Primal Dividing and Dual Pruning: Output-Sensitive Construction of 4-d Polytopes and 3-d Voronoi Diagrams*

Timothy M. Chan
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4, Canada

Jack Snoeyink
Department of Computer Science
University of British Columbia
Vancouver, B.C. V6T 1Z4, Canada

Chee-Keng Yap
Courant Institute of Mathematical Sciences
New York University
New York, NY 10012, USA

Abstract

In this paper, we give an algorithm for output-sensitive construction of an f-face convex hull of a set of n points in general position in E^4. Our algorithm runs in $O((n+f)\log^2 f)$ time and uses $O(n+f)$ space. This is the first algorithm within a polylogarithmic factor of optimal $O(n \log f + f)$ time over the whole range of f. By a standard lifting map, we obtain output-sensitive algorithms for the Voronoi diagram or Delaunay triangulation in E^3 and for the portion of a Voronoi diagram that is clipped to a convex polytope. Our approach simplifies the "ultimate convex hull algorithm" of Kirkpatrick and Seidel in E^2 and also leads to improved output-sensitive results on constructing convex hulls in E^d for any even constant $d > 4$.

1 Introduction

Geometric structures induced by n points in Euclidean d-dimensional space, such as the convex hull, Voronoi diagram, or Delaunay triangulation, can be of larger size than the point set that defines them. In many practical situations, however, they do not attain the worst-case size. *Output-sensitive algorithms*, which compute structures in time depending on their size, are therefore appealing.

In this paper, we consider computing the convex hull of a set of n points in general position in E^d, concentrating on $d = 4$. The dual problem corresponds to computing the intersection of n halfspaces [14, 21]. In both problems, the output is a (convex) polytope. Because Delaunay triangulations and Voronoi diagrams in E^{d-1} are related to convex hulls and halfspace intersections in E^d by a lifting map [3, 17], our output-sensitive convex hull algorithm gives output-sensitive algorithms for these problems as well.

*The authors have been supported in part by a Killam Graduate Fellowship, an NSERC Graduate Fellowship, an NSERC Research Grant, a B.C. Advanced Systems Institute Fellowship, and NSF grants #CCR-87-03458 and #CCR-94-02464. The first author's present address is: Department of Mathematics and Computer Science, University of Miami, Coral Gables, FL 33124, USA.

In E^d, the convex hull of n points is a polytope with as many as $\Theta(n^{\lfloor d/2 \rfloor})$ faces [14, 30]. For $d = 2$ and $d = 3$, $\Theta(n \log n)$ worst-case time is both necessary and sufficient to compute the facial lattice of the polytope [34]. For any constant dimension d, Chazelle [8] has given a worst-case optimal algorithm running in $O(n \log n + n^{\lfloor d/2 \rfloor})$ time.

For randomly generated point sets [12, 35] and point sets used in practice, however, convex hulls often have fewer faces than the worst-case bound. Thus, a number of convex hull (or halfspace intersection) algorithms have been analyzed not only in terms of n, the size of the input, but also in terms of f, the number of faces of the output polytope. The only known lower bound in terms of n and f is $\Omega(n \log f + f)$ time.

In 1986 Kirkpatrick and Seidel [23] published a paper entitled "The Ultimate Planar Convex Hull Algorithm?" which computes the convex hull of n points in the plane in $O(n \log f)$ time. This algorithm is, therefore, output-sensitive and worst-case optimal. The "ultimate algorithm" is a divide-and-conquer algorithm based on a "marriage before conquest" principle: it computes the merge of two subproblems before it recursively solves the subproblems. Edelsbrunner and Shi [18] applied "marriage before conquest" in 3-d to obtain an $O(n \log^2 f)$ time convex hull algorithm. Using a different approach, Chazelle and Matoušek [10] have reported that derandomizing an algorithm of Clarkson and Shor [11] gives an $O(n \log f)$ time algorithm in 3-d. Recently, Chan [5] has obtained a simple $O(n \log f)$ time method for both 2-d and 3-d convex hulls.

In dimensions higher than three, the fastest output-sensitive algorithms currently known (excluding the results of this paper) are an improvement of the "gift-wrapping" method [22, 38] by Chan [4] and an improvement of Seidel's "shelling" algorithm [37] by Matoušek [28]. The former runs in $O(n \log f + (nf)^{1-1/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n)$ time and the latter runs in $O(n^{2-2/(\lfloor d/2 \rfloor+1)} \log^{O(1)} n + f \log n)$ time.

In this paper we give a convex hull algorithm in four dimensions that runs in $O((n+f)\log^2 f)$ time and uses $O(n+f)$ space. Our basic strategy is divide-and-conquer. In order to obtain an output-sensitive method, the subproblems we solve cannot have asymptotically more faces than the original polytope. Therefore, we make each subproblem compute some restricted portion of the original polytope. For each of the subproblems defined, we show that sufficiently many input points can be removed without changing the subproblem. As in "marriage before conquest" and quicksort-like recursions, the merge step is trivial once we have devised a partitioning scheme for dividing a problem into subproblems.

In the next section we set up notation and point out some preliminary facts about the construction of convex hulls. Section 3 gives a simple $O(n \log f)$ convex hull algorithm in the plane and compares it with Kirkpatrick and Seidel's algorithm [23]. Section 4 details and analyzes the algorithm in four dimensions. We briefly describe the application to Voronoi diagrams in Section 5. An extension of our 4-d algorithm to higher dimensions is given in Section 6.

Note: A preliminary version of this paper, written for the problem of computing halfspace intersections rather than convex hulls, appears in [6]. The 4-d intersection algorithm given there and its 2-d specialization are dualizations of the convex hull algorithms in this paper. We feel that the present version, in the primal setting, is clearer and provides a better understanding of the method.

2 Preliminaries

We first review some standard definitions, introduce key concepts concerning the divide-and-conquer computation of convex hulls, and then describe tools that we need.

2.1 Polytopes

A *polytope* $\mathcal{P} \subseteq E^d$ is the intersection of a finite set of (closed) halfspaces. Suppose that $\mathcal{P}$ has a non-empty interior. If h is a hyperplane that intersects the boundary of $\mathcal{P}$ but not its interior, then $h \cap \mathcal{P}$ is a *face* of $\mathcal{P}$. A face is a *j-face* ($0 \le j < d$) if it has dimension j—that is, if it is contained in some j-flat but not in a $(j-1)$-flat. A $(d-1)$-face is called a *facet*, a $(d-2)$-face is called a *ridge*, a 1-face is called an *edge*, and a 0-face is called a *vertex*. Note that the faces of $\mathcal{P}$, together with the empty set $\emptyset$ and $\mathcal{P}$ itself, form a lattice under inclusion, and the union of the facets of $\mathcal{P}$ is the boundary of $\mathcal{P}$.

2.2 Convex hulls and upper hulls

Suppose that we are given a set $P \subseteq E^d$ of n points, where d is a fixed constant. To avoid degenerate cases, we make a general position assumption on our input. This assumption is not really necessary in our 2-d algorithm but is needed in our 4-d algorithm and its higher-dimensional extension. Specifically, we assume that no $d+1$ points of P lie in a common hyperplane and no d points of P lie in a vertical hyperplane. (Throughout this paper, the terminology "vertical," "above/below," and "upward/downward" are with respect to the last coordinate.) Perturbation techniques [16, 19] can be used to simulate general position, with the possible cost of increasing the output size for degenerate polytopes.

The *convex hull* of P is defined as the smallest convex set containing P, or equivalently, the intersection of all halfspaces containing P. It is well known that the convex hull is a polytope, and our goal is to compute the facial structure of the polytope.

For convenience, we focus our attention only on the upper portion of the convex hull called the *upper hull* (see Figure 1(a)). It consists of faces of the convex hull that have an upward outer normal vector. The upper hull of P can be thought of as the bounded faces of the convex hull of $P \cup \{(0, \ldots, 0, -\infty)\}$. Once we have a method for computing the upper hull of P, we can also compute the lower hull of P in a similar manner by reflection and join the two hulls to form the convex hull of P.

Notation. Let $F(P)$, $R(P)$, and $V(P)$ be the set of all facets, ridges, and vertices (respectively) of the upper hull of P.

2.3 Facets and their duals

To simplify representational issues, we require our algorithm to output only the set $F(P)$ of all facets of the upper hull of P. From this set, we can then generate all faces and build the complete lattice structure of the faces (the Hasse diagram) using a dictionary in $O(|F(P)| \log |F(P)|)$ time; this additional cost will be absorbed in the cost of the algorithm. Our algorithm for computing $F(P)$ is based on divide-and-conquer: to compute all the facets in $F(P)$, we partition $F(P)$ into suitable subsets and recursively compute these subsets of facets.

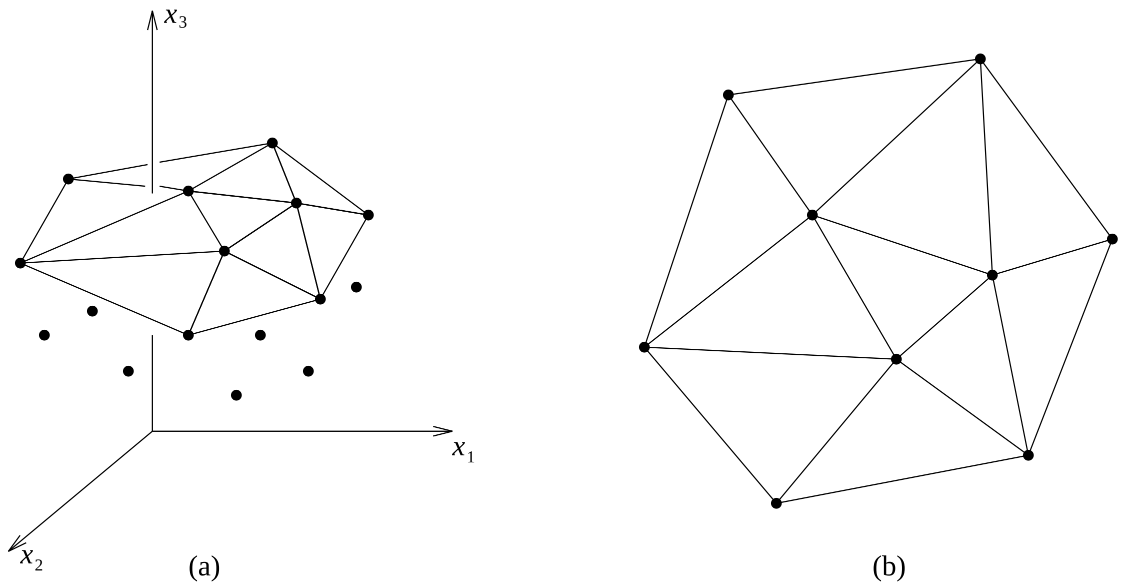

Figure 1: (a) The upper hull of a point set in E^3, and (b) the vertical projection of its facets.

The following provides a simple characterization of $F(P)$. First by the non-degeneracy assumption, $F(P)$ consists only of $(d-1)$-dimensional simplices with vertices all from P. Let f be such a simplex and let $h(f)$ denote the unique hyperplane containing f. Then $f \in F(P)$ iff all points of P lie on or below $h(f)$.

Before we discuss further properties of facets of the upper hull, we first introduce some useful notation used throughout the paper.

Notation. Let $\downarrow$ denote the *vertical projection* operator: $p\downarrow = (x_1, \dots, x_{d-1})$ if $p = (x_1, \dots, x_d)$, and $P\downarrow = \{p\downarrow : p \in P\}$ for any $P \subseteq E^d$. Given $P \subseteq E^d$ and $S \subseteq E^{d-1}$, let $P_{|S} = \{p \in P : p\downarrow \in S\}$ be the *restriction of* P *to* S. Let $\operatorname{int} S$ denote the interior of S and ∂S denote the boundary of S.

Observe that the vertical projection of the facets in $F(P)$ forms a collection of $(d-1)$-dimensional simplices in E^{d-1} that have disjoint interiors, that is, $\operatorname{int}(f\downarrow) \cap \operatorname{int}(f'\downarrow) = \emptyset$ for any two distinct facets $f, f' \in F(P)$. In fact, the vertical projection of all faces of the upper hull forms a *simplicial complex*. For example, if $d = 3$, then $\{f\downarrow : f \in F(P)\}$ forms a triangulation in the plane, as shown in Figure 1(b). Thus, one possible divide-and-conquer approach is to use these vertical projections to partition $F(P)$.

An equally natural approach is to use the vertical projections of the facets' *duals* to partition $F(P)$. For each $f \in F(P)$, we can define a point $f^D \in E^d$ via the standard duality transformation of [14]: if the hyperplane $h(f)$ is given by $\{(x_1, \dots, x_d) : x_d = \xi_1 x_1 + \cdots + \xi_{d-1} x_{d-1} - \xi_d\}$, then we let $f^D = (\xi_1, \dots, \xi_d)$. The projection of the dual $f^D\downarrow = (\xi_1, \dots, \xi_{d-1})$ is geometrically just the gradient of $h(f)$; for example, if $d = 2$, then this is just the slope.

To allow us to speak about the two divide-and-conquer approaches more succinctly, we make the following definitions:

Definition. Given sets $S, \Delta \subseteq E^{d-1}$, let $F_S(P) = \{f \in F(P) : f\!\downarrow\, \subseteq S\}$ be the *primal restriction of* $F(P)$ *to* S and $F^{\Delta}(P) = \{f \in F(P) : f^D\!\downarrow\, \in \Delta\}$ be the *dual restriction of* $F(P)$ *to* Δ.

As we will see, duality [14, 34] plays a cruical role in the design of our algorithms. In the remainder of the section, we discuss specific tools that our algorithms use.

2.4 Cuttings for divide and conquer

The work of many researchers, notably Matoušek [25, 26] and Chazelle [7], has developed $(1/r)$-cuttings for divide-and-conquer algorithms for hyperplanes. A *cutting* in E^d is a covering of E^d with closed (possibly unbounded) simplices with disjoint interiors; the *size* of the cutting is the number of simplices. For a set H of n hyperplanes, a cutting Ξ is a $(1/r)$-cutting if any simplex of Ξ intersects at most n/r hyperplanes of H. Chazelle and Friedman [9] showed that $(1/r)$-cuttings of size $O(r^d)$ exist. Several theorems have been proved about their deterministic construction—we use a relatively simple one.

Theorem 2.1 ([24, Theorem 6.1]) *Given n hyperplanes in a fixed dimension d, a $(1/r)$-cutting of size $O(r^d)$ can be computed in $O(nr^d)$ time.*

We need this theorem only for the special case when r is just a constant, say 2.

2.5 Queries on polytopes

A number of researchers [28, 29] have devised data structures for answering queries on a polytope $\mathcal{P} \subseteq E^d$ defined as an intersection of n halfspaces. Two common types of queries that have been studied are *ray shooting* (identify the point where a query ray ρ intersects $\mathcal{P}$ assuming that the origin of the ray lies in $\mathcal{P}$), and *linear programming* (find the point $\xi \in \mathcal{P}$ that maximizes $a \cdot \xi$ for a query vector $a \in E^d$). It turns out that one can answer these queries efficiently without having to construct the polytope $\mathcal{P}$ explicitly.

The following theorem is a direct consequence of applying the ray shooting structure of Matoušek and Schwarzkopf [29] and the linear programming structure of Matoušek [28] and choosing appropriate tradeoffs between preprocessing and query time (e.g., see [4]).

Theorem 2.2 *An (online) sequence of q ray shooting and linear programming queries on a polytope defined by n halfspaces in E^d can be performed in $O((n + q + (nq)^{1-1/(\lfloor d/2 \rfloor+1)}) \log^{O(1)} n)$ time.*

By dualizing points into halfspaces, we can use this result to answer queries on the convex hull of an n-point set. For example, given a point $q \in E^{d-1}$, we can find a facet $f \in F(P)$ with $q \in f\!\downarrow$ (if one exists) by performing a linear programming query on the dual polytope; given a facet $f \in F(P)$ and a ridge $r \in R(P)$ incident on f, we can find the other facet $f' \in F(P)$ that r is incident on (if it exists) by performing a ray shooting query in dual space. Theorem 2.2 is used in an extension of our convex hull algorithm in higher dimensions.

2.6 Counting the cost

To evaluate the cost of our recursive convex hull algorithm, we prove a general lemma concerning recursion trees. Let T be a rooted tree in which each node ν is assigned a cost $c(\nu) \in (0, \infty)$. We say that the cost function c is *α-fading* for a constant $\alpha \in (0,1)$ if $c(\mu) \leq \alpha\, c(\nu)$ for every node μ and its parent ν. As part of the analysis of their 3-d output-sensitive convex hull algorithm, Edelsbrunner and Shi [18, Lemma 3.1] proved that the total cost in such a tree is asymptotically bounded by the per-level cost times the logarithm of the number of nodes. Their proof uses a path compression operation that transforms T into a balanced tree. We give a simple, short proof of their result that avoids path compression altogether; we then improve the bound to depend on the number of leaves rather than the number of nodes.

Lemma 2.3 *In a recursion tree T with m nodes and ℓ leaves and an α-fading cost function c, if the sum of the costs at each level is bounded by C, then the sum of the costs of all nodes in T is (i) less than $C(\log_{1/\alpha} m + 2)$ and (ii) less than $C(\log_{1/\alpha} \ell + 1 + 1/(1-\alpha))$.*

Proof: Number the levels of the tree 0, 1, 2, ... with the root at level zero. Let $k = \left\lfloor \log_{1/\alpha} m \right\rfloor$. The sum of the costs at levels $0, 1, \ldots, k$ is bounded by $C(k+1) \leq C(\log_{1/\alpha} m + 1)$. Furthermore, by the α-fading property, each node on a level greater than k has cost bounded by $C\alpha^{k+1} < C/m$; hence, the sum of the costs at level $k+1, k+2, \ldots$ is bounded by C. Part (i) follows.

To prove part (ii), we choose $k = \left\lfloor \log_{1/\alpha} \ell \right\rfloor$ instead. As before, the sum of the costs at levels $0, 1, \ldots, k$ is bounded by $C(k+1) \leq C(\log_{1/\alpha} \ell + 1)$. Thus, we just have to account for the costs of nodes at levels greater than k. Note that each node belongs to some root-to-leaf path in T. By the α-fading property, the sum of the costs at levels $k+1, k+2, \ldots$ along such a path is bounded by

$$C\alpha^{k+1} + C\alpha^{k+2} + \cdots \;=\; \frac{C\alpha^{k+1}}{1-\alpha} \;<\; \frac{C}{(1-\alpha)\ell}.$$

Since there are ℓ root-to-leaf paths in total, the sum of the costs at levels $k+1, k+2, \ldots$ is bounded by $C/(1-\alpha)$. Part (ii) follows. □

3 A simplified "Ultimate Planar Convex Hull Algorithm"

Our 4-d convex hull algorithm grows out of the following simple convex hull algorithm in the plane. This algorithm is in turn based on the "prune-and-search" linear programming algorithm of Dyer [13] and Megiddo [31]. Rather than using pruning to search, we use pruning for divide-and-conquer.

Given an n-point set $P \subseteq E^2$, we want to compute the upper hull of P. We first pair the points of P arbitrarily and calculate the slope of the line through each pair. We then find the median slope m and compute the upper-hull vertex p_m that has a supporting line of slope m; this vertex can be computed by taking the maximum along a certain projection of P. The x-coordinate of p_m is then used to divide P into two parts: P_ℓ, which contains p_m and all points to its left, and P_r, which contains p_m and all points to its right.

Now, if a pair has slope less than m, then the right point in the pair cannot participate in the upper hull of P_ℓ and thus can be pruned from P_ℓ. Similarly, if a pair has slope greater than m, then

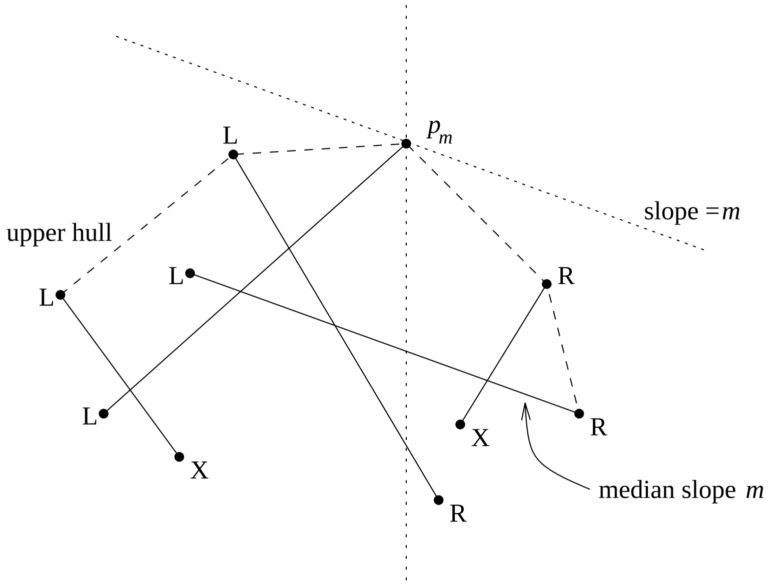

Figure 2: Pairing and pruning points in the plane. Points marked L belong to P_ℓ, points marked R belong to P_r, and points marked X belong to neither sets.

its left point in the pair cannot participate in the upper hull of P_r and can be pruned from P_r. Since half (i.e., $n/4$) of the pairs have slope less than the median m and half have slope greater than m, pruning ensures that P_ℓ and P_r each contain at most $3n/4$ points. We then recursively compute the upper hull of P_ℓ and P_r. See Figure 2 for an example.

The pseudocode of the algorithm is given below. For convenience, we assume that the leftmost and rightmost points p_ℓ and p_r of P have been identified and we let n be the cardinality of the set $P^\bullet = P - \{p_\ell, p_r\}$ instead. In the interest of practical efficiency, line 1 has been added to the algorithm; it does not affect asymptotic worst-case performance.

Algorithm $\texttt{Hull2}(P^\bullet, p_\ell, p_r)$

[Given n-point set $P^\bullet \subseteq E^2$ and points $p_\ell, p_r \in E^2$ such that $x[p_\ell] < x[p] < x[p_r]$ for all $p \in P^\bullet$, return the set of edges of the upper hull of $P = P^\bullet \cup \{p_\ell, p_r\}$.]

1. discard points from $P^\bullet$ that lie below $\overline{p_\ell p_r}$
2. if $P^\bullet = \emptyset$ then return $\{\overline{p_\ell p_r}\}$
   if $P^\bullet = \{p\}$ then return $\{\overline{p_\ell p}, \overline{p p_r}\}$
3. arbitrarily choose $\lfloor n/2 \rfloor$ disjoint pairs $\{\{s_1, t_1\}, \ldots, \{s_{\lfloor n/2 \rfloor}, t_{\lfloor n/2 \rfloor}\}\}$ from $P^\bullet$
   and order each pair so that $x[s_i] < x[t_i]$
4. let $m_i = (y[t_i] - y[s_i]) \,/\, (x[t_i] - x[s_i]),\ \ i = 1, \ldots, \lfloor n/2 \rfloor$
   and $m =$ median of $\langle m_1, \ldots, m_{\lfloor n/2 \rfloor} \rangle$
5. let $p_m =$ point in P that maximizes $y[p_m] - m \cdot x[p_m]$
6. let $P_\ell^\bullet = \{p \in P^\bullet : x[p] < x[p_m]\} - \{t_i : m_i \le m\}$
   $P_r^\bullet = \{p \in P^\bullet : x[p] > x[p_m]\} - \{s_i : m_i \ge m\}$
7. if $p_m = p_r$ then return $\texttt{Hull2}(P_\ell^\bullet, p_\ell, p_r)$
   if $p_m = p_\ell$ then return $\texttt{Hull2}(P_r^\bullet, p_\ell, p_r)$
   otherwise return $\texttt{Hull2}(P_\ell^\bullet, p_\ell, p_m) \ \cup\ \texttt{Hull2}(P_r^\bullet, p_m, p_r)$

As median-finding (line 4) can be done in linear time, the running time of the algorithm satisfies the following recurrence (where n is the number of input points excluding p_ℓ and p_r, f is the number of output edges, and c is some constant):

$$T(n,f) \le \begin{cases} c & \text{if } n \le 1 \\ T(n_\ell, f) + cn & \text{if } n \ge 2 \text{ and } f_r = 0 \\ T(n_r, f) + cn & \text{if } n \ge 2 \text{ and } f_\ell = 0 \\ T(n_\ell, f_\ell) + T(n_r, f_r) + cn & \text{if } n \ge 2 \text{ and } f_\ell, f_r \ge 1 \end{cases}$$

for some $0 \le n_\ell, n_r \le \lceil 3n/4 \rceil$ and $f_\ell, f_r \ge 0$ with $n_\ell + n_r < n$ and $f_\ell + f_r = f$.

Using the concavity of the logarithm, one can then prove that $T(n,f) = O(n \log f)$ by induction. We note that a simpler proof follows from using Lemma 2.3 to analyze the recursion tree. It is clear that the sum of the costs at each level of the tree is bounded by cn and that the cost function satisfies the $(3/4)$-fading property. Since the number of leaves is at most f (as a new edge is discovered at every leaf), Lemma 2.3(ii) immediately implies that the total cost of the algorithm is bounded by $cn \log_{4/3} f + O(n)$. We have thus shown:

Theorem 3.1 *Algorithm* $\texttt{Hull2}()$ *computes the f-edge upper hull of an n-point set $P \subseteq E^2$ in $O(n \log f)$ time and $O(n)$ space.*

Remarks:

1. $\texttt{Hull2}()$ can be viewed as a simplification of the recursive algorithm of Kirkpatrick and Seidel [23]. Their algorithm uses prune-and-search to find a hull edge at each node of the recursion tree. Our algorithm finds only a hull vertex at each node and thus requires a simpler partitioning strategy: "pruning" is done in the same manner but "searching" is bypassed entirely. In terms of running time, our algorithm is faster by a constant factor: if finding the median of n numbers takes bn time, then in the worst case, Kirkpatrick and Seidel's algorithm spends $3bn \log_2 f + O(n)$ time

and our algorithm spends $\frac{1}{2}bn\log_{4/3} f + O(n) \approx 1.2bn\log_2 f + O(n)$ time in median-finding (which is the most costly operation in both algorithms).

2. One can also view `Hull2()` as a variant of QuickHull [34], since it recursively uses an extreme vertex to divide a convex hull into two. But as pruning is not done in QuickHull, its worst-case complexity can be $\Theta(nf)$. We learned recently that Wenger [39] has proposed a randomized version of QuickHull that performs pruning. His algorithm, with an $O(n\log f)$ expected running time, is similar to ours, except that finding the median slope in line 4 is replaced by randomly selecting a slope. Chan [5] has recently reported a different deterministic algorithm that can also compute two-dimensional convex hulls in $O(n\log f)$ time. Chan's algorithm is likely to be faster in terms of worst-case complexity as it does not need median-finding.

4 An output-sensitive algorithm in E^4

We now extend algorithm `Hull2()` to four dimensions. A high-level description of the extension is as follows. Recall that in line 4 of algorithm `Hull2()`, the median of a set of $n/2$ numbers is computed. Since the median can be thought of as a one-dimensional (1/2)-cutting from Section 2.4, we extend this step to d dimensions by computing the (1/2)-cutting of a set of $n/2$ hyperplanes in E^{d-1}. In line 5, a vertex p_m, used for dividing the hull, is computed by taking the maximum of a set of numbers formed by projecting the input points along a direction of slope m. Of course, the maximum of a set of numbers can be interpreted as the upper hull of a one-dimensional point set. In d dimensions, p_m then becomes a collection of ridges computed by projecting the input points along certain directions and taking the upper hulls of the resulting $(d-1)$-dimensional point sets. For $d = 4$, these upper hulls are 3-dimensional and are therefore of linear size.

In the 2-d algorithm, p_m divides the upper hull of P into two parts: the portion of the hull to the left of p_m and the portion to the right of p_m. Observe that the left hull is also the portion with slope less than m, and similarly the right hull is the portion with slope greater than m. We have thus used p_m to partition the upper hull in two ways: (i) by x-coordinate and (ii) by slope. The restriction of the upper hull with x-coordinate inside a given interval is just primal restriction, in the terminology of Section 2.3, and the restriction of the upper hull with slope within a given interval is just dual restriction. In our extension to 4-d, we adopt the same strategy of using both primal and dual restrictions to partition the upper hull.

In the planar case, dividing the point set by x-coordinate ensures that the two subproblems do not share any input points except for the vertex p_m, and pruning by slope ensures that each of the two subproblems has at most 3/4 of the input points. In the same manner for E^4, dividing by primal restrictions controls the sum of the sizes of the subproblems and pruning by dual restrictions guarantees that no subproblem receives more than a fixed fraction of the input. Subproblems can now share more than one input point, but we argue that the number of points shared is proportional to the size of the output. The analysis then follows from an application of Lemma 2.3: primal dividing bounds the per-level cost of the recursion tree and dual pruning ensures the α-fading property.

4.1 Primal dividing

Our convex hull algorithm computes the primal restrictions of $F(P)$ to certain regions in E^{d-1} recursively. The regions are not arbitrary but are of a special form that we call *simple regions*.

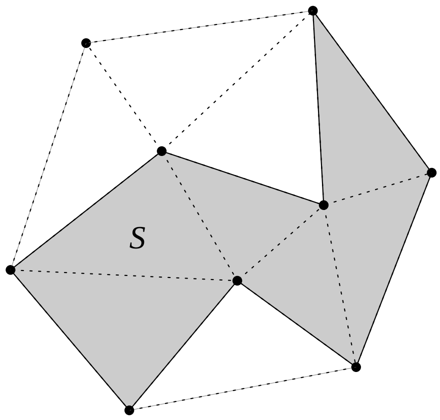

Figure 3: A simple region S of the point set in Figure 1.

Definition. A set $S \subseteq E^{d-1}$ is a *simple region* of P if it is the vertical projection of a union of facets in $F(P)$.

Figure 3 shows an example of a simple region. A simple region S may be disconnected. There may even exist a non-empty open $(d-1)$-ball centered on ∂S, of an arbitrarily small radius, whose intersection with $\operatorname{int} S$ is not homeomorphic to any $(d-1)$-ball. This intersection cannot be empty however, as S is a union of full-dimensional simplices; in particular, this rules out "spikes," e.g., $(d-2)$-simplices that are attached to the boundary.

The following lemma lists some useful properties concerning simple regions and primal restrictions. Part (a) is an identity that follows from definition and is important for proving other parts of the lemma. Parts (b) and (c) discuss when points can be removed without changing the primal restriction. Parts (d) and (e) provide bounds on the number of vertices and ridges restricted to a simple region. Finally, (f) and (g) describe properties of the boundary of a simple region.

Lemma 4.1 *Let S be a simple region of P. The following statements are true:*

(a) $\bigcup_{f \in F_S(P)} f{\downarrow} = S$. *(The projection of the facets of the primal restriction to S covers the region S.)*

(b) If $Q \subseteq P$ contains all vertices of the facets in $F_S(P)$, then S is a simple region of Q and $F_S(P) = F_S(Q)$. (Points that do not contribute to facets in $F_S(P)$ can be removed from P without affecting $F_S(P)$.)

(c) S is a simple region of the restricted point set $P_{|S}$ and the restricted facets $F_S(P_{|S}) = F_S(P)$.

(d) $|V(P_{|S})| \leq d\,|F_S(P)|$. *(The number of facets in the primal restriction gives a bound for the number of vertices.)*

(e) $|\{r \in R(P) : r{\downarrow} \subseteq S\}| \leq d\,|F_S(P)|$. *(The number of facets in the primal restriction gives a bound for the number of ridges.)*

(f) ∂S *is the vertical projection of a union of ridges in* $R(P)$*. Thus, we can represent* ∂S *as a set of at most* $d\,|F_S(P)|$ *ridges.*

(g) $P_{|\partial S} = \{v : v$ *is a vertex of some ridge* r *in* $\partial S\}$*, and* $|P_{|\partial S}| \le d\,|F_S(P)|$*. (The number of vertices in the boundary* ∂S *is bounded.)*

Proof: Recall that $\{\mathrm{int}\,(f{\downarrow}) : f \in F(P)\}$ are disjoint. Then (a) is immediate from the definition of the primal restriction $F_S(P)$.

To prove (b), first note that $F_S(P) \subseteq F_S(Q)$ follows directly from the hypothesis. This implies that S is a simple region of Q. Now, (a) says that $\bigcup_{f\in F_S(P)} f{\downarrow} = S = \bigcup_{f\in F_S(Q)} f{\downarrow}$. We therefore must have equality: $F_S(P) = F_S(Q)$.

Statement (c) is a direct consequence of (b).

To prove (d), let p be a point in $V(P_{|S})$. Since the projection $p{\downarrow} \in S$, by (a) we have $p{\downarrow} \in f{\downarrow}$ for some facet $f \in F_S(P)$. Since $p \in V(P_{|S})$, p must be a vertex of f. Then (d) follows as each facet has d vertices incident on it (by the non-degeneracy assumption).

To prove (e), observe that each ridge r with $r{\downarrow} \subseteq S$ is incident on some facet in $F_S(P)$, by (a). Then (e) follows as each facet has d ridges incident on it.

The first part of (f) is immediate from the definition of a simple region. The cardinality bound is just a consequence of (e).

The first part of (g) follows from (f) and the non-degeneracy assumption. In particular, this implies that $P_{|\partial S} \subseteq V(P)$ and consequently, $P_{|\partial S} \subseteq V(P_{|S})$. So the second part follows from (d). $\square$

To compute the primal restriction $F_S(P)$ for a simple region S of P, our divide-and-conquer algorithm first subdivides S into smaller simple regions $\{S_i\}$ with disjoint interiors and then recursively computes $F_{S_i}(P)$ for each of the S_i's. In computing $F_{S_i}(P)$ we may consider only those input points that belong to $P_{|S_i} = P_{|\,\mathrm{int}\,S_i} \cup P_{|\partial S_i}$ by Lemma 4.1(c). Since $\mathrm{int}\,S_i$ are disjoint, the points shared between subproblems are points restricted to the boundary of the S_i's and we can bound the size of these boundaries in terms of the number of output facets by Lemma 4.1(f,g).

Note that when $d = 2$, the boundary of a connected simple region consists of just two points. In higher dimensions, the boundary becomes more complex and its manipulation demands more care.

4.2 Dual pruning

To find a good strategy for subdividing a simple region, we switch to dual space. We show how to partition E^{d-1} into a constant number of simplices such that in computing the dual restriction of $F(P)$ to each of the simplices, a fraction of the points of P can be pruned.

Lemma 4.2 *In* $O(|P|)$ *time, one can find closed simplices* $\Delta_1, \ldots, \Delta_k \subseteq E^{d-1}$ *and* $P_1, \ldots, P_k \subseteq P$ *such that (i)* $\bigcup_{i=1}^{k} \Delta_i = E^{d-1}$ *and* $\{\mathrm{int}\,\Delta_i\}$ *are disjoint, (ii)* $F^{\Delta_i}(P) = F^{\Delta_i}(P_i)$*, and (iii)* $|P_i| \le \alpha\,|P|$*, for all* $i = 1, \ldots, k$*. Here,* k *and* $0 < \alpha < 1$ *are both constants depending only on* d *(assuming that* $|P|$ *exceeds a certain constant).*

Proof: A proof of this lemma in the dual setting can be found in Edelsbrunner's exposition [14] of Megiddo's linear programming algorithm [31]. The algorithm is based on the prune-and-search paradigm, and this lemma represents its "prune step." In Megiddo's original approach, the constant k is quite large and α is very close to 1. We observe an alternative solution using results on cuttings.

Let $|P| = n$. First, form the set H of dual hyperplanes by mapping each point $p = (p_1, \dots, p_d)$ of P to the hyperplane $\{(\xi_1, \dots, \xi_d) : \xi_d = p_1\xi_1 + \cdots + p_{d-1}\xi_{d-1} - p_d\}$. Then each facet f of the upper hull of P corresponds to a vertex of the lower envelope of H. (The *lower envelope* of H consists of faces of the polytope $\mathcal{P} = \{\xi \in E^d : \xi \text{ is below every hyperplane of } H\}$.) Furthermore, if $\Delta \subseteq E^{d-1}$, then a facet f in the dual restriction $F^{\Delta}(P)$ corresponds to a vertex of the lower envelope that has vertical projection in Δ.

Arbitrarily pair the n hyperplanes in H, compute the intersection of each pair, and vertically project these intersections. This gives us $n/2$ hyperplanes in E^{d-1}. Compute a constant-sized $(1/2)$-cutting $\{\Delta_i\}$ of these $(d-1)$-dimensional hyperplanes by Theorem 2.1. Consider a simplex Δ_i from the cutting. At least half of the $n/2$ pairs have an intersection whose vertical projection lies completely outside Δ_i. For such a pair, one of the two hyperplanes cannot participate in the restriction of the lower envelope of H to Δ_i and is thus redundant. Therefore, in computing the dual restriction $F^{\Delta_i}(P)$, at least $n/4$ of the points can be pruned. This proves the lemma with $\alpha = 3/4$. □

Remark: In practice, one would compute the cutting using a random sampling approach [11, 32] rather than using Theorem 2.1. Furthermore, a different notion of cuttings known as *shallow cuttings* [27] can be used to prove Lemma 4.2; this alternative approach may reduce the value of the constant k in higher dimensions.

4.3 Converting from dual to primal

In this subsection we show how to convert the partitioning $\{\Delta_i\}$ of the dual space obtained from Lemma 4.2 into a partitioning in the primal space. Specifically, we show that for any simplex $\Delta \subseteq E^{d-1}$, we can define a region $S_\Delta = S_\Delta(P) \subseteq E^{d-1}$ such that the primal restriction of $F(P)$ to S_Δ is the same as the dual restriction of $F(P)$ to Δ (i.e., $F_{S_\Delta}(P) = F^{\Delta}(P)$).

We start with the case when Δ is just a halfspace. Without loss of generality, assume that Δ has the form $\{(\xi_1, \dots, \xi_{d-1}) : \xi_1 + m_2\xi_2 + \cdots + m_{d-1}\xi_{d-1} \le m_d\}$. Define a projection $\pi_\Delta : E^d \to E^{d-1}$ that sends a point $(x_1, \dots, x_d)$ to $(x_2 - m_2x_1,\ \dots,\ x_d - m_dx_1)$. Consider the upper hull of the $(d-1)$-dimensional point set $\pi_\Delta(P)$: its facets are projection of ridges in the upper hull of P, that is, $F(\pi_\Delta(P)) \subseteq \{\pi_\Delta(r) : r \in R(P)\}$. Let "boundary" B_Δ be the union of all $r{\downarrow}$ with $\pi_\Delta(r) \in F(\pi_\Delta(P))$. Then B_Δ is monotone in the following sense: a line of the form $\{(x_1 + t,\ x_2 + m_2t,\ \dots,\ x_{d-1} + m_{d-1}t) : t \text{ real}\}$ can intersect B_Δ at most once. We define S_Δ to be the region "right of" B_Δ: $(x_1, \dots, x_{d-1}) \in S_\Delta$ iff $(x_1 + t,\ x_2 + m_2t,\ \dots,\ x_{d-1} + m_{d-1}t) \in B_\Delta$ for some $t \le 0$. Figure 4(a) shows an example with $m_2 = \cdots = m_{d-1} = 0$.

The following lemma can now be established for a halfspace Δ by verifying definitions.

Lemma 4.3 $F_{S_\Delta}(P) = F^{\Delta}(P)$.

Remark: In the dual setting, as described in the proof of Lemma 4.2, the $(d-1)$-dimensional upper hull of the projected point set $\pi_\Delta(P)$ corresponds to the $(d-1)$-dimensional intersection of the lower envelope of H with $\partial\Delta$. Thus, ridges appearing in B_Δ correspond to edges of the lower envelope of H that intersect $\partial\Delta$.

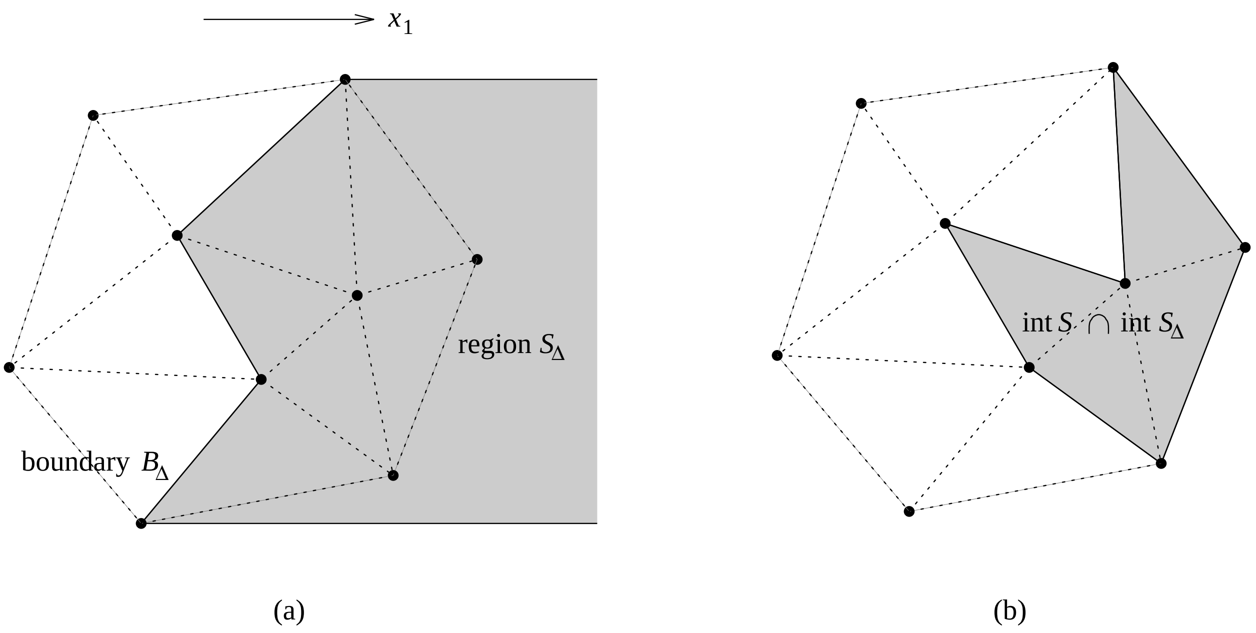

Figure 4: (a) The region S_Δ and (b) the intersection of its interior with the interior of the simple region S from Figure 3.

We now extend the definition of S_Δ to the case when Δ is a simplex rather than a halfspace: Since Δ is a simplex, write Δ as an intersection of a constant number of halfspaces $\{\delta_j\}$. Then define S_Δ to be a region with interior $\bigcap_j \operatorname{int} S_{\delta_j}$. It is not difficult to see that Lemma 4.3 holds for simplices Δ as well.

4.4 Specializing for $d = 4$

We now show that the region S_Δ as defined in the previous subsection satisfies some nice computational properties if $d = 4$. We first consider the case in which Δ is a halfspace.

For $d = 4$, the projected point set $\pi_\Delta(P)$ is 3-dimensional. We can compute the facets of the upper hull $F(\pi_\Delta(P))$, and thus, the boundary B_Δ, in $O(|P| \log |V(P)|)$ time by either Chazelle and Matoušek's algorithm [10] or Chan's algorithm [5]. This permits computations involving the region S_Δ to be done efficiently, such as deciding if a point lies in the interior of S_Δ.

Lemma 4.4 *Suppose that $d = 4$. Then the restricted point set $P_{|\operatorname{int} S_\Delta}$ can be computed in $O(|P| \log |V(P)|)$ time using $O(|P|)$ space.*

Proof: Compute the facets of the 3-dimensional upper hull $F(\pi_\Delta(P))$ and store $\{\pi_\Delta(r)\!\downarrow\, : \pi_\Delta(r) \in F(\pi_\Delta(P))\}$, which is a set of $O(|V(P)|)$ triangles in E^2 with disjoint interiors, in a planar point location structure [15, 33, 36]; this takes $O(|P| \log |V(P)|)$ time. For each $p \in P$, we can then test whether $p\!\downarrow\, \in \operatorname{int} S_\Delta$ in logarithmic time by finding a facet $\pi_\Delta(r)$ of $F(\pi_\Delta(P))$ with $\pi_\Delta(p)\!\downarrow\, \in \pi_\Delta(r)\!\downarrow$ and then determining which side of $r\!\downarrow$ the point $p\!\downarrow$ lies on. □

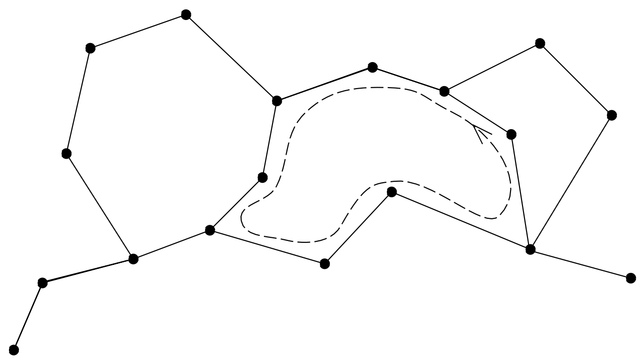

Figure 5: Tracing a boundary component ($d = 3$).

Another operation on the region S_Δ that we need is that of intersecting S_Δ with a simple region (see Figure 4(b)). To ensure that the resulting region is simple, we intersect their interiors only. We represent a simple region S by its boundary, which is a set of $O(|F_S(P)|)$ ridges by Lemma 4.1(f). We assume that each ridge r of ∂S is given an orientation to indicate which side of $r\!\downarrow$ the region S lies on.

Lemma 4.5 *Suppose that $d = 4$. Given the boundary ∂S for a simple region S of P, one can construct the boundary $\partial S'$ for a new simple region S' of P with $\operatorname{int} S' = \operatorname{int} S \cap \operatorname{int} S_\Delta$ in $O((|P| + |F_S(P)|) \log |V(P)|)$ time using $O(|P| + |F_S(P)|)$ space.*

Proof: Call a subset of S a *subregion* if it is the closure of a connected component of $E^{d-1} - (\partial S \cup B_\Delta)$. A subregion is a simple region, so we can define the new simple region S' to be the union of all subregions contained in S_Δ. The boundary $\partial S'$ is made up of *boundary components*, which are connected components of the boundary of subregions. To decide whether a given boundary component B contributes to $\partial S'$, take a point q near B but inside the region bounded by B (this requires an examination of the orientation of a ridge in B), and then test if $q \in \operatorname{int} S_\Delta$ using the point location method from the previous lemma. Therefore, to compute $\partial S'$, it suffices to produce all the boundary components. This is done using depth-first search as follows.

We first record the ridges of the boundaries ∂S and B_Δ in a dictionary; as the $(d-1)$-dimensional upper hull $F(\pi_\Delta(P))$ and the boundary B_Δ can be computed in $O(|P| \log |V(P)|)$ time for $d = 4$, this takes $O((|P| + |F_S(P)|) \log |V(P)|)$ time. We make two copies of a ridge to represent the two "sides" of a ridge and assign different orientations to them. We then generate all the $(d-3)$-subfaces of these ridges, and for each such $(d-3)$-face σ, we create the list of ridges incident to σ in sorted order and store σ in a dictionary for $(d-3)$-faces. The ordering of these ridges is based on the angles made by their vertical projections with a fixed hyperplane through $\sigma\!\downarrow$ in E^{d-1}.

Then, given an (oriented) ridge in a boundary component B, we can identify its $d-1$ adjacent ridges in B in constant time by following pointers. (Here, two oriented ridges r_1 and r_2 are *adjacent* in B if there is a common $(d-3)$-subface σ incident on both ridges and there is no other ridge r' in B that σ is incident on, such that $r'\!\downarrow$ lies within the angle range defined by $r_1\!\downarrow$ and $r_2\!\downarrow$ around $\sigma\!\downarrow$.) Using a depth-first search to visit the adjacent ridges recursively, we can then trace all ridges that belong to the same boundary component B, as indicated in Figure 5. All boundary components can then be generated by ensuring that all ridges in ∂S are visited. The time required by the depth-first search is proportional to the number of ridges in ∂S and B_Δ and is therefore only $O(|F_S(P)|+|V(P)|)$. □

It remains to extend the above lemmas to the case when Δ is a simplex rather than a halfspace. Recall that we have defined S_Δ such that $\operatorname{int} S_\Delta = \bigcap_j \operatorname{int} S_{\delta_j}$, if Δ is written as an intersection of a constant number of halfspaces $\{\delta_j\}$. By applying Lemmas 4.4 and 4.5 to each halfspace δ_j individually, we see that the lemmas are also true for the simplex Δ.

4.5 The algorithm

We now have all the pieces needed for an output-sensitive convex hull algorithm in E^4. Let `Dual-Partition`(P) represent a dual partitioning $\{(P_i, \Delta_i)\}_{i=1}^k$ obtained from Lemma 4.2. Let `Restrict-Interior`(P, Δ) represent the restricted point set $P_{|\operatorname{int} S_\Delta}$ as computed by Lemma 4.4 and let `Restrict-Boundary`(P, B, Δ) be the boundary of the simple region S' returned in Lemma 4.5 for $B = \partial S$. The following provides an outline of our recursive algorithm.

Algorithm `Hull4`$(P^\bullet, B)$
[Given $P^\bullet = P_{|\operatorname{int} S}$ and $B = \partial S$ for a simple region S of a point set $P \subseteq E^4$ where $B \neq \emptyset$ is represented as a set of (oriented) ridges, return the set of facets $F_S(P)$.]
1. $P \leftarrow P^\bullet \cup \{v : v \text{ is a vertex of some ridge } r \text{ in } B\}$
2. if $|P| \leq n_0$ for a constant n_0 then return $F_S(P)$ in constant time
3. $\{(P_i, \Delta_i)\}_{i=1}^k \leftarrow$ `Dual-Partition`(P) by computing a (1/2)-cutting (Lemma 4.2)
4. for $i = 1, \ldots, k$ do
5. $\quad P_i^\bullet \leftarrow P_i \cap P^\bullet \cap$ `Restrict-Interior`(P, Δ_i) by computing a 3-d upper hull and performing 2-d point location (Lemma 4.4)
6. $\quad B_i \leftarrow$ `Restrict-Boundary`(P, B, Δ_i) by computing a 3-d upper hull and performing depth-first search on the boundary ridges (Lemma 4.5)
7. return $\bigcup \{$`Hull4`$(P_i^\bullet, B_i) : B_i \neq \emptyset\}$

We first argue that the algorithm indeed computes the primal restriction $F_S(P)$. In the first line of the algorithm, we reset P to the point set $P_{|\operatorname{int} S} \cup P_{|\partial S} = P_{|S}$, according to Lemma 4.1(g); the justification is provided by Lemma 4.1(c): $F_S(P) = F_S(P_{|S})$. Line 2 provides the base case. Line 3 gives us a constant number of simplices $\{\Delta_i\}$ with disjoint interiors, covering E^{d-1}; for each Δ_i, we are also given a subset P_i of P, of cardinality at most $\alpha\,|P|$, such that $F^{\Delta_i}(P) = F^{\Delta_i}(P_i)$.

Let S_i denote the simple region with interior $\operatorname{int} S \cap \operatorname{int} S_{\Delta_i}$. Since $F_{S_{\Delta_i}}(P) = F^{\Delta_i}(P)$ by Lemma 4.3, we know that the S_i's have disjoint interiors and that their union is S. Furthermore, as $F_{S_i}(P) \subseteq F_{S_{\Delta_i}}(P) = F^{\Delta_i}(P) = F^{\Delta_i}(P_i)$, all facets in $F_{S_i}(P)$ have vertices from P_i, which implies that $F_{S_i}(P) = F_{S_i}(P_i)$ by Lemma 4.1(b).

In line 5 we set $P_i^\bullet = P_i \cap P^\bullet \cap P_{|\operatorname{int} S_{\Delta_i}} = P_{i\,|\operatorname{int} S_i}$ and in line 6 we let B_i be the boundary of S_i. Then line 7 returns $\bigcup_i F_{S_i}(P_i) = \bigcup_i F_{S_i}(P) = F_S(P)$, as claimed.

Having argued that `Hull4`() correctly computes the primal restriction $F_S(P)$, we can use the algorithm to compute the set $F(P)$ of all facets of the upper hull. The initial simple region S_0

we use is just the convex hull of $P{\downarrow}$, which can be computed using the 3-dimensional algorithm of Chazelle and Matoušek [10] or Chan [5]. Thus, by letting $P^\bullet = \{p \in P : p{\downarrow} \text{ is not a vertex of } S_0\}$ and $B = \partial S_0$, a call to `Hull4`$(P^\bullet, B)$ then returns $F(P)$, as desired.

4.6 Analysis

We now analyze the running time of the algorithm. We do so by counting the cost of the recursion tree produced by the calls to `Hull4`(). Let n be the number of input points and f be the number of facets of the upper hull. Let P_ν and S_ν denote the input point set and the simple region associated with a node ν of the recursion tree. Let $n_\nu = |P_{\nu\,|\,\mathrm{int}\,S_\nu}|$ and $f_\nu = |F_{S_\nu}(P_\nu)|$.

By Lemmas 4.2, 4.4, and 4.5, the non-recursive part of the algorithm (lines 1–6) requires $O((|P_\nu| + |F_{S_\nu}(P_\nu)|)\log|V(P_\nu)|) = O((|P_\nu| + f_\nu)\log f_\nu)$ time at node ν, since $|V(P_\nu)| = |V(P_{\nu\,|S_\nu})| \le d\,|F_{S_\nu}(P_\nu)|$ by Lemma 4.1(d). To get the total running time, we just have to sum this cost over all nodes in the recursion tree.

We first analyze the cost contributed by the $O(|P_\nu|\log f_\nu)$ term. By Lemma 4.2(iii), this cost is α-fading, so we can apply Lemma 2.3. To sum the costs on a given level of the tree, we write $|P_\nu| = |P_{\nu\,|\,\mathrm{int}\,S_\nu}| + |P_{\nu\,|\partial S_\nu}| \le n_\nu + df_\nu$ by Lemma 4.1(g). Since the S_ν's have disjoint interiors over all nodes ν of one level, we have $\sum_\nu n_\nu < n$ and $\sum_\nu f_\nu = f$ for each level of the recursion tree. This gives us an $O((n+f)\log f)$ bound on the cost-per-level. The tree has $O(f)$ leaves, as each leaf discovers at least one facet (note that $F_{S_\nu}(P_\nu) = \emptyset$ only if $\partial S_\nu = \emptyset$ by definition of a simple region). Lemma 2.3(ii) says that the total contribution is $O((n+f)\log^2 f)$.

Next we analyze the cost contributed by the $O(f_\nu \log f_\nu)$ term. This cost may not be α-fading, so we cannot apply Lemma 2.3. But since $\sum_\nu f_\nu = f$ and the recursion tree has depth at most $\log_{1/\alpha} n$ by Lemma 4.2(iii), we can bound the sum of these costs by $O(f \log f \log n)$, which never dominates $O((n+f)\log^2 f)$.

We conclude that the total running time of the algorithm is $O((n+f)\log^2 f)$. Total space is $O(n+f)$ as long as we free up the space used to store the boundary B before we make the recursive calls in line 7.

Theorem 4.6 *Algorithm* `Hull4`() *computes the f-face upper hull of an n-point set $P \subseteq E^4$ in $O((n+f)\log^2 f)$ time and $O(n+f)$ space.*

Remarks:

1. Algorithm `Hull4`() can be considered as a primal-based divide-and-conquer algorithm since it recursively computes the primal restriction of $F(P)$ to a simple region. Alternatively, one may consider an algorithm that computes the dual restriction of $F(P)$ to a simplex recursively. This dual-based approach is perhaps less complex since simplices are easier to handle than boundaries of simple regions. However, the problem with this approach is that the dual analogue of Lemma 4.1(b) is not true in general: that $Q \subseteq P$ contains all vertices of the facets in $F^\Delta(P)$ does not necessarily imply that $F^\Delta(P) = F^\Delta(Q)$—one can construct a counterexample using 3-d point sets and a triangle for Δ.

2. Although the cutting techniques used for dual pruning in our algorithm have been well-studied, our strategy for primal dividing appears new. This strategy provides a simple way to guarantee that the total problem size at any level of the recursion is $O(n+f)$; it would be difficult to obtain such a bound using the existing cutting techniques alone. Previously, primal dividing was used only in two

and three dimensions, notably in the algorithms of Kirkpatrick and Seidel [23] and Edelsbrunner and Shi [18], and our algorithm can be regarded as an extension of these approaches. In fact, the three-dimensional version of our algorithm simplifies Edelsbrunner and Shi's algorithm in the same way as our two-dimensional algorithm simplifies Kirkpatrick and Seidel's. The "contour"-based approach used in a recent parallel 3-d convex hull algorithm by Amato, Goodrich, and Ramos [1] can also be interpreted as a form of primal dividing. There, contours play a role similar to the lower-dimensional upper hulls of $\pi_\Delta(P)$ in Section 4.3 and are used to ensure that the total problem size at any level of the recursion remains $O(n)$; but since they describe their method in the dual setting, its geometry is less apparent in some places.

3. `Hull4()` can return not only $F(P)$ but also a point location structure for the set $\{f\!\downarrow\, : f \in F(P)\}$ of tetrahedra in E^3. We simply maintain the recursion tree and store the planar point location structures from Lemma 4.4 at every node; this requires $O((n+f)\log f)$ space. Then we can find a facet of which the vertical projection contains our given query point by just following a path down the recursion tree. Since the tree has depth at most $\log_{1/\alpha} n$, the query time is $O(\log^2 n)$. For small output size f, we can further reduce the space and query time bound to $O(f \log f)$ and $O(\log^2 f)$ by first calling `Hull4()` to identify the vertices $V(P)$ and then building the point location structure for $V(P)$ instead of P (as $F(V(P)) = F(P)$). We thus achieve the same performance as Goodrich and Tamassia's 3-d point location structure [20].

4. *Some practical issues.* With no additional work, `Hull4()` can return the incidence structure between facets and ridges; this fact can be used to reduce the number of dictionary operations needed. Moreover, with an appropiate choice of coordinate system, it is not necessary to compute the upper and lower hulls separately; we choose to do so here merely because vertical projections are easier to visualize. We should mention that degeneracies may occur in the projected point set $\pi_\Delta(P)$ even though the point set P is itself non-degenerate; in such a situation, we may wish to apply a perturbation to Δ.

5 Clipped Voronoi diagrams

In this section we describe one important application of our four-dimensional convex hull algorithm, namely the output-sensitive computation of Voronoi diagrams in three dimensions. Let P be a set of n point sites in E^d. The *Voronoi region of a site* $p \in P$ consists of all points $q \in E^d$ such that q is closer to p than to any other point in P (with respect to Euclidean distance). The *Voronoi diagram of* P is the collection of all Voronoi regions.

For a given site $p = (p_1, \ldots, p_d) \in P$, we can use a "lifting map" [14, 17] to define a halfspace p^* in E^{d+1}: $p^* = \{(x_1, \ldots, x_{d+1}) : x_{d+1} \geq 2p_1x_1 + \cdots + 2p_dx_d - p \cdot p\}$. It is well known that the Voronoi regions are just the vertical projection of the facets of the polytope $\bigcap_{p \in P} p^*$. Thus, the computation of a Voronoi diagram in E^d is reduced to the computation of an intersection of halfspaces in E^{d+1}. Since the computing an intersection of halfspaces is equivalent to the computing convex hulls by duality, Theorem 4.6 has the following consequence:

Theorem 5.1 *The Voronoi diagram of n point sites in E^3 can be computed in $O((n+f)\log^2 n)$ time and $O(n+f)$ space, where f is the size of the Voronoi diagram ($\Omega(n) = f = O(n^2)$).*

In certain applications, only the portion of a Voronoi diagram lying in a given area is needed, and the size of this portion may be much smaller than the size of the entire Voronoi diagram. What we want is then an output-sensitive algorithm to compute the *Voronoi diagram of P clipped to a region W*, defined simply as the collection of all non-empty intersections of the Voronoi regions with W.

Suppose that W is a k-dimensional polytope $(\bigcap \Gamma) \cap F$, where Γ is a set of m halfspaces and F is a k-flat in E^d. Lift each halfspace $\gamma \in \Gamma$ to a vertical halfspace denoted by γ^* and lift the k-flat to a vertical $(k+1)$-flat denoted by F^*. Then the clipped Voronoi diagram is just the vertical projection of the facets of the polytope $\bigcap_{p\in P} p^* \cap \bigcap_{\gamma\in\Gamma} \gamma^* \cap F^*$. Thus, the clipped Voronoi diagram can be computed by constructing the intersection of the halfspaces $\{p^* \cap F^* : p \in P\} \cup \{\gamma^* \cap F^* : \gamma \in \Gamma\}$ inside the $(k+1)$-flat F^*. If $k = 3$, we can use Theorem 4.6 to compute the intersection of these $n + m$ halfspaces of dimension $k + 1$.

Theorem 5.2 *Let $d \geq 4$ be a constant. The Voronoi diagram of n point sites in E^d clipped to a 3-dimensional polytope defined by m halfspaces can be computed in $O((n + m + f) \log^2 f)$ time and $O(n + m + f)$ space, where f is the size of the clipped Voronoi diagram ($\Omega(1) = f = O(n^2)$).*

6 Extensions

In this section, we describe one way that algorithm `Hull4`() can be extended to higher dimensions. It suffices to provide higher-dimensional analogues of Lemmas 4.4 and 4.5 from Section 4.4, since other parts of the algorithm work in any fixed dimension. Again we may assume that Δ is a halfspace.

The main difficulty that arises when $d > 4$ is that we do not have control on the size of the $(d-1)$-dimensional upper hull of the projected point set $\pi_\Delta(P)$. For $d \leq 4$, the size is $O(|V(P)|)$, which we can bound by $O(|F_S(P)|)$ if $P{\downarrow} \subseteq S$ by Lemma 4.1(d). For $d > 4$, we can only bound the size by $O(|F(P)|)$ and this can be much larger than the actual output size $|F_S(P)|$, since there may exist facets in $F(P)$ with vertical projection outside S even if $P{\downarrow} \subseteq S$. To overcome this difficulty, we do not construct all the hull facets in $F(\pi_\Delta(P))$ but instead apply the results in Section 2.5 to perform queries on $F(\pi_\Delta(P))$.

Lemma 6.1 *Suppose that $d > 4$. Then the restricted point set $P_{|\operatorname{int} S_\Delta}$ can be computed in $O((|P| + (|P|\,|V(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |P|)$ time.*

Proof: As in the proof of Lemma 4.4, we can test whether $p{\downarrow} \in \operatorname{int} S_\Delta$ for a given point $p \in P$ by finding a facet $\pi_\Delta(r)$ of $F(\pi_\Delta(P))$ with $\pi_\Delta(p){\downarrow} \in \pi_\Delta(r){\downarrow}$ and then determining which side of $r{\downarrow}$ the point $p{\downarrow}$ lies on. This facet can be found by performing a linear programming query on a polytope $\mathcal{P}$ defined by $|P|$ dual halfspaces in E^{d-1} corresponding to the $|P|$ points in the projected point set $\pi_\Delta(P) \subseteq E^{d-1}$. As we need $|P|$ queries for each point $p \in P$, the cost is $O(|P|^{2-2/\lceil d/2\rceil} \log^{O(1)} |P|)$ by Theorem 2.2.

To further reduce the running time, we first identify the vertices $\pi_\Delta(v)$ of $V(\pi_\Delta(P))$ $(v \in V(P))$; using an output-sensitive algorithm for computing extreme points by Chan [4], this requires $O((|P| + (|P|\,|V(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |V(P)|)$ time (remember that the point set $\pi_\Delta(P)$ is just of dimension $d-1$). Because $F(\pi_\Delta(P)) = F(V(\pi_\Delta(P)))$, we only need the halfspaces corresponding to these $\leq |V(P)|$ vertices to define our polytope $\mathcal{P}$. The cost of the $|P|$ queries is then only $O((|P| + (|P|\,|V(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |P|)$, according to Theorem 2.2. $\square$

Lemma 6.2 *Suppose that $d > 4$. Given ∂S for a simple region S of P, one can construct $\partial S'$ for a new simple region S' of P with $\operatorname{int} S' = \operatorname{int} S \cap \operatorname{int} S_\Delta$ in $O((|P| + |F_S(P)| + (|P|\,|F_S(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |P|)$ time.*

Proof: As in the proof of Lemma 4.5, it suffices to compute all the boundary components. Then we can select which boundary components contribute to the boundary $\partial S'$—that is, which boundary components correspond to a subregion inside S_Δ—by the techniques of the previous lemma. This requires at most $O(|F_S(P)|)$ linear programming queries on $|P|$ halfspaces in E^{d-1}, and thus can be done within $O((|P| + |F_S(P)| + (|P|\,|F_S(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |P|)$ time.

The boundary computation is again based on depth-first search, but now we cannot generate all the ridges of the boundary B_Δ in advance as we cannot afford to compute all the facets in $F(\pi_\Delta(P))$; rather, a ridge is generated when it is needed.

As before, we store the ridges in ∂S (but not B_Δ) and their $(d-3)$-subfaces in a dictionary and sort these ridges around each $(d-3)$-face σ. Suppose B is a boundary component and we are given a ridge r in B (with its orientation). We first describe how we can generate the $d-1$ ridges that are adjacent to r in B.

These adjacent ridges can be classified into two types: (i) ones that are in ∂S, and (ii) ones that are in the boundary B_Δ. We deal with the adjacent ridges that are in ∂S first. A ridge adjacent to r must share a common $(d-3)$-subface, so let us consider one $(d-3)$-subface σ of r. Look up the dictionary to see if σ is a $(d-3)$-face of ∂S. If so, by performing a binary search on the list of ridges that σ is incident on, we can identify a candidate ridge in ∂S that r may be adjacent to. Repeating this procedure for every $(d-3)$-subface σ of r, we get all the ridges in ∂S that r may be adjacent to.

Next we deal with the adjacent ridges that are in the boundary B_Δ. Again we consider a $(d-3)$-subface σ of r. Determine whether $\pi_\Delta(\sigma)$ is a ridge of $R(\pi_\Delta(P))$; this test can be reduced to a linear programming query on a $(d-1)$-dimensional polytope defined by $|P|$ dual halfspaces. If the test is true, the linear programming query can be used to get a facet $\pi_\Delta(r')$ of $F(\pi_\Delta(P))$ ($r' \in R(P)$) that $\pi_\Delta(\sigma)$ is incident on. Then a ray shooting query in dual space can be used to find (if it exists) the other facet $\pi_\Delta(r'')$ of $F(\pi_\Delta(P))$ ($r'' \in R(P)$) that $\pi_\Delta(\sigma)$ is also incident on. These ridges r' and r'' in B_Δ give two possible candidates for the adjacent ridges of r. We repeat this procedure for every $(d-3)$-subface σ of r.

We now have a list of possible candidates for ridges that may be adjacent to r in B. By performing some local tests, we can deduce which of these ridges are actually adjacent. As in the proof of Lemma 4.5, we can then trace the complete boundary component B by visiting the adjacent ridges recursively in a depth-first manner. To compute all boundary components, we ensure that all ridges in ∂S are visited.

To evaluate total time needed by this computation, observe that the number of ridges visited by the depth-first search is $O(|F_S(P)|)$ by Lemma 4.1(e) since we only generate ridges r with $r{\downarrow} \subseteq S$. The work is then dominated by $O(|F_S(P)|)$ linear programming and ray shooting queries in E^{d-1}, which, by Theorem 2.2, require $O((|P| + |F_S(P)| + (|P|\,|F_S(P)|)^{1-1/\lceil d/2\rceil}) \log^{O(1)} |P|)$ time. $\square$

To get a convex hull algorithm in E^d, we just have to replace Lemmas 4.4 and 4.5 by Lemmas 6.1 and Lemma 6.2 in the algorithm outline for `Hull4()` from Section 4.5. We follow the same notation from Section 4.6 to analyze the running time.

By Lemmas 6.1 and 6.2, the non-recursive part of the algorithm now takes $O((|P_\nu| + |F_{S_\nu}(P_\nu)| +$

$(|P_\nu|\,(|V(P_\nu)+F_{S_\nu}(P_\nu)|))^{1-1/\lceil d/2\rceil})\log^{O(1)}|P_\nu|) = O((|P_\nu|+f_\nu+(|P_\nu|\,f_\nu)^{1-1/\lceil d/2\rceil})\log^{O(1)} n)$ time at node ν, if we recall that $|V(P_\nu)| = |V(P_{\nu\,|S_\nu})| \le d\,|F_{S_\nu}(P_\nu)|$ by Lemma 4.1(d).

To sum this cost, we recall that $\sum_\nu n_\nu < n$ and $\sum_\nu f_\nu = f$ over every level of the recursion tree. Since $|P_\nu| = |P_{\nu\,|\,\mathrm{int}\,S_\nu}| + |P_{\nu\,|\partial S_\nu}| \le n_\nu + df_\nu$ by Lemma 4.1(g), we also have $\sum_\nu |P_\nu| \le n + df$. Using Hölder's inequality, we obtain the following cost-per-level bound, ignoring polylogarithmic factors:

$$\begin{aligned}\sum_\nu \left(|P_\nu| + f_\nu + (|P_\nu|\,f_\nu)^{1-1/\lceil d/2\rceil}\right) &= O\left(n+f \;+\; n^{1-2/\lceil d/2\rceil}\sum_\nu \left(|P_\nu|^{1/\lceil d/2\rceil} f_\nu^{1-1/\lceil d/2\rceil}\right)\right)\\ &= O(n+f \;+\; n^{1-2/\lceil d/2\rceil}(n+f)^{1/\lceil d/2\rceil} f^{1-1/\lceil d/2\rceil})\\ &= O(n+(nf)^{1-1/\lceil d/2\rceil}+fn^{1-2/\lceil d/2\rceil}).\end{aligned}$$

Summing over all $O(\log_{1/\alpha} n)$ levels, we get $O((n+(nf)^{1-1/\lceil d/2\rceil}+fn^{1-2/\lceil d/2\rceil})\log^{O(1)} n)$ as the total running time.

Theorem 6.3 *Let $d > 4$ be a constant. The convex hull of an n-point set in E^d can be computed in $O((n+(nf)^{1-1/\lceil d/2\rceil}+fn^{1-2/\lceil d/2\rceil})\log^{O(1)} n)$ time.*

For odd dimensions d, this method is of no use since it is more complicated and not better than Chan's method [4]; however, for even dimensions d, we do obtain improvement over previous results for a certain range of f. For example, when $f = \Theta(n)$, both Matoušek's method [28] and Chan's method [4] achieves $O(n^{2-2/(\lfloor d/2\rfloor+1)}\log^{O(1)} n)$ time; the method here achieves $O(n^{2-2/\lceil d/2\rceil}\log^{O(1)} n)$ time. In general, if the output size is linear or sublinear, Theorem 6.3 provides the best upper bound currently known for the convex hull problem, ignoring polylogarithmic factors.

An important problem that is left open is then to find a convex hull algorithm in E^d ($d > 4$) with close to $O(n\log f + f)$ running time for the whole range of output size f. Improving our $O((n+f)\log^2 f)$ bound in E^4 would also be interesting.

Note: After the submission of this paper, Amato and Ramos [2] have recently announced an extension of our 4-d algorithm to 5-d, running in $O((n+f)\log^3 f)$ time. They also describe how to adapt our 4-d algorithm to work with degenerate point sets.

Acknowledgements

We thank Nancy Amato, Otfried Schwarzkopf, John Hershberger, and Subhash Suri for discussions on these problems.

References

[1] N. M. Amato, M. T. Goodrich, and E. A. Ramos. Parallel algorithms for higher-dimensional convex hulls. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 683–694, 1994.

[2] N. M. Amato and E. A. Ramos. On computing Voronoi diagrams by divide-prune-and-conquer. In *Proceedings of the 12th Annual ACM Symposium on Computational Geometry*, pages 166–175, 1996.

[3] K. Q. Brown. *Geometric transforms for fast geometric algorithms.* PhD thesis, Carnegie–Mellon University, Pittsburg, Penn., 1980.

[4] T. M. Chan. Output-sensitive results on convex hulls, extreme points, and related problems. *Discrete & Computational Geometry*, to appear.

[5] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, to appear.

[6] T. M. Chan, J. Snoeyink, and C.-K. Yap. Output-sensitive construction of polytopes in four dimensions and clipped Voronoi diagrams in three. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 282–291, 1995.

[7] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete & Computational Geometry*, 9:145–158, 1993.

[8] B. Chazelle. An optimal convex hull algorithm for point sets in any fixed dimension. *Discrete & Computational Geometry*, 10:377–409, 1993.

[9] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[10] B. Chazelle and J. Matoušek. Derandomizing an output-sensitive convex hull algorithm in three dimensions. *Computational Geometry: Theory and Applications*, 5:27–32, 1995.

[11] K. L. Clarkson and P. W. Shor. Applications of random sampling in computational geometry, II. *Discrete & Computational Geometry*, 4:387–421, 1989.

[12] R. A. Dwyer. Higher-dimensional Voronoi diagrams in linear expected time. *Discrete & Computational Geometry*, 6:343–367, 1991.

[13] M. E. Dyer. Linear time algorithms for two- and three-variable linear programs. *SIAM Journal on Computing*, 13(1):31–45, 1984.

[14] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, 1987.

[15] H. Edelsbrunner, L. Guibas, and J. Stolfi. Optimal point location in a monotone subdivision. *SIAM Journal on Computing*, 15:317–340, 1986.

[16] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. *ACM Transactions on Graphics*, 9(1):66–104, 1990.

[17] H. Edelsbrunner and R. Seidel. Voronoi diagrams and arrangements. *Discrete & Computational Geometry*, 1:25–44, 1986.

[18] H. Edelsbrunner and W. Shi. An $O(n \log^2 h)$ time algorithm for the three-dimensional convex hull problem. *SIAM Journal on Computing*, 20:259–277, 1991.

[19] I. Emiris and J. Canny. A general approach to removing degeneracies. *SIAM Journal on Computing*, 24:650–664, 1995.

[20] M. Goodrich and R. Tamassia. Dynamic trees and dynamic point location. In *Proceedings of the 23rd Annual ACM Symposium on Theory of Computing*, pages 523–533, 1991.

[21] B. Grünbaum. *Convex Polytopes.* John Wiley & Sons, London, 1967.

[22] R. A. Jarvis. On the identification of the convex hull of a finite set of points in the plane. *Information Processing Letters*, 2:18–21, 1973.

[23] D. Kirkpatrick and R. Seidel. The ultimate planar convex hull algorithm? *SIAM Journal on Computing*, 15:287–299, 1986.

[24] J. Matoušek. Approximations and optimal geometric divide-and-conquer. *Journal of Computer and System Sciences*, 50:203–208, 1995.

[25] J. Matoušek. Cutting hyperplane arrangements. *Discrete & Computational Geometry*, 6:385–406, 1991.

[26] J. Matoušek. Efficient partition trees. *Discrete & Computational Geometry*, 8:315–334, 1992.

[27] J. Matoušek. Reporting points in halfspaces. *Computational Geometry: Theory and Applications*, 2:169–186, 1992.

[28] J. Matoušek. Linear optimization queries. *Journal of Algorithms*, 14:432–448, 1993.

[29] J. Matoušek and O. Schwarzkopf. On ray shooting in convex polytopes. *Discrete & Computational Geometry*, 10(2):215–232, 1993.

[30] P. McMullen and G. C. Shephard. *Convex Polytopes and the Upper Bound Conjecture*. Cambridge University Press, 1971.

[31] N. Megiddo. Linear programming in linear time when the dimension is fixed. *Journal of the Association for Computing Machinery*, 31:114–127, 1984.

[32] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms*. Prentice-Hall, Englewood Cliffs, N.J., 1993.

[33] F. P. Preparata. Planar point location revisited. *International Journal of Foundations of Computer Science*, 1(1):71–86, 1990.

[34] F. P. Preparata and M. I. Shamos. *Computational Geometry—An Introduction*. Springer-Verlag, New York, 1985.

[35] A. Rényi and R. Sulanke. Über die konvexe Hülle von n zufällig gerwähten Punkten I. *Z. Wahrsch. Verw. Gebiete*, 2:75–84, 1963.

[36] N. Sarnak and R. E. Tarjan. Planar point location using persistent search trees. *Communications of the Association for Computing Machinery*, 29:669–679, 1986.

[37] R. Seidel. Constructing higher-dimensional convex hulls at logarithmic cost per face. In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, pages 404–413, 1986.

[38] G. F. Swart. Finding the convex hull facet by facet. *Journal of Algorithms*, 6:17–48, 1985.

[39] R. Wenger. Randomized quick hull. *Algorithmica*, to appear.