].

1 Introduction

Background. *Dynamic* data structures that can support insertions and deletions of data have been a fundamental topic in computational geometry since the beginning of the field. For example, in 1981 an early landmark paper by Overmars and van Leeuwen [31] presented a fully dynamic data structure for *2D convex hulls* with $O(\log n)$ query time and $O(\log^2 n)$ update time; the $\log^2 n$ bound was later improved in a series of work [9, 8, 14] for various basic types of hull queries, e.g., finding extreme points along given directions.

One of the key results in the area was the author's fully dynamic data structure for *3D convex hulls* [12], which was the first to achieve polylogarithmic query and update time for basic types of hull queries. The original solution required $O(\log^2 n)$ query time for extreme point queries, and $O(\log^6 n)$ amortized update time. (A previous solution by Agarwal and Matoušek [5] had $O(n^\varepsilon)$ query or update time for an arbitrarily small constant $\varepsilon > 0$.) Recently Kaplan et al.

[*]Department of Computer Science, University of Illinois at Urbana-Champaign (tmc@illinois.edu). Work supported in part by NSF Grant CCF-1814026. A preliminary version of this work appeared in *Proc. 35th Sympos. Comput. Geom. (SoCG)*, pages 24:1–24:13, 2019.

[25] noted a small modification of the data structure, improving the update time to $O(\log^5 n)$. The result has numerous applications, including dynamic *2D nearest or farthest neighbor search* (by the standard lifting map). Another application is dynamic *2D bichromatic closest pair* (i.e., computing $\min_{p\in P}\min_{q\in Q}\|p-q\|$ for two planar point sets P and Q) or dynamic *2D diameter* (i.e., computing $\max_{p\in P}\max_{q\in P}\|p-q\|$ for a planar point set P): Eppstein [22] gave a clever, general technique reducing dynamic closest/farthest pair problems to dynamic nearest/farthest neighbor search, which increased the update time by a $\log^2 n$ factor; when combined with the above, this yielded an $O(\log^7 n)$ update time bound.

For many other problems, polylogarithmic update time appears more difficult, and getting sublinear update time is already challenging. For example, in SoCG 2001, the author [10] obtained a dynamic data structure for the *width* of a 2D point set with $\widetilde{O}(\sqrt{n})$ amortized update time.[1] (Part of the difficulty is that the width problem is neither "decomposable" nor "LP-type".) Sublinear update time is known for a few other assorted geometric problems, such as *dynamic connectivity* for the intersection graph of geometric objects [17].

In SODA 2002, the author [11] explored still more challenging dynamic geometric problems, including maintaining

(i) the number of vertices and facets of a 3D convex hull, or its volume,

(ii) the *largest empty circle* for a 2D point set (with center restricted to be inside a fixed triangle),

(iii) the *Hausdorff distance* for two 2D point sets P and Q (i.e., computing $\max_{q\in Q}\min_{p\in P}\|p-q\|$ for two planar point set), and

(iv) the *discrete 1-center* of a 2D point set P (i.e., computing $\min_{q\in P}\max_{p\in P}\|p-q\|$).

The paper [11] obtained sublinear results only for the *insertion-only* case and the *off-line* case (where we are given the entire update sequence in advance), or a generalization of both, known as the *semi-online* case (as defined by Dobkin and Suri [21], where we are given the deletion time of an element when it is inserted). The update time bounds were $O^*(n^{7/8})$ for (i) and (ii), and $O^*(n^{5/6})$ for (iii) and (iv).

None of these four problems are "decomposable" (when a point set is decomposed in two subsets, knowing the answers for the subsets is not sufficient to deduce the answer for the overall set). In particular, problem (i) is nontrivial since known methods such as [12] for 3D convex hull queries do not maintain the global hull explicitly, unlike Overmars and van Leeuwen's original data structure for 2D convex hulls. Problem (ii) also seems to require explicit maintenance of a 3D convex hull (lifted from the 2D farthest-point Voronoi diagram). Problems (iii) and (iv) are max-min or min-max problems, and lack the symmetry of min-min and max-max problems that enable Eppstein's technique. For all these problems, the fully dynamic case has remained open.

New results.

1. We present the first fully dynamic data structures with sublinear update time for Problems (i)–(iv). The amortized update time bounds are $O^*(n^{11/12})$ for (i) and (ii), and $O^*(n^{5/6})$ for (iii) and (iv).

[1] Throughout the paper, we use the $\widetilde{O}$ notation to hide polylogarithmic factor, and O^* notation to hide n^ε factors for an arbitrarily small constant $\varepsilon > 0$.

The approach is general enough to be applicable to many more problems; for example, we can maintain the number of maximal or "skyline" points (points that are not dominated by other points) in a 3D point set in $\widetilde{O}(n^{2/3})$ amortized time.

2. For basic 3D convex hull queries (e.g., extreme point queries) and 2D nearest neighbor search, as mentioned, Kaplan et al. [25] have lowered the amortized update time of the author's fully dynamic data structure [12], from $O(\log^6 n)$ to $O(\log^5 n)$. We describe a further logarithmic-factor improvement, from $O(\log^5 n)$ to $O(\log^4 n)$.

   Although this improvement is admittedly small, the importance of the result stems from its many applications [12]; for example, we can now compute the *convex (or onion) layers* of a 3D point set in $O(n \log^4 n)$ time, and the *k-level* in an arrangement of planes in 3D in $O(n \log n + f \log^4 n)$ time where f is the output size.

3. For bichromatic closest pair and diameter in 2D, combining Eppstein's technique [22] with the above new result on dynamic nearest neighbor search already gives a slightly improved amortized update time of $O(\log^6 n)$. We describe a further, more substantial improvement that eliminates the two extra logarithmic factors caused by Eppstein's technique [22]. The new update time bound is $O(\log^4 n)$.

   Dynamic bichromatic closest pair has applications to other problems. For example, we can now maintain the Euclidean minimum spanning tree of a 2D point set with $O(\log^6 n)$ amortized update time by using another reduction of Eppstein [22] combined with known results for dynamic minimum spanning trees for graphs [24].

Techniques. The common thread in all of our new methods is the use of *shallow cuttings*: Let H be a set of n hyperplanes in $\mathbb{R}^d$. The *level* of a point q refers to the number of hyperplanes of H strictly below q. A (k, K)-*shallow cutting* is a collection of cells covering all points of level at most k, such that each cell intersects at most K hyperplanes. The *conflict list* H_Δ of a cell Δ refers to the subset of all hyperplanes of H intersecting Δ.

Matoušek [28] proved the existence of shallow cuttings with small number of cells. Specifically, in 3D, the main lemma can be stated as follows:[2]

Lemma 1.1. (Shallow Cutting Lemma) *Given a set H of n planes in $\mathbb{R}^3$ and a parameter $k \in [1, n]$, there exists a $(k, O(k))$-shallow cutting with $O(n/k)$ cells, where each cell is a "downward" tetrahedron containing $(0, 0, -\infty)$. The cutting, together with the conflict lists of all its cells, can be constructed in $O(n \log n)$ time.*

The construction time was first shown by Ramos [32] with a randomized algorithm. Later, Chan and Tsakalidis [18] obtained the first $O(n \log n)$-time deterministic algorithm.

To see how static shallow cuttings may be useful for dynamic geometric data structures, observe that most of the problems considered here are related to the lower envelope of a dynamic set of planes in $\mathbb{R}^3$ (via duality or the standard lifting transformation). Usually, the bottleneck lies in deletions rather than insertions. Basically, a shallow cutting provides a compact implicit representation of the $(\leq k)$-level, which is guaranteed to cover the lower envelope even when up to k deletions have occurred.

[2]Matoušek's original formulation in $\mathbb{R}^d$ states the existence of a $(k, n/r)$-shallow cutting with $O(r^{\lfloor d/2 \rfloor}(1 + kr/n)^{\lceil d/2 \rceil})$ cells.

A further idea behind all our solutions is to classify planes into two types, those that intersect few cells of the shallow cutting, and those that intersect many cells. The latter type of planes may be bad in that they slow down updates, but the key is to observe that there are not too many bad elements.

The new sublinear solutions to Problems (i)–(iv), described in Sections 2–3, are obtained by incorporating the shallow cutting idea with the previous techniques from [11], based on periodic rebuilding. The entire solution is conceptually not complicated at all, and the description for Problem (i) fits in under two pages, assuming the availability of known range searching structures. As are typical in other works on data structures with sublinear update time with "funny" exponents, parameters are judiciously chosen to balance several competing costs.

The shallow cutting idea has actually been exploited before in dynamic data structures for basic 3D convex hull queries: Agarwal and Matoušek [5] used shallow cuttings recursively, which caused some loss of efficiency, while the author [12] used a hierarchy of shallow cuttings, for logarithmically many values of k. The above application of shallow cuttings to Problems (i)–(iv) is even more elementary—we only need a single cutting. (This makes it all the more embarassing that the idea was missed till now.)

For basic 3D convex hull queries and 2D nearest neighbor search, our improvement is less innovative. Described in Section 4 (which can be read independently of the previous sections), it is based on the author's original data structure [12], with Kaplan et al.'s logarithmic-factor improvement [25], plus one extra idea to remove a second logarithmic factor: the main observation is that Chan and Tsakalidis's algorithm for shallow cuttings [18] can construct an entire hierarchy of $O(\log n)$ cuttings in $O(n \log n)$ time, not just a single cutting. However, the hierarchy needed for the data structure in [12] requires some planes to be pruned as we go from one cutting to the next, so Chan and Tsakalidis's algorithm cannot be applied immediately. Still, we show that some nontrivial but technical changes (as explained in the appendix) can fix the problem.

For 2D bichromatic closest pair and diameter, our $\log^2 n$-factor improvement, described in Section 5, is a bit more interesting. We still do not know how to improve Eppstein's general reduction [22] from dynamic closest pair to dynamic nearest neighbor search,[3] but intuitively the blind combination of Eppstein's technique with the author's dynamic data structure for 2D nearest neighbor search seems wasteful, since both share some commonalities (both are sophisticated variants of the *logarithmic method* [7], and both handle deletions via re-insertions of elements into smaller subsets). To avoid the redundancy, we show how to directly modify our dynamic data structure for 2D nearest neighbor search to solve the dynamic 2D bichromatic closest pair problem. The resulting modification completely bypasses Eppstein's "conga line" structure [22, 23], and turns out to cause no increase to the $O(\log^4 n)$ update bound.

2 Dynamic 3D Convex Hull Size

We begin with our new sublinear-time fully dynamic data structure for maintaining the number of vertices/facets of the convex hull of a dynamic 3D point set. The solution is based on the use of shallow cuttings (Lemma 1.1) and the author's previous semi-online data structure [11].

[3] See [15] for a new reduction, discovered after the present work, which still requires two extra logarithmic factors in general.

Theorem 2.1. *We can maintain the number of vertices, edges, and facets for the convex hull of a dynamic set of n points in $\mathbb{R}^3$, in general position, with $O^*(n)$ preprocessing time and $O^*(n^{11/12})$ amortized insertion and deletion time.*

Proof. It suffices to maintain the number of convex hull facets, which determines the number of vertices and edges (assuming general position). It suffices to compute the number of upper hull facets, since by symmetry we can compute the number of lower hull facets. We describe our solution in dual space, where the problem is to compute the number of vertices in $\mathrm{LE}(H)$ for a dynamic set H of n planes in $\mathbb{R}^3$.

Let k and s be parameters to be set later. We divide the update sequence into phases of k updates each. We maintain a decomposition of the set H into a deletion-only set H_0 and a small set H_{bad} of "bad" planes.

Preprocessing for each phase. At the beginning of each phase, we construct a $(k, O(k))$-shallow cutting Γ of H with $O(n/k)$ cells, together with all their conflict lists, by Lemma 1.1. We set

$$H_0 = \{h \in H : h \text{ intersects at most } n/s \text{ cells}\} \quad \text{and} \quad H_{\mathrm{bad}} = H - H_0.$$

Since the total conflict list size is $O(n/k \cdot k) = O(n)$ and each plane in H_{bad} participates in at least n/s conflict lists, we have $|H_{\mathrm{bad}}| = O(s)$.

Let V_0 and E_0 be the set of vertices and edges of the portion of $\mathrm{LE}(H_0)$ covered by Γ, respectively. There are $O(k)$ such vertices and edges per cell of Γ, and hence, $|V_0|, |E_0| = O(n/k \cdot k) = O(n)$. We preprocess V_0 and E_0 in $O^*(n)$ time by known range searching and intersection searching techniques, so that

- we can count the number of points in V_0 inside a query tetrahedron in $O^*(n^{2/3})$ time (this is 3D simplex range searching) [2, 13, 27];
- we can count the number of line segments in E_0 intersecting a query triangle in $O^*(n^{3/4})$ time (as noted in [11], we can first solve the case of lines and query halfplanes in $\mathbb{R}^3$ by mapping lines to 4-dimensional points using Plücker coordinates, and then applying known results on semi-algebraic range searching [4] in $\mathbb{R}^4$; we can then extend the solution to line segments and query triangles by a multi-level data structure [2]).

These data structures can support insertions and deletions of points in V_0 and line segments in E_0 in $O^*(1)$ time each. In addition, we preprocess H_0 in a known dynamic lower envelope data structure in $\widetilde{O}(n)$ time, to support ray shooting queries in $\mathrm{LE}(H_0)$ in $\widetilde{O}(1)$ time and deletions in $\widetilde{O}(1)$ time (e.g., see [12] or Section 4). The total preprocessing time per phase is $O^*(n)$. Amortized over k updates, the cost is $O^*(n/k)$.

Inserting a plane h. We just insert h to the list H_{bad}. Note that $|H_{\mathrm{bad}}| = O(s + k)$ at all times, since there are at most k insertions per phase.

Deleting a plane h from H_{bad}. We just remove h from the list H_{bad}.

Deleting a plane h from H_0. We consider each cell $\Delta \in \Gamma$ intersected by h, and compute $\mathrm{LE}((H_0)_\Delta)$ from scratch in $O(k \log k)$ time (since $|(H_0)_\Delta| = O(k)$). As the number of cells intersected by h is at most n/s, this computation requires $O^*(kn/s)$ total time. The sets V_0 and E_0 undergo at most $O(kn/s)$ changes, and their associated data structures can be updated in $O^*(kn/s)$ time.

Computing the answer. To compute the number of vertices of $\mathrm{LE}(H) = \mathrm{LE}(H_0 \cup H_{\mathrm{bad}})$, we first construct $\mathrm{LE}(H_{\mathrm{bad}})$ in $O((s+k)\log(s+k))$ time, and triangulate all its $O(s+k)$ faces. For each triangle τ in this triangulation:

- we count the number of vertices of V_0 that lie directly below τ, in $O^*(n^{2/3})$ time; and
- we count the number of edges of E_0 that intersect τ, in $O^*(n^{3/4})$ time.

We sum up all these counts. In addition, for each edge of $\mathrm{LE}(H_{\mathrm{bad}})$, we test whether it intersects $\mathrm{LE}(H_0)$, and if so, add the number of intersections to the count; the number of intersections is at most 2 (since $\mathrm{LE}(H_0)$ is the boundary of a convex polyhedron), and we can find them by at most 2 ray shooting queries in $\widetilde{O}(1)$ time. For each vertex of $\mathrm{LE}(H_{\mathrm{bad}})$, we test whether it is underneath $\mathrm{LE}(H_0)$ by vertical ray shooting in $\widetilde{O}(1)$ time, and increment the count if true. Note that $\mathrm{LE}(H)$ is covered by Γ at all times, since there are at most k deletions per phase. The overall count thus gives the answer. The total time to compute the answer is $O^*((s+k)n^{3/4})$.

Analysis. The overall amortized update time is

$$O^*(n/k + kn/s + (s+k)n^{3/4}).$$

The theorem follows by setting $s = k^2$ and $k = n^{1/12}$. $\square$

The preprocessing time can be made $O(n \log n)$ and space made $O(n)$ by increasing the update time by an n^{ε} factor, via known trade-offs for range/intersection searching (with larger-degree partition trees). The method can be deamortized, using existing techniques [30].

The same method can be adapted to maintain the sum or maximum of $f(v)$ over all vertices v of $\mathrm{LE}(H)$, for a general class of functions f. Instead of range counting, we store the set V_0 of points for range sum or range maximum queries (which have similar complexity as range counting). For the set E_0 of line segments, the base level of its multi-level data structure requires data structures $\mathcal{S}_L$ for each canonical subset L of lines in $\mathbb{R}^3$, so that we can return the sum or maximum of $f(\ell \cap h)$ over all $\ell \in L$ for a query plane h in $O^*(|L|^{\alpha})$ time, supporting insertions and deletions in L in $O^*(1)$ time. If $\alpha \le 3/4$, the final time bound of our algorithm remains $O^*(n^{11/12})$.

Theorem 2.2. *We can maintain the volume of the convex hull for a dynamic set of n points in $\mathbb{R}^3$, with $O^*(n)$ preprocessing time and $O^*(n^{11/12})$ amortized insertion and deletion time.*

Proof. Let o be a fixed point sufficiently far below all the input points. It suffices to maintain the sum of the volume of the tetrahedra $op_1p_2p_3$ over all upper hull facets $p_1p_2p_3$, since by symmetry we can maintain a similar sum for lower hull facets and subtract. We map each point p to its dual plane h_p. Then the problem fits in the above framework, with $f(v)$ equal to the volume of the tetrahedron $op_1p_2p_3$ for a vertex v defined by the planes $h_{p_1}, h_{p_2}, h_{p_3}$. Let $\ell_{p_1p_2}$ denote the line defined by the planes h_{p_1} and h_{p_2}. To design the data structure $\mathcal{S}_L$ for a given canonical subset L of lines in $\mathbb{R}^3$, observe that $f(\ell_{p_1p_2} \cap h_p)$ is a linear function in the 3 coordinates of p, since the volume of op_1p_2p can be expressed as a determinant. Thus, the sum of $f(\ell_{p_1p_2} \cap h_p)$ over all $\ell_{p_1p_2} \in L$ is also a linear function in three variables, and can be evaluated in $O(1)$ time for any query point p, after precomputing the four coefficients.

However, the above assumes that p lies on a fixed (left or right) side of op_1p_2 for all $\ell_{p_1p_2} \in L$ (otherwise we would need to sum the absolute values of the determinants). To fix this issue, we first find the subset of all lines $\ell_{p_1p_2} \in L$ such that the query point p is left (resp. right) of op_1p_2.

This step reduces to halfspace range searching in dual space—the dimension is in fact 2 since o is fixed. By “find”, we mean expressing the answer as a union of smaller canonical subsets. We can then apply the above to these smaller canonical subsets. The query cost for 2-dimensional halfspace range searching [2, 27] is $O^*(\sqrt{|L|})$ with a near-linear space data structure that supports updates in $O^*(1)$ time. Hence, we get $\alpha = 1/2$. ☐

Theorem 2.3. *We can maintain the largest empty circle of a dynamic set of n points in $\mathbb{R}^2$, under the restriction that the center lies inside a given triangle Δ_0, with $O^*(n)$ preprocessing time and $O^*(n^{11/12})$ amortized insertion and deletion time.*

Proof. By the standard lifting transformation, map each input point $p = (a, b) \in \mathbb{R}^2$ to the plane h_p with equation $z = -2ax - 2by + a^2 + b^2$ in $\mathbb{R}^3$. Add 3 near-vertical planes along the edges of Δ_0. The largest empty circle problem reduces to finding a vertex $v = (x, y, z)$ of the lower envelope of these planes, maximizing $f(v) = x^2 + y^2 + z$. To design the data structure $\mathcal{S}_L$ for a canonical subset L of lines in $\mathbb{R}^3$, we map each line to a 4-dimensional point using Plücker coordinates. Deciding whether $\max_{\ell \in L} f(\ell \cap h) \geq r$ for a query plane h and a query value r reduces to semi-algebraic range searching in this 4-dimensional point set, and can be done in $O^*(|L|^{3/4})$ time [5, 6, 26] with a near-linear space data structure that supports updates in $O^*(1)$ time. By standard techniques (e.g., parametric search, as in [3]), we can find $\max_{\ell \in L} f(\ell \cap h)$ for a query plane h in the same time bound, ignoring polylogarithmic factors. Hence, we get $\alpha = 3/4$. ☐

We can obtain sublinear update time bounds for other similar problems, e.g., maintaining the minimum/maximum-area Delaunay triangle of a dynamic 2D point set. Another application is computing the number of maximal points, also called “skyline points” (which are points not dominated by other points), in a dynamic 3D point set:

Theorem 2.4. *We can maintain the number of maximal points in a dynamic set P of n points in $\mathbb{R}^3$, with $\widetilde{O}(n)$ preprocessing time and $\widetilde{O}(n^{2/3})$ amortized insertion and deletion time.*

Proof. The maximal points are vertices of the upper envelope of orthants $(-\infty, a] \times (-\infty, b] \times (-\infty, c]$ over all input points $(a, b, c) \in P$ (this upper envelope is an orthogonal polyhedron). As is well known, an analogue of the shallow cutting lemma holds for such orthants in 3D (in fact, there is a transformation that maps such orthants to halfspaces in 3D); for example, see [16]. The same method can thus be adapted. In fact, it can be simplified. The data structure for V_0 is for orthogonal range searching [19], which has $\widetilde{O}(1)$ query and update time. The data structure E_0 is not needed. The overall update time becomes

$$\widetilde{O}(n/k + kn/s + (s + k)).$$

The theorem follows by setting $s = k^2$ and $k = n^{1/3}$. ☐

We can similarly maintain the volume of a union of n boxes in $\mathbb{R}^3$ in the case when all the boxes have a common corner point at the origin (this is called the *hypervolume indicator* problem) with $\widetilde{O}(n^{2/3})$ update time (previously, an $\widetilde{O}(\sqrt{n})$ bound was known only in the semi-online setting [11]).

3 Dynamic 2D Hausdorff Distance

The method in Section 2 can also be adapted to solve the dynamic 2D Hausdorff distance problem:

Theorem 3.1. *We can maintain the Hausdorff distance between two dynamic sets P and Q of at most n points in $\mathbb{R}^2$, with $O^*(n)$ preprocessing time and $O^*(n^{5/6})$ amortized insertion and deletion time.*

Proof. By the standard lifting transformation, map each point $p = (a, b) \in P$ to the plane h_p with equation $z = -2ax - 2by + a^2 + b^2$ in $\mathbb{R}^3$. Let H be the resulting set of planes. For each point $q \in Q$, let $\lambda_H(q)$ denote the point on LE(H) at the vertical line at q. The problem is to find the maximum of $f(\lambda_H(q))$ over all $q \in Q$, where $f(x, y, z) = x^2 + y^2 + z$, for a dynamic set H of at most n planes and a dynamic set Q of at most n points.

Let k and s be parameters to be set later. We divide the update sequence into phases of k updates each. We maintain a decomposition of the set H into a deletion-only set H_0 and a small set H_{bad} of "bad" planes, and a decomposition of the set Q into a deletion-only set Q_0 and a small set Q_{bad} of "bad" points.

Preprocessing for each phase. At the beginning of each phase, we construct a $(k, O(k))$-shallow cutting Γ of H with $O(n/k)$ cells, together with all their conflict lists, by Lemma 1.1. We further subdivide the cells to ensure that each cell contains at most k points of Q in its xy-projection; this can be done by $O(n/k)$ additional vertical plane cuts, so the number of cells remains $O(n/k)$. We set

$$H_0 = \{h \in H : h \text{ intersects at most } n/s \text{ cells}\} \quad \text{and} \quad H_{\text{bad}} = H - H_0.$$

Since the total conflict list size is $O(n/k \cdot k) = O(n)$, we have $|H_{\text{bad}}| = O(s)$.

We set $Q_0 = Q$. We compute $\lambda_{H_0}(q)$ for all $q \in Q$ in $O(n \log n)$ time. Let Λ_0 be the subset of points in $\{\lambda_{H_0}(q) : q \in Q\}$ covered by Γ. We preprocess the point set Λ_0 in known 3D simplex range searching data structures [2, 13, 27] in $O^*(n)$ time, to support the following queries in $O^*(n^{2/3})$ time:

- compute the maximum of $f(v)$ over all points $v \in \Lambda_0$ inside a query tetrahedron;
- compute the maximum of $f(\lambda_{\{h_p\}}(x, y))$ over all points $v = (x, y, z) \in \Lambda_0$ inside a query tetrahedron for a query plane h_p; note that maximizing $f(\lambda_{\{h_p\}}(x, y))$ is equivalent to maximizing the distance from (x, y) to p (so we can use a 2-level data structure, combining simplex range searching with 2D farthest neighbor searching).

The $O^*(n^{2/3})$ query cost can be improved, by observing that the point set Λ_0 is in convex position. Sharir and Zaban [33] has shown that in the convex-position case (called "1-shallow" in their paper), the cost of 3D simplex range searching reduces to $O^*(\sqrt{n})$.

The data structures can support insertions and deletions of points in Λ_0 in $O^*(1)$ time each. In addition, we preprocess H_0 in a known dynamic lower envelope data structure in $\widetilde{O}(n)$ time, to support ray shooting queries in LE(H_0) in $\widetilde{O}(1)$ time and deletions in $\widetilde{O}(1)$ time (e.g., see [5] or Section 4).

Inserting a plane h to H or a point q to Q. We just insert h to the list H_{bad} or q to the list Q_{bad}. Note that $|H_{\text{bad}}| = O(s + k)$ and $|Q_{\text{bad}}| = O(k)$ at all times.

Deleting a plane h from H_{bad} or a point q from Q_{bad}. We just remove h from the list H_{bad} or q from the list Q_{bad}.

Deleting a point q from Q_0. We just remove $\lambda_{H_0}(q)$ from the set Λ_0 in $O^*(1)$ time.

Deleting a plane h from H_0. We consider each cell $\Delta \in \Gamma$ intersected by h, and compute $\lambda_{(H_0)_\Delta}(q)$ for all $q \in Q$ in the xy-projection of Δ from scratch in $O(k \log k)$ time (since Δ is intersected by $O(k)$ planes in H and contains $O(k)$ points of Q in its xy-projection). As the number of cells intersected by h is at most n/s, this computation takes $O^*(kn/s)$ total time. The set Λ_0 undergoes at most $O(kn/s)$ changes, and its associated data structures can be updated in $O^*(kn/s)$ time.

Computing the answer. To compute the maximum of $f(\lambda_H(q))$ over all $q \in Q$, we first construct $\mathrm{LE}(H_{\mathrm{bad}})$ in $O((s+k)\log(s+k))$ time, and triangulate all its $O(s+k)$ faces. For each triangle τ in this triangulation:

- We compute the maximum of $f(v)$ over all $v = (x, y, z) \in \Lambda_0$ that lie directly below τ, in $O^*(n^{2/3})$ time. Note that for all such v, the $\lambda_H(x, y) = \lambda_{H_0}(x, y) = v$.
- We let h be the plane through τ and compute the maximum of $f(\lambda_{\{h\}}(x, y))$ over all $v = (x, y, z) \in \Lambda_0$ that lie directly above τ, in $O^*(n^{2/3})$ time. Note that for all such v, $\lambda_H(x, y) = \lambda_{\{h\}}(x, y)$.

In addition, for each $q \in Q_{\mathrm{bad}}$, we compute $\lambda_H(q)$ by vertical ray shooting in $\mathrm{LE}(H_0)$ and $\mathrm{LE}(H_{\mathrm{bad}})$ in $O^*(1)$ time; we take the maximum of $f(\lambda_H(q))$ for these points. Note that $\mathrm{LE}(H)$ is covered by Γ at all times, since there are at most k deletions per phase. The overall maximum thus gives the answer. The total time to compute the answer is $O^*((s+k)\sqrt{n})$.

Analysis. The overall amortized update time is

$$O^*(n/k + kn/s + (s+k)\sqrt{n}).$$

The theorem follows by setting $s = k^2$ and $k = n^{1/6}$. $\square$

We can similarly solve the dynamic 2D discrete 1-center problem, by switching lower with upper envelopes and maximum with minimum:

Theorem 3.2. *We can maintain the discrete 1-center of a dynamic set of n points in $\mathbb{R}^2$, with $O^*(n)$ preprocessing time and $O^*(n^{5/6})$ amortized insertion and deletion time.*

Incidentally, the above idea of using 3D simplex range searching in the convex-position case [33] can also improve the author's previous result in the semi-online setting [11] from $O^*(n^{5/6})$ to $O^*(n^{3/4})$ update time.

It remains open whether the dynamic Hausdorff distance and discrete 1-center problem in dimensions $d \geq 3$ can similarly be solved in sublinear time. The author's previous paper [11] gave an $O^*(n^{1-1/(d+1)(\lceil d/2 \rceil+1)})$-time algorithm but only in the semi-online setting. In higher dimensions, the size of shallow cuttings becomes too large for the approach to be effective.

4 Dynamic 3D Convex Hull Queries

In this section, we present a slightly improved data structure for extreme point queries for a dynamic 3D convex hull, by combining the author's previous data structure [12] (as refined by Kaplan et al. [25]) with a modification of Chan and Tsakalidis's algorithm for constructing a hierarchy of shallow cuttings [18].

To describe the latter, we need a definition: Given a set H of n planes in $\mathbb{R}^3$ and a collection Γ_{in} of cells, a Γ_{in}*-restricted* (k, K)*-shallow cutting* is a collection Γ_{out} of cells covering $\{p \in \mathbb{R}^3 : p$ is covered by Γ_{in} and has level at most $k\}$, such that each cell in Γ_{out} intersects at most K planes. We say that a cutting is *convex* if its cells are interior-disjoint and their union is a convex polyhedron. We note that Chan and Tsakalidis's algorithm, with some technical modifications, can prove the following lemma. The proof requires knowledge of Chan and Tsakalidis's paper, and is deferred to the appendix.

Lemma 4.1. *There exist constants b, c, and c' such that the following is true: For a set H of at most n planes in $\mathbb{R}^3$ and a parameter $k \in [1, n]$, given a convex $(-\infty, cbk)$-shallow cutting*[4] *Γ_{in} with at most $c'n/(bk)$ downward cells, together with their conflict lists, we can construct a convex Γ_{in}-restricted (k, ck)-shallow cutting Γ_{out} with at most $c'n/k$ downward cells, together with their conflict lists, in $O(n + (n/k)\log(n/k))$ deterministic time.*

We now redescribe the author's previous data structure [12] for 3D extreme point queries, with slight changes to incorporate Lemma 4.1. The redescription uses a recursive form of the logarithmic method [7], which should be a little easier to understand than the original description.

Theorem 4.2. *We can maintain a set of n points in $\mathbb{R}^3$, with $O(n \log n)$ preprocessing time, $O(\log^2 n)$ amortized insertion time, and $O(\log^4 n)$ amortized deletion time, so that we can find the extreme point of the convex hull along any query direction in $O(\log^2 n)$ time.*

Proof. We describe our solution in dual space, where we want to answer vertical ray shooting queries for LE(H), i.e., find the lowest plane of H at a query vertical line, for a dynamic set H of n planes in $\mathbb{R}^3$.

Preprocessing. Our preprocessing algorithm is given by the pseudocode below (ignoring trivial base cases), with the constants b, c, c' from Lemma 4.1:[5]

```
preprocess($H$):
1.  $H_0 = H$,  $\Gamma_0 = \{\mathbb{R}^3\}$,  $\ell = \log_b n$
2.  for $i = 1, \ldots, \ell$ do {
3.     $\Gamma_i$ = a convex $\Gamma_{i-1}$-restricted $(n/b^i, cn/b^i)$-shallow cutting of $H_{i-1}$ with at most $c'b^i$ cells
4.     $H_i = H_{i-1} - \{h \in H : h$ intersects more than $2cc'\ell$ cells of $\Gamma_1 \cup \cdots \cup \Gamma_i\}$
5.     for each $\Delta \in \Gamma_i$, compute the conflict list $(H_i)_\Delta$ and initialize $k_\Delta = 0$
    }
6.  preprocess $H_\ell$ for static vertical ray shooting
7.  $H_{\text{bad}} = H - H_\ell$
8.  preprocess($H_{\text{bad}}$)
```

Note that Γ_{i-1} is a $(-\infty, cn/b^{i-1})$-shallow cutting of H_{i-2}, and consequently a $(-\infty, cn/b^{i-1})$-shallow cutting of H_{i-1}, since $H_{i-1} \subseteq H_{i-2}$. Given Γ_{i-1} and its conflict lists, we can thus apply Lemma 4.1 to compute Γ_i and its conflict lists, in $O(n + b^i \log b^i)$ time. The total time for lines 1–5 is $O(n \log n + \sum_{i=1}^{\ell} b^i \log b^i) = O(n \log n)$. Line 6 takes $O(n \log n)$ time (by a planar point location method [19]).

[4] In a $(-\infty, k)$-shallow cutting, the cells are not required to cover any particular region.

[5] Line 4 is where Kaplan et al.'s improvement lies [25]. The original data structure from [12] basically had $H_i = H_{i-1} - \{h \in H : h$ intersects more than $2cc'\ell$ cells of $\Gamma_i\}$.

We claim that $|H_{\text{bad}}| \leq n/2$. To see this, consider each $h \in H_{\text{bad}}$. Let i be the index with $h \in H_{i-1} - H_i$. Then h intersects more than $2cc'\ell$ cells of $\Gamma_1 \cup \cdots \cup \Gamma_i$; send a charge from h to each of these cells. Each cell in Γ_j receives charges only from planes in H_{j-1} that intersect the cell. Thus, the total number of charges is at least $2cc'\ell|H_{\text{bad}}|$ and is at most $\sum_{j=1}^{\ell} cn/b^j \cdot c'b^j = cc'\ell n$. The claim follows. The preprocessing time thus satisfies the recurrence $P(n) \leq P(n/2)+O(n\log n)$, which gives $P(n) = O(n \log n)$.

Inserting a plane h. We simply insert h to H_{bad} recursively. When $|H_{\text{bad}}|$ reaches $3n/4$, we rebuild the data structure for H. It takes $\Omega(n)$ updates for a rebuild to occur. The amortized insertion time thus satisfies the recurrence $I(n) \leq I(3n/4) + O(P(n)/n) = I(3n/4) + O(\log n)$, which gives $I(n) = O(\log^2 n)$.

Deleting a plane h. The deletion algorithm is as follows:

```
    delete($H, h$):
1.  for $i = 1, \ldots, \ell$ do
2.      for each $\Delta \in \Gamma_i$ with $h \in (H_i)_\Delta$ do {
3.          increment $k_\Delta$
4.          if $k_\Delta \geq n/b^{i+1}$ then
5.              for all $h \in (H_i)_\Delta$ that are still in $H$ but not yet in $H_{\text{bad}}$, insert $h$ to $H_{\text{bad}}$
        }
6.  if $h \in H_{\text{bad}}$ then delete($H_{\text{bad}}, h$)
```

Let i be the largest index with $h \in H_i$. Then h intersects at most $2cc'\ell = O(\log n)$ cells of $\Gamma_1 \cup \cdots \cup \Gamma_i$. Thus, in each deletion, lines 3–5 are executed $O(\log n)$ times.

In lines 3–5, it takes n/b^{i+1} increments of k_Δ to cause the $|(H_i)_\Delta| \leq cn/b^i$ planes to be inserted to H_{bad}. Thus, each increment triggers $O(1)$ amortized number of insertions to H_{bad}, and so a deletion triggers $O(\log n)$ amortized number of insertions to H_{bad}. The amortized deletion time thus satisfies the recurrence $D(n) \leq D(3n/4) + O(\log n)I(3n/4) = D(3n/4) + O(\log^3 n)$, which gives $D(n) = O(\log^4 n)$.

Answering the query for a vertical line q. We first answer the query for the static set H_ℓ in $O(\log n)$ time (by planar point location); if the returned plane has already been deleted, ignore the answer. We then recursively answer the query for H_{bad}, and return the lowest of all the planes found. The query time satisfies the recurrence $Q(n) \leq Q(3n/4) + O(\log n)$, which gives $Q(n) = O(\log^2 n)$.

Correctness of the query algorithm. To prove correctness, let h^* be the lowest plane at q and $v^* = h^* \cap q$. If $h^* \in H_{\text{bad}}$, correctness follows by induction. So, assume that $h^* \notin H_{\text{bad}}$.

If v^* is covered by Γ_ℓ, say, by the cell $\Delta \in \Gamma_\ell$, then either v^* is on $\text{LE}(H_\ell)$, in which case the algorithm would have correctly found h^*, or some plane in $(H_\ell)_\Delta$ has been deleted from H, in which case all active planes of $(H_\ell)_\Delta$, including h^*, would have been inserted to H_{bad}.

Otherwise, let i be an index such that v^* is not covered by Γ_i but is covered by Γ_{i-1}, say, by the cell $\Delta \in \Gamma_{i-1}$. Since Γ_i is a Γ_{i-1}-restricted $(n/b^i, cn/b^i)$-shallow cutting of H_{i-1}, it follows that v^* must have level more than n/b^i in H_{i-1}. In order for v^* to be the answer, the more than n/b^i planes of H_{i-1} below v^* must have been deleted from H. But then all active planes of $(H_{i-1})_\Delta$, including h^*, would have been inserted to H_{bad}. $\square$

By the standard lifting transformation, we obtain:

Corollary 4.3. *We can maintain a set of n points in $\mathbb{R}^2$, with $O(n \log n)$ preprocessing time, $O(\log^2 n)$ amortized insertion time, and $O(\log^4 n)$ amortized deletion time, so that we can find the nearest neighbor to any query point in $O(\log^2 n)$ time.*

The space usage in the above data structure is $O(n \log n)$, but can be improved to $O(n)$, by following an idea mentioned in [12] (due to Afshani): instead of storing conflict lists explicitly, generate conflict lists on demand by using a known optimal (static) linear-space data structure for halfspace range reporting [1].

As in [12], the same dynamic data structure can be used to answer other basic types of 3D convex hull queries, for example:

- *Gift wrapping queries* (finding the two tangents of the convex hull with a query line outside the hull): In dual space, this reduces to finding the two intersection points of $\mathrm{LE}(H)$ with a nonvertical query line—this is essentially *nonvertical ray shooting*. By symmetry, it suffices to compute the top intersection point. The same query algorithm and analysis works (except that we store $\mathrm{LE}(H_\ell)$ in a Dobkin–Kirkpatrick hierarchy [20] to support nonvertical ray shooting queries in $O(\log n)$ time). We can check whether the query line actually intersects $\mathrm{LE}(H)$ by verifying that the answer is indeed a point on $\mathrm{LE}(H)$ (which we can do by a vertical ray shooting query). The overall query time is $O(\log^2 n)$ as before.

- *Line-intersection queries* (intersecting the convex hull with a query line): In dual space, this reduces to a *linear programming query* (finding a point on $\mathrm{LE}(H)$ that is extreme along a query direction). As noted in [12], linear programming queries can be reduced to vertical ray shooting by multi-dimensional parametric search [29]; the query time increases to $O(\log^4 n \log^4 \log n)$. (The update time is unchanged, since the data structure is unchanged.)

Many applications follow, as noted in the previous paper [12]. For example, the *smallest enclosing circle* for a dynamic 2D point set can now be maintained in $O(\log^4 n \log^4 \log n)$ time: by the standard lifting transformation, the problem reduces to convex programming over $\mathrm{LE}(H)$ for a dynamic set H of n planes in $\mathbb{R}^3$, and the multi-dimensional parametric search technique is still applicable to convex programming. See [12] for more applications. The dynamic data structure from [14] for 3D *halfspace range reporting* queries can also be immediately improved.

5 Dynamic 2D Bichromatic Closest Pair

We now adapt the data structure in Section 4 to solve the dynamic 2D bichromatic closest pair problem:

Theorem 5.1. *We can maintain the closest pair between two dynamic sets P and Q of at most n points in $\mathbb{R}^2$, with $O(n \log n)$ preprocessing time, $O(\log^2 n)$ amortized insertion time, and $O(\log^4 n)$ amortized deletion time.*

Proof. By the standard lifting transformation, map each input point $p = (a, b)$ to the plane h_p with equation $z = -2ax - 2by + a^2 + b^2$ in $\mathbb{R}^3$. Let $H = \{h_p : p \in P\}$. For each point $q \in Q$, let $\lambda_H(q)$ denote the point on $\mathrm{LE}(H)$ at the vertical line at q. Let $J = \{h_q : q \in Q\}$. For each point $p \in P$, define $\lambda_J(p)$ similarly. We want to compute the minimum of $f(\lambda_H(q))$ over all $q \in Q$, where $f(x, y, z) = x^2 + y^2 + z$, which is equivalent to the minimum of $f(\lambda_J(p))$ over all $p \in P$.

Preprocessing. We maintain a global heap, whose minimum gives the answer. We modify the preprocess(H) algorithm in Section 4:

```
preprocess($H, J$):
1. run lines 1–7 of the preprocess($H$) algorithm on $H$
2. for each $h_q \in J$, add $f(\lambda_{H_\ell}(q))$ to the heap
3. run lines 1–7 of the preprocess($H$) algorithm but with $H$'s replaced by $J$'s
4. for each $h_p \in H$, add $f(\lambda_{J_\ell}(p))$ to the heap
5. preprocess($H_{\mathrm{bad}}, J_{\mathrm{bad}}$)
```

As in Section 4, the preprocessing time satisfies the recurrence $P(n) \le P(n/2) + O(n \log n)$, which gives $P(n) = O(n \log n)$.

Inserting a plane h_p to H. We recursively insert h_p to H_{bad}. We also compute $\lambda_{J_\ell}(p)$ in $O(\log n)$ time (by planar point location), and add $f(\lambda_{J_\ell}(p))$ to the heap.

When $|H_{\mathrm{bad}}|$ or $|J_{\mathrm{bad}}|$ reaches $3n/4$, we rebuild the data structure for H and J. It takes $\Omega(n)$ updates for a rebuild to occur. The amortized insertion time thus satisfies the recurrence $I(n) \le I(3n/4) + O(\log n) + O(P(n)/n) = I(3n/4) + O(\log n)$, which gives $I(n) = O(\log^2 n)$.

Inserting a plane h_q to J. Symmetric to the above.

Deleting a plane h_p from H. We run lines 1–5 of the delete(H, h) algorithm in Section 4 (with $h = h_p$). In the heap, we remove all entries $f(\lambda_{H_\ell}(q))$ that has $\lambda_{H_\ell}(q) = \lambda_{\{h_p\}}(q)$. If $h_p \in H_{\mathrm{bad}}$, we further recursively delete h_p from H_{bad}. We also remove $f(\lambda_{J_\ell}(p))$ from the heap.

For the analysis, we can charge removals of entries from the heap to their corresponding insertions, by amortization. The amortized deletion time thus satisfies the recurrence $D(n) \le D(3n/4) + O(\log n) I(3n/4) = D(3n/4) + O(\log^3 n)$, which gives $D(n) = O(\log^4 n)$.

Deleting a plane h_q from J. Symmetric to the above.

Correctness. Let p^*q^* be the closest pair with $p^* \in P$ and $q^* \in Q$. If both $h_{p^*} \in H_{\mathrm{bad}}$ and $h_{q^*} \in J_{\mathrm{bad}}$, correctness follows by induction. Otherwise, assume without loss of generality that $h_{p^*} \notin H_{\mathrm{bad}}$. (The case $J_{q^*} \notin J_{\mathrm{bad}}$ is symmetric.) Let $v^* = \lambda_H(q^*)$. The rest of the correctness argument is essentially identical to that in Section 4:

If v^* is covered by Γ_ℓ, say, by the cell $\Delta \in \Gamma_\ell$, then either v^* is on LE(H_ℓ), in which case the algorithm would have included $f(\lambda_H(q^*))$ in the heap, or some plane in $(H_\ell)_\Delta$ has been deleted from H, in which case all active planes of $(H_\ell)_\Delta$, including h_{p^*}, would have been inserted to H_{bad}.

Otherwise, let i be an index such that v^* is not covered by Γ_i but is covered by Γ_{i-1}, say, by the cell $\Delta \in \Gamma_{i-1}$. Since Γ_i is a Γ_{i-1}-restricted $(n/b^i, cn/b^i)$-shallow cutting of H_{i-1}, it follows that v^* must have level more than n/b^i in H_{i-1}. In order for v^* to be the answer, the more than n/b^i planes of H_{i-1} below v^* must have been deleted from H. But then all active planes of $(H_{i-1})_\Delta$, including h_{p^*}, would have been inserted to H_{bad}. □

We can similarly solve the diameter problem, by replacing min with max and lower with upper envelopes:

Theorem 5.2. *We can maintain the diameter of a dynamic set of n points in $\mathbb{R}^2$, with $O(n \log n)$ preprocessing time, $O(\log^2 n)$ amortized insertion time, and $O(\log^4 n)$ amortized deletion time.*

Acknowledgement. I thank Sariel Har-Peled for discussions on other problems that indirectly led to the results of this paper. Thanks also to Mitchell Jones for discussions on range searching for points in convex position (used in the proof of Theorem 3.1), Pankaj Agarwal for pointing out an error in an earlier proof of Theorem 2.3, and the referees for all their comments.

References

[1] Peyman Afshani and Timothy M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th ACM-SIAM Sympos. Discrete Algorithms (SODA)*, pages 180–186, 2009. URL: `http://dl.acm.org/citation.cfm?id=1496770.1496791`.

[2] Pankaj K. Agarwal. Simplex range searching and its variants: A review. In M. Loebl, J. Nešetril, and R. Thomas, editors, *Journal through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 1–30. Springer, 2017.

[3] Pankaj K. Agarwal and Jiří Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993. `doi:10.1137/0222051`.

[4] Pankaj K. Agarwal and Jiří Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994. `doi:10.1007/BF02574015`.

[5] Pankaj K. Agarwal and Jiří Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13(4):325–345, 1995. `doi:10.1007/BF01293483`.

[6] Pankaj K. Agarwal, Jiří Matoušek, and Micha Sharir. On range searching with semialgebraic sets. II. *SIAM J. Comput.*, 42(6):2039–2062, 2013. `doi:10.1137/120890855`.

[7] 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`.

[8] Gerth Stølting Brodal and Riko Jacob. Dynamic planar convex hull. In *Proc. 43rd Sympos. Found. Comput. Sci. (FOCS)*, pages 617–626, 2002. `doi:10.1109/SFCS.2002.1181985`.

[9] Timothy M. Chan. Dynamic planar convex hull operations in near-logarithmic amortized time. *J. ACM*, 48(1):1–12, 2001. Preliminary version in FOCS 1999. `doi:10.1145/363647.363652`.

[10] Timothy M. Chan. A fully dynamic algorithm for planar width. *Discrete Comput. Geom.*, 30(1):17–24, 2003. Preliminary version in SoCG 2001. `doi:10.1007/s00454-003-2923-8`.

[11] Timothy M. Chan. Semi-online maintenance of geometric optima and measures. *SIAM J. Comput.*, 32(3):700–716, 2003. Preliminary version in SODA 2002. `doi:10.1137/S0097539702404389`.

[12] 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. Preliminary version in SODA 2006. `doi:10.1145/1706591.1706596`.

[13] Timothy M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47(4):661–690, 2012. Preliminary version in SoCG 2010. `doi:10.1007/s00454-012-9410-z`.

[14] Timothy M. Chan. Three problems about dynamic convex hulls. *Int. J. Comput. Geom. Appl.*, 22(4):341–364, 2012. Preliminary version in SoCG 2011. `doi:10.1142/S0218195912600096`.

[15] Timothy M. Chan. Dynamic generalized closest pair: Revisiting Eppstein's technique. In *Proc. 3rd SIAM Symposium on Simplicity in Algorithms (SOSA)*, pages 33–37, 2020. `doi:10.1137/1.9781611976014.6`.

[16] Timothy M. Chan, Kasper Green Larsen, and Mihai Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th ACM Sympos. Comput. Geom. (SoCG)*, pages 1–10, 2011. `doi:10.1145/1998196.1998198`.

[17] Timothy M. Chan, Mihai Pătraşcu, and Liam Roditty. Dynamic connectivity: Connecting to networks and geometry. *SIAM J. Comput.*, 40(2):333–349, 2011. Preliminary version in FOCS 2008. `doi:10.1137/090751670`.

[18] 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. Preliminary version in SoCG 2015. `doi:10.1007/s00454-016-9784-4`.

[19] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications*. Springer, 3rd edition, 2008.

[20] David P. Dobkin and David G. Kirkpatrick. Determining the separation of preprocessed polyhedra - A unified approach. In *Proc. 17th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 400–413, 1990. `doi:10.1007/BFb0032047`.

[21] David P. Dobkin and Subhash Suri. Maintenance of geometric extrema. *J. ACM*, 38(2):275–298, 1991. `doi:10.1145/103516.103518`.

[22] David Eppstein. Dynamic Euclidean minimum spanning trees and extrema of binary functions. *Discrete Comput. Geom.*, 13:111–122, 1995. `doi:10.1007/BF02574030`.

[23] David Eppstein. Fast hierarchical clustering and other applications of dynamic closest pairs. *ACM Journal of Experimental Algorithmics*, 5:1, 2000. `doi:10.1145/351827.351829`.

[24] Jacob Holm, Eva Rotenberg, and Christian Wulff-Nilsen. Faster fully-dynamic minimum spanning forest. In *Proc. 23rd European Sympos. Algorithms (ESA)*, pages 742–753, 2015. `doi:10.1007/978-3-662-48350-3_62`.

[25] Haim Kaplan, Wolfgang Mulzer, Liam Roditty, Paul Seiferth, and Micha Sharir. Dynamic planar Voronoi diagrams for general distance functions and their algorithmic applications. In *Proc. 28th ACM–SIAM Sympos. Discrete Algorithms (SODA)*, pages 2495–2504, 2017. `doi:10.1137/1.9781611974782.165`.

[26] Vladlen Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. *J. ACM*, 51(5):699–730, 2004. `doi:10.1145/1017460.1017461`.

[27] Jiří Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992. `doi:10.1007/BF02293051`.

[28] Jiří Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2:169–186, 1992. `doi:10.1016/0925-7721(92)90006-E`.

[29] Jiří Matoušek. Linear optimization queries. *J. Algorithms*, 14(3):432–448, 1993. `doi:10.1006/jagm.1993.1023`.

[30] Mark H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes in Computer Science*. Springer, 1983. `doi:10.1007/BFb0014927`.

[31] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23(2):166–204, 1981. `doi:10.1016/0022-0000(81)90012-X`.

[32] Edgar A. Ramos. On range reporting, ray shooting and k-level construction. In *Proc. 15th Sympos. Comput. Geom. (SoCG)*, pages 390–399, 1999. `doi:10.1145/304893.304993`.

[33] Micha Sharir and Shai Zaban. Output-sensitive tools for range searching in higher dimensions. Manuscript, 2013. URL: `http://www.cs.tau.ac.il/~michas/shai.pdf`.

A Proof of Lemma 4.1

In this appendix, we describe how a proof by Chan and Tsakalidis [18, Theorem 5] can be adapted to show Lemma 4.1, as restated below:

Lemma 4.1. *There exist constants b, c, and c' such that the following is true: For a set H of at most n planes in $\mathbb{R}^3$ and a parameter $k \in [1, n]$, given a convex $(-\infty, cbk)$-shallow cutting Γ_{in} with at most $c'n/(bk)$ downward cells, together with their conflict lists, we can construct a convex Γ_{in}-restricted (k, ck)-shallow cutting Γ_{out} with at most $c'n/k$ downward cells, together with their conflict lists, in $O(n + (n/k)\log(n/k))$ deterministic time.*

Proof. As in the previous paper [18], it is more convenient to work with shallow cuttings in vertex form: given a set H of n planes in $\mathbb{R}^3$, a (k, K)-*shallow cutting in vertex form* is a set V of points whose upper hull $\text{UH}(V)$ covers all points in $\mathbb{R}^3$ of level at most k, such that every point in V has level at most K. The *conflict list* of a point refers to the list of all planes of H below the point.

Chan and Tsakalidis [18, Theorem 5] proved the following statement for some constants B, C, and C':

> For a set H of at most n planes in $\mathbb{R}^3$ and a parameter $k \in [1, n]$, given a (Bk, CBk)-shallow cutting V_{in} in vertex form with at most $C'n/(Bk)$ vertices, together with their conflict lists, we can construct a (k, Ck)-shallow cutting V_{out} in vertex form with at most $C'n/k$ vertices, together with their conflict lists, in $O(n + (n/k)\log(n/k))$ deterministic time.

By a close inspection of their proof, we actually get the following stronger statement for *any* choice of constants B, C', and t, where a_0', c_0, and c_0' are absolute constants:

> For a set H of at most n planes in $\mathbb{R}^3$ and a parameter $k \in [1, n]$, given a $(12c_0^2k, CBk)$-shallow cutting V_{in} in vertex form with at most $C'n/(Bk)$ vertices, together with their conflict lists, we can construct a (k, Ck)-shallow cutting V_{out} in vertex form with at most $C''n/k$ vertices, together with their conflict lists, in $O(n + (n/k)\log(n/k))$ deterministic time, where $C = 12c_0^2 + 1$ and $C'' = 2c_0' + \frac{8a_0'c_0'CC'}{3c_0\sqrt{t}}$.

Set $b = B/6$, $c = 3C$, and $c' = C'/18$. To derive Lemma 4.1 from the above statement, we first convert Γ_{in} to vertex form, simply by letting V_{in} to be the vertex set of the convex polyhedron $\mathcal{P}_{\text{in}}$ formed by unioning the cells in Γ_{in}. The polyhedron $\mathcal{P}_{\text{in}}$ has at most $c'n/(bk)$ faces, at most $3c'n/(3bk) = c'n/(bk)$ vertices, and at most $3c'n/(2bk)$ edges. It is helpful to assume that each vertex has degree 3 in $\mathcal{P}_{\text{in}}$; this can be guaranteed by intersecting $\mathcal{P}_{\text{in}}$ with extra planes infinitesimally close to each vertex (the number of new vertices is equal to twice the number of old edges). After this modification, $\mathcal{P}_{\text{in}}$ has $|V_{\text{in}}| \leq 3c'n/(bk) = C'n/(Bk)$ vertices, and at most $2c'n/(bk) = 2C'n/(3Bk)$ faces.

We cannot apply the above result to H directly. Instead, we make $12c_0^2k$ copies of the plane through each face of $\mathcal{P}_{\text{in}}$, and add them to H. The new set $\widehat{H}$ of planes has size $\widehat{n} \leq n + (12c_0^2k)2C'n/(3Bk) \leq 2n$, assuming $B \geq 8c_0^2C'$. Then $\mathcal{P}_{\text{in}} = \text{UH}(V_{\text{in}})$ covers all points of level at most $12c_0^2k$ in $\widehat{H}$. Each point in V_{in} has level at most $cbk + 3(12c_0^2k) \leq CBk$ in $\widehat{H}$, assuming $B \geq 72c_0^2$.

We can now apply the above result to $\widehat{H}$, and obtain a (k, Ck)-shallow cutting V_{out} for $\widehat{H}$ in vertex form. We can set Γ_{out} to be the vertical decomposition of $\text{UH}(V_{\text{out}})$, which has at most

$2|V_{\text{out}}|$ cells. Each cell of Γ_{out} intersects at most $3Ck = ck$ planes of H. Every point covered by Γ_{in} with level at most k in H has level at most k in $\widehat{H}$ and is thus covered by Γ_{out}. Furthermore, $|\Gamma_{\text{out}}| \leq 2C''\widehat{n}/k \leq 4C''n/k = 4(2c_0' + \frac{8a_0'c_0'CC'}{3c_0\sqrt{t}})n/k \leq 12c_0'n/k$ by setting $t = (\frac{8a_0'CC'}{3c_0})^2$. Thus, $|\Gamma_{\text{out}}| \leq C'n/k$ by setting $C' = 36 \cdot 12c_0'$. □