On computing Voronoi diagrams by divide-prune-and-conquer

NANCY M. AMATO
Texas A&M University
amato@cs.tamu.edu

EDGAR A. RAMOS
DIMACS/Rutgers University
ramose@dimacs.rutgers.edu

Abstract

Using a divide, prune, and conquer approach based on geometric partitioning, we obtain: (1) An output sensitive algorithm for computing a weighted Voronoi diagram in \( \mathbb{R}^4 \) (the projection of certain polyhedra in \( \mathbb{R}^5 \)) that runs in time \( O((n + f) \log^3 f) \) where \( n \) is the number of sites and \( f \) is the number of output cells; and (2) a deterministic parallel algorithm in the EREW PRAM model for computing an algebraic planar Voronoi diagram (in which bisectors between sites are simple curves consisting of a constant number of algebraic pieces of constant degree) that runs in time \( O(\log^2 n) \) using optimal \( O(n \log n) \) work. The first result implies an algorithm for the problems of computing the convex hull of a point set and the intersection of a set of halfspaces in \( \mathbb{R}^5 \), and computing the Euclidean Voronoi diagram in \( \mathbb{R}^4 \).

The second result implies both sequential and parallel work-optimal deterministic algorithms for a number of Voronoi diagram problems (including line segments in the plane), and other non-Voronoi diagram problems that can fit in the framework (including the intersection of equal radius balls in \( \mathbb{R}^3 \) and some lower envelope problems in \( \mathbb{R}^3 \)).

1 Introduction

1.1 Problems and results

Voronoi diagrams. Voronoi diagrams are a basic and very important construction in computational geometry, so a considerable amount of work has been done on algorithms for computing them. Given a finite set of sites \( S \) in a space \( X \) and a distance function \( d \) on \( X \), the Voronoi diagram \( V(S) \) is the decomposition of \( X \) into the sets of points not farther from a particular point than from any other, that is, into the Voronoi cells \( V_p = \{ x \in X : d(x, p) \leq d(x, q), q \in S \} \) for \( p \in S \) (possibly empty).

Euclidean Voronoi diagrams. It is well-known that for \( S \subseteq \mathbb{R}^d \) under the Euclidean metric, \( V(S) \) is the downward projection of the boundary complex of a polyhedron obtained as the intersection of halfspaces bounded below by non-vertical hyperplanes. Thus, for an arbitrary collection, \( H \), of such halfspaces in \( \mathbb{R}^{d+1} \), we call such a projection in \( \mathbb{R}^d \), \( V(H) \), a weighted Voronoi diagram; a weighted Voronoi diagram is sometimes referred to as a power diagram [4]. Optimal worst case algorithms (both simple randomized (Clarkson and Shor [12], Seidel [29])) and complicated deterministic (Chazelle [10]) are known for general \( d \); their running time is \( O(n \log n + n^{d/2}) \). But since for \( n = |H| \) the size of \( V(H) \) can be as small as \( \Theta(1) \) or as large as \( \Theta(n^{d/2}) \), it is of interest to have output sensitive algorithms. The aim is to obtain an algorithm with running time that matches the lower bound \( \Omega(n \log f + f) \).

Seidel [28] gave an algorithm with a running time \( O(n^2 + f \log f) \), and Matoušek [25] reduced the \( n^2 \) term (using complicated techniques) to a subquadratic term which is \( n^{4/3} \) (times a polylog factor) for \( d = 3, 4 \). On the other hand, the “gift wrapping” approach by Swart [31] has running time \( O(nf) \), and has been improved by Chan [6] (for \( d = 3, 4 \) the resulting time is \( O(n \log f + (nf)^{4/3} \log^2 n) \)). Chan, Snoeyink and Yap [7, 8] made important progress by giving an algorithm that runs in time \( O((n + f) \log^2 f) \) for \( d = 3 \). Our contribution is to extend this to \( d = 4 \) with an additional log factor: \( O((n + f) \log^3 f) \). Our new insight leads to an approach for arbitrary dimension, but unfortunately it runs into difficulties, so the results that are obtained in higher dimensions are not very interesting. As in [8], some marginal improvement is possible using trade-offs between preprocessing and query times in closest point data structures [6]. A negative feature of both Chan et al.’s algorithm and ours is that nondegeneracy is needed for their analysis. To handle arbitrary input the algorithm is supposed to use a symbolic perturbation technique [15, 32]. Then, for degenerate input the algorithm guarantees the same running time but with \( f \) replaced by the number of faces of the perturbed Voronoi diagram. We verify that without using perturbation the algorithm

---

\(^1\) Supported by a DIMACS (Center for Discrete Mathematics and Theoretical Computer Science) postdoctoral fellowship.
for \( d = 3 \) can be adapted to work for degenerate input with the same time bound. This is interesting because \( d = 3 \) is the most important case in applications. We do not know if the same is true for \( d = 4 \).

**Algebraic planar Voronoi diagrams.** Elaborating on the 2-dimensional version of the algorithm for weighted Voronoi diagrams, we obtain an algorithm for a large class of Voronoi diagrams in the plane (or on the 2-sphere), in which bisectors between sites are simple curves consisting of a constant number of algebraic pieces of bounded degree. We call these **algebraic planar Voronoi diagrams.** They include a large and important subset of the generalized Voronoi diagrams considered by Klein [21]. The algorithm is deterministic and parallel; in the EREW PRAM model it runs in time \( O(\log^2 n) \) using optimal \( O(n \log n) \) work, where \( n \) is the number of sites. This includes, for example, the Euclidean Voronoi diagram of nonintersecting segments in the plane for which the previous best deterministic result, by Goodrich, Ó'Dúnlaing and Yap [19], used time \( O(\log^2 n) \) and \( O(n \log^2 n) \) work in the CREW PRAM model. In another example, computing the intersection of equal radius balls in \( \mathbb{R}^3 \), this results in the only known optimal deterministic sequential algorithm. This latter result was announced in Amato, Goodrich and Ramos [2], but the presentation there was very brief. Here, we remedy that and, at the same time, within a more general framework, give a cleaner algorithm that includes a larger family of problems. Of course, taking advantage of the properties of a concrete problem, a simpler algorithm may be possible; we plan to discuss some particular cases in the final version.

### 1.2 Basic approach

The basic approach uses divide-and-conquer based on geometric partitioning together with a pruning of the input to the subproblems that enforces a global invariant on the total size of the subproblems in each level of the recursion. It dates back to Clarkson and Shor’s [12] randomized output sensitive algorithm for computing halfspace intersections in \( \mathbb{R}^3 \). That approach was refined by Reif and Sen [27] and Amato, Goodrich and Ramos [2] with the aim of obtaining parallel algorithms for halfspace intersections in \( \mathbb{R}^3 \). Chan, Snoeyink and Yap [7, 8] extended the approach to halfspace intersections in \( \mathbb{R}^4 \). The approach of Edelsbrunner and Shi [17] also fits in the same framework, and it is even closer to the primal/dual approach in [8], which we are adopting here (see remark below).

A subproblem has as input a pair \((\sigma, S_\sigma)\) where \( \sigma \subseteq X \) and \( S_\sigma \subseteq S \), and computes the restriction of \( V_\sigma(S) \) to \( \sigma \), denoted \( V_\sigma(S_\sigma) \); \( S_\sigma \) is selected so that \( V_\sigma(S_\sigma) = V_\sigma(S) \). The approach is based on the ability to obtain a \((1/c)\)-cutting of \( \sigma \) for \( S_\sigma \), where \( c > 1 \) is a constant: a decomposition of \( \sigma \) into a set of interior disjoint regions, and for each such region \( \tau \), a **conflict list** \( S_\tau \subseteq S_\sigma \) so that (i) \( S_\tau \) contains at least those sites whose Voronoi cells intersect \( \tau \) (it may contain more), and (ii) \(|S_\tau| \leq |S_{\sigma}| / c\). Let \( T(\sigma) \) denote the collection of these pairs \((\tau, S_\tau)\). The algorithm, on input \((\sigma, S_\sigma)\) performs the following steps:

1. **Cutting:** Compute a \((1/c)\)-cutting \( T(\sigma) \) for \((\sigma, S_\sigma)\).

For each \((\tau, S_\tau) \in T(\sigma) \) do steps 2–4:

2. **Contour:** Compute the **contour** \( V_{bd(\tau)}(S_\tau) \).

3. **Pruning:** Prune \( S_\tau \) to obtain \( S_\sigma \) so that it contains only those sites for which \( V_\sigma \) either touches the contour or is interior to \( \tau \).

4. **Recursion:** If \(|S_\tau| \geq C\) then recurse with \((\tau, S_\tau)\), else finish in \( O(1) \) time.

In step 2, the contour is a lower dimensional Voronoi diagram; a lower dimensional version of the same algorithm may be used if it satisfies the necessary properties. In step 3, the **pruning rule** keeps two types of sites: sites touching the contour, and sites interior to the region. Assuming that Voronoi cells are connected, if a site does not touch any contour, then it can only be interior to one region. This results in an upper bound of at most \( n \) interior sites in all subproblems at the same level of recursion. Keeping all touching sites is only a preliminary rule; it does not produce an efficient algorithm because a Voronoi cell may intersect a region of the cutting without contributing inside it any feature to which the corresponding work can be charged. In fact, if the contour determines the Voronoi diagram inside, then the recursion should stop. Thus, the main point of this work is to establish the right framework where a good pruning rule can be given. In step 4, \( C \) is an appropriate constant.

Another algorithm presented in [5] is flawed and apparently has not been fixed.
Remark. The original output sensitive algorithm by Chan et al. [7] was presented in the dual space, that is, as an algorithm that computes the intersection of halfspaces. Remaining in that context we found the extension. Their revised version [8] uses both the primal and dual. Since this approach leads to a presentation that it is easier to visualize, and possibly more appropriate for an implementation, we follow it here. Thus, a considerable portion of the next section is dedicated to defining the problem in the primal and dual spaces. Once this is done, the algorithm is quite natural.

Contents. The next section describes the application to lower convex hulls, and the following one presents the application to algebraic planar Voronoi diagrams.

2 Output sensitive computation of lower convex hulls

2.1 Preliminaries

Definitions are given for completeness and to establish the notation. A standard reference is [14].

Polytopes and polyhedral. A (convex) polytope in $\mathbb{R}^d$ is the convex hull of a finite point set $S$ (the smallest convex set containing $S$), denoted $\text{conv}(S)$. More general, a (convex) polyhedron is the intersection of a finite set of halfspaces $H$, denoted $\bigcap H$. Let $P$ be a polyhedron. A hyperplane $h$ supports $P$ if it intersects $bd(P)$ and a halfspace bounded by $h$ contains $P$. Then $h \cap P$ is called a face of $P$, and it is said to be supported by $h$. A face $f$ is also a polyhedron (and a polytope if so is $P$). The set of all faces is the boundary complex of $P$. $P$ is simple if the intersection of $k$ facets is $(d + 1 - k)$-dimensional or empty, for $k = 0, \ldots, d + 1$. $P$ is simplicial if each $k$-face is a $k$-simplex with its $k + 1$ vertices in $S$.

Primal and dual spaces. In $\mathbb{R}^{d+1}$, let $\pi_d : \mathbb{R}^{d+1} \rightarrow \mathbb{R}^d$ be the vertical projection $\pi_d(x_0, \ldots, x_d, x_{d-1}) = (x_0, \ldots, x_{d-1})$. We use a symbol with a hat to denote an object in $\mathbb{R}^{d+1}$ and the corresponding symbol without hat to denote its projection under $\pi_d$. Thus, for example, we write $\hat{x} = (x, x_d) \in \mathbb{R}^{d+1}$ where $x = \pi_d(\hat{x}) \in \mathbb{R}^d$. Let the primal and dual spaces, denoted $\hat{P}$ and $\hat{D}$, be copies of $\mathbb{R}^{d+1}$. Let $\hat{P} = \pi_d(\hat{P})$ and $\hat{D} = \pi_d(\hat{D})$. For $\hat{m} = (m, m_d) \in \mathbb{R}^{d+1}$, let $h(\hat{m})$ be the non-vertical hyperplane $\{\hat{y} \in \mathbb{R}^{d+1} : m_d + y_d \cdot m = m \cdot \hat{y}\}$ in $\mathbb{R}^{d+1}$, where $\cdot$ denotes vector product. Let $h(\hat{m})^+$ denote the halfspace in $\mathbb{R}^{d+1}$ bounded from below by $h(\hat{m})$. This establishes a relation between a point in $\hat{P}$ (resp. $\hat{D}$) and a non-vertical hyperplane in $\hat{D}$ (resp. $\hat{P}$). $\hat{p}$ is on (resp. above, and below) $h(\hat{x})$ in $\hat{D}$ iff $\hat{x}$ is on (resp. above and below) $h(\hat{p})$ in $\hat{D}$.

Lower convex hulls and halfspace intersections. Let $\hat{S}$ be a finite point set in $\hat{P}$. The lower convex hull of $\hat{S}$, denoted $\hat{P}(\hat{S})$, is $\bigcap\{h(\hat{x}) : \hat{x} \in \hat{S}, \hat{x} \subseteq h(\hat{x})^+\}$. Let $\hat{D}(\hat{S})$ consist of the finite faces in the boundary complex of $\hat{P}(\hat{S})$ (equivalently, those faces of $\text{conv}\{S\}$ supported by a non-vertical hyperplane from below). Let $\hat{D}(\hat{S})$ be the projection $\pi_d(\hat{D})$. We assume that $\hat{S}$ is in general position, that is, no $k + 2$ points lie in a $k$-flat ($k$-dimensional affine space), for $k = 0, 1, \ldots, d + 1$. Under this condition, $\hat{D}$ is simplicial. $|\hat{D}|$ is the convex hull of $S$. In $\hat{D}$, consider the dual hyperplanes $H(\hat{S}) = \{\hat{h}(\hat{p}) : \hat{p} \in \hat{S}\}$, and let $Q(S) = Q(H(\hat{S}))$ be the intersection of the corresponding halfspaces $\bigcap\{h^+ : h \in H(\hat{S})\}$. Let $V(\hat{S})$ be the boundary complex of $Q(\hat{S})$, and let $\hat{V}(\hat{S})$ be its projection onto $\hat{D}$. Under the general position assumption, $V$ is simple. Note that $\hat{p} \in \hat{P}$ iff $Q \subseteq h(\hat{p})^+$. Let $\hat{V}_P$ be the facet of $Q$ supported by $h(\hat{p})$ (which may be empty). The faces in $\hat{D}$ and in $V$ (hence, those in $\hat{D}$ and in $\hat{V}$) are in one-to-one correspondence: The $k$-face $\hat{f} = \text{conv}(\hat{T})$ where $\hat{T} = \{\hat{p}_0, \ldots, \hat{p}_k\}$ in $\hat{D}$ is in correspondence with the $(d - k)$-face $\hat{f} = \bigcap\{\hat{V}_P : \hat{p} \in \hat{T}\}$ in $\hat{V}$; we write $\hat{g} = D(\hat{f})$ and $\hat{f} = P(\hat{g})$, and similarly for the projections.

Delaunay triangulations and Voronoi diagrams. For $S \subseteq \hat{P}$, let $\hat{S} = \{(p, \frac{1}{2} \hat{p} \cdot \hat{p}) : \hat{p} \in S\}$. Then $\hat{D}(\hat{S})$ and $\hat{V}(\hat{S})$ are the usual Delaunay triangulation and Voronoi diagram of $S$ ($\hat{V}_P = \pi_d(\hat{V}_P)$ is the Voronoi cell of $p$). Thus, for arbitrary $\hat{S}$, we may call $\hat{D}(\hat{S})$ a weighted Delaunay triangulation, and $\hat{V}(\hat{S})$ a weighted Voronoi diagram (this is also called a power diagram in the literature).

Primal and dual restrictions. Consider a $d$-polytope $\sigma$ in $\hat{D}$, and let $\sigma_\sigma$ be the restriction of $\hat{V}$ to $\sigma$: $\sigma_\sigma = \{f \cap \sigma : f \in \hat{V}\}$. We assume that $\sigma$
is in general position, that is, if \( f \) is \( k \)-dimensional then \( f \cap \sigma \) is \( (k - 1) \)-dimensional.\(^6\) We also assume that \( V_\sigma \) includes vertices, for otherwise \( V_\sigma \) is completely determined by the contour of \( \sigma \). \( P \) is extended to \( V_\sigma \) by defining \( P( \sigma \cap f) = P(f) \). Let \( D_\sigma = P(V_\sigma) = \{ P(f) : f \in V, f \cap \sigma \neq \emptyset \} \) be the corresponding subcomplex of \( D \). Similarly for the lifted counterparts. The underlying space of \( D_\sigma \) is connected; its boundary consists of the union of those \( (d - 1) \)-simplices \( s \in D \) for which \( D(s) \cap \text{bd}(\sigma) \neq \emptyset \). Thus, \( \text{bd}(D_\sigma) \) corresponds to the contour of \( \sigma \) in \( D \). Topologically, \( \text{bd}(D_\sigma) \) is a \( (d - 1) \)-sphere with patches identified (corresponding to faces of \( V \) that intersect \( \text{bd}(\sigma) \) in more than one component). Let \( \sigma_\alpha = \text{cl}(\text{int}(D_\sigma)) \) and let \( \text{bc}(\sigma_\alpha) \) denote its boundary complex. Note that in \( \sigma_\alpha \subseteq P \), \( (d - 1) \)-dimensional patches which are identified disappear because they do not bound the interior. As a result, \( \sigma_\alpha \) may break into connected components and \( \text{bc}(\sigma_\alpha) \) may consist of several boundary components. See figure 1. The primal restriction to \( \sigma_\alpha \) is \( D_{\sigma_\alpha} = \{ f : f \cap \sigma_\alpha \neq \emptyset \} \).\(^7\) This is equal to \( \{ f \in D : f \subseteq \sigma_\alpha \} \) because of the definition of \( \sigma_\alpha \) as the underlying space of a subcomplex.

![Figure 1: Primal and dual restrictions.](image)

\(^6\)Algorithmically, this must be enforced through symbolic perturbation. This is in addition to the general position assumption for \( \hat{S} \) already stated.

\(^7\)As in [8], for a complex in \( P \) we use superscript for a dual restriction (with respect to a region in \( D \)), and a lowerscript for a primal restriction (with respect to a region in \( P \)).

the desired cutting is obtained. This can be improved by the approach of resampling. However, from a rough calculation, it appears the size of, say, a \((1/2)\)-cutting is impractically large already for \( d = 3, 4 \) which are our main interest. An alternative, as pointed out in [8] is the partitioning introduced by Meggido [26] (see [14]). This is worse in its asymptotic growth with \( d \), but comparable for sufficiently small \( d \). Chan et al [7, 8] combine the first step of Meggido’s approach (pairing hyperplanes and projecting down their pairwise intersections) and a cutting in \( D \) for hyperplanes. This does not seem better than the sampling scheme above. The problem of whether it is possible to reduce these constants to practical values for small \( d \), say \( d \leq 4 \), deserves careful consideration.

**Refining a region with a halfspace.** In a divide step, \( \sigma \) is split into smaller polytopes via a cutting. As in [8], it is sufficient to consider a refinement \( \tau = \sigma \cap h' \) where \( h' \) is a halfspace bounded by a hyperplane \( h = \{ x \in D : \mathbf{m} \cdot x = m_d \} \) in \( D \) with \( \mathbf{m} \neq 0.\(^8\) \) \( V_h \) is a \( (d - 1) \)-dimensional Voronoi diagram. We assume that \( h \) is in general position so that for a \( k \)-face \( f \) in \( V \), \( f \cap h \) is a \( (k - 1) \)-face in \( V_h \) or empty, and, hence, the primal counterpart \( P(V_h) \), denoted \( D_h \), is a \( (d - 1) \)-dimensional subcomplex of \( D \). Let \( \hat{M} \) be the \( d \)-subspace of \( \hat{P} \) orthogonal to \( \mathbf{m} = (m_m, m_d) \) and let \( M \) be the \( (d - 1) \)-subspace of \( P \) orthogonal to \( \mathbf{m} \) (so \( M = \hat{M} \cap P \)). Let \( \pi_M : \hat{P} \rightarrow \hat{M} \) and \( \pi_M : P \rightarrow M \) be orthogonal projections. \( \pi_M(D_h) \) is the \( d \)-dimensional lower convex hull of \( \pi_M(S) \) in \( \hat{M} \) (where the vertical direction in \( \hat{M} \) is the projection of the vertical direction in \( \hat{P} \); note that \( \hat{M} \) is not horizontal). Thus, \( D_h \) is monotone in the direction of \( \mathbf{m} \) in \( P \). \( D_h \) splits \( D_{\sigma_\alpha} \) into two parts, one of them corresponding to \( \tau \). The boundary complex \( \text{bc}(\sigma_\alpha) \) can be obtained from \( D_h \) and \( \text{bc}(\sigma_\alpha) \). For a point \( p \), to determine on which side of \( D_h \) it lies, perform a point location for \( \pi_M(p) \) in \( \pi_M(D_h) \).

**2.3 The algorithm**

**Key idea.** The key idea in the algorithm [8] is, when recursing on \( \sigma \subseteq D \), to restrict work to \( \sigma_\alpha \subseteq P \) to \( \sigma_\alpha \); instead of computing \( D_\sigma \), the subproblem computes \( D_{\sigma_\alpha} \). Thus, those portions of \( \text{bd}(D_\sigma) \) that do not bound the interior are disregarded and no further work is wasted on

\(^8\)This directly fits the partitioning by Meggido since it consists of a tree of partitions by hyperplanes. But every polytope in any arbitrary partition can also be produced in this manner, using for each region of the cutting, a partitioning hyperplane for each facet in its boundary.
recomputing already known faces in further levels of the computation. The algorithm keeps track of the boundary complex \( bc(\sigma) \) of the regions in \( \mathcal{P} \) corresponding to \( \sigma \) where \( \mathcal{D} \) is still to be computed. Our improvement for \( d + 1 = 5 \) results by computing contours by using the same algorithm recursively in the next lower dimension. We can give a general plan for arbitrary dimension, but it can only be completed efficiently for \( d + 1 \leq 5 \) (or rather patched up).

**Basic algorithm.** The input for a subproblem in the algorithm is a triple \((\sigma, bc(\sigma), \hat{S}_\sigma)\) where \( \hat{S}_\sigma \) is an appropriate subset of \( \hat{S} \) so that \( \mathcal{D}_{\sigma_0}(\hat{S}_\sigma) = \mathcal{D}_{\sigma_0}(\hat{S}) \). \( \hat{S}_\sigma \) contains all points \( \hat{p} \in \hat{S} \) with \( p \in \sigma \) except possibly some redundant points detected by the cuttings, and including points on the boundary \( bc(\sigma) \). The components of \( \sigma \) are not dealt with independently because it is difficult to split \( \hat{S}_\sigma \) between them. The following outline follows closely that in [8]:

**Algorithm Hull** \( (\sigma, bc(\sigma), \hat{S}_\sigma) \)
1. If \(|\hat{S}_\sigma| \leq C\) then return \( \mathcal{D}_{\sigma_0}(\hat{S}_\sigma) \) in \( O(1) \) time
2. Compute a \((1/c)\)-cutting \( T(\sigma) \) for \((\sigma, \hat{S}_\sigma)\)
3. For each \((\tau, \hat{S}_\tau) \in T(\sigma)\) do
   4. Compute \( bc(\tau_\sigma) \) where \( \tau_\sigma = cl(\text{int}(\mathcal{D}(\tau_\sigma) \cap \sigma)) \)
5. Prune \( \hat{S}_\tau \) to obtain \( \hat{S}_\tau \)
6. Return \( \bigcup \{\text{Hull}(\tau, bc(\tau_\sigma), \hat{S}_\tau) : bc(\tau_\sigma) \neq \emptyset\} \)

**Pruning rule.** \( \check{x} \in \hat{S}_\tau \) is retained in \( \hat{S}_\tau \) if either (i) \( \check{x} \) is a vertex in \( bc(\tau_\sigma) \), or (ii) \( \check{x} \) is interior to \( \tau_\sigma \). This rule results in a global bound on the size of the subproblems at each level of the recursion. This is analogized later. Type (i) points are directly obtained from \( bc(\tau_\sigma) \); type (ii) points are identified using point location as observed earlier. The algorithmic problem is discussed later.

**Recursion with a lower dimensional problem.** We continue assuming \( \tau = \sigma \cap h' \) where \( h' \) is a halfspace bounded by a hyperplane \( h \) in \( \mathcal{D} \). The first step is to compute \( \mathcal{D}_h(\hat{S}) = \{f \in \mathcal{D}_h : f \subseteq \sigma\} \). In general, outside \( \sigma \), \( \mathcal{D}_h(\hat{S}) \) cannot be computed from \( \hat{S}_\sigma \). In fact, outside \( \sigma \), \( \mathcal{D}_h(\hat{S}_\sigma) \) may have spurious faces, that is, faces that are not in \( \mathcal{D}(\hat{S}) \). Let \( \mu = |\pi_M(\mathcal{D}_h(\hat{S}))| \) and \( bc(\mu) \) be its boundary complex. \( bc(\mu) \) is the projection of a subcomplex of \( bc(\sigma) \); it can be determined by inspecting each face in \( bc(\sigma) \) to decide whether its projection is in \( bc(\mu) \); there is further explanation on this later. Now, the algorithm recurses with \( (\mu, bc(\mu), \hat{S}_\mu) \) where \( \mu = \sigma \cap h \) and \( \hat{S}_\mu = \pi_M(\hat{S}_\sigma) \cap \hat{S}^{\mu-1}(\mu) \), having \( \hat{M} \) and \( \pi_{\sigma-1}(\text{affine}(\mu)) \) as new primal and dual spaces of dimension \( d \). This is a hull computation in the next lower dimension and the result is \( \pi_M(\mathcal{D}_h(\hat{S})) \) from which \( \mathcal{D}_h(\hat{S}) \) is directly obtained. Note that because of the recursion, in step 4, \( \tau_\sigma \) is in general not equal to \( cl(\text{int}(\mathcal{D}_h(\tau_\sigma))) \). The intersection with \( \pi_{\sigma-1}(\mu) \) is to obtain only points whose downward projection is in \( \mu \). Algorithmically, this requires a point location data structure, which is not feasible in general, but the difficulty can be solved for \( d + 1 \leq 5 \).

**Computing \( bc(\tau_\sigma) \).** We now have \( \mathcal{D}_h(\hat{S}) \) and \( bc(\sigma) \). \( bc(\tau_\sigma) \) can be obtained from \( \mathcal{D}_h(\hat{S}) \) as follows. The algorithm concentrates on determining the facets and ridges in \( bc(\tau_\sigma) \) from which the complex can be constructed. \( \mathcal{D}_h(\hat{S}) \) and \( bc(\sigma) \) are merged into a structure in which each ridge has its incident facets sorted around it. Facets are directed: those in \( \mathcal{D}_h(\hat{S}) \) with the direction corresponding to \( \mu \), and those in \( bc(\sigma) \) towards the interior. By walking on the resulting graph of facets and edges, those components of \( bc(\tau_\sigma) \) with facets in \( \mathcal{D}_h(\hat{S}) \) are identified. Other components are either completely inside or outside. This can be determined with the point location used to determine interior points. Further details are given in [8], and will be given in the complete version of this paper.

**Determining \( bc(\mu) \).** For a ridge \( r \) in \( bc(\sigma) \), we want to determine whether \( r' = \pi_M(r) \) is a facet in \( bc(\mu) \) (note the difference in dimension). Let \( f \) and \( g \) be the facets in \( bc(\sigma) \) incident to \( r \). \( \pi_M(r) \) is in \( bc(\mu) \) if \( f \) and \( g \) are on opposite sides of \( \mathcal{D}_h(\hat{S}) \). Let \( f' \) be the facet in \( \mathcal{D} \) incident to \( f \) outside \( \sigma \), and \( g' \) is in \( bc(\mu) \) if \( f \) and \( g \) are on opposite sides. Alternatively it would be sufficient if \( f \) and \( g \) are on the boundary of \( cl(\text{int}(\mathcal{D}_h(\tau_\sigma))) \). This latter case is always true in the first level of the recursion of Hull but not true in general in deeper levels. Then, it is necessary to compute attaching facets. We lack an efficient procedure to do this in general, but we can get around it for \( d + 1 \leq 5 \). Figure 2 illustrates the situation in the dual space: the 2-face \( p \) corresponding to \( r \) is shown in its affine space, the triangle is the intersection with \( \sigma \) and the line segment is the intersection with \( h \). The portion of the boundary \( b \) of \( p \) already computed is

*Note: A different approach is not to enforce this and to allow points with projection outside \( \mu \); still the bound on total subproblem size holds because such points are retained only in one subproblem when splitting with a cutting. They are identified as being outside at the completion of subproblem \( \mu \). The presentation is cleaner if those points are not allowed.*
shown dashed. Vertices on \( b \) correspond to facets, and they must be known to be able to determine whether one of the intersections of \( b \) with \( h \) is known while the other is not.

\[
\text{Figure 2: .}
\]

**Performing point location.** If \( D^h(\hat{S}) \) were always computed (complete, not just its restriction to \( D_{\sigma_a} \)) in every level of recursion, then the resulting tree structure could be used to answer point location queries in time \( O(\log^d n) \). \( \log n \) can be changed into \( \log f \) if at every level the construction is repeated a second time without the redundant sites. Unfortunately, we can only afford to construct \( D^h_{\sigma_a} \), and as a consequence the point location capability breaks down: the resulting structure has gaps, and some queries may fail by ending in a gap. Fortunately, we can get around this for \( d + 1 \leq 5 \).

**Total size of subproblems.** The total conflict list size of all subproblems generated during the computation of the modified algorithm is \( O(n + f) \) times a polylog factor. Thus, one could afford polylog work per site in a subproblem and still obtain work \( O(n + f) \) times a polylog factor for the algorithm. We can only do this for \( d + 1 \leq 5 \) by appropriately patching up the algorithm.

**Lemma 1** The total conflict list size of all subproblems generated during the computation of the modified algorithm is \( O((n + f) \log^d f) \).

**Proof:** Let \( \sigma_0 \) be an initial “large” \( d \)-simplex in \( D \). Let \( \sigma \) be a top \( k \)-polytope, that is, not contained in a larger \( k \)-polytope in the computation. Let \( T_k(\sigma) \) be the tree generated by the recursive algorithm on \( \sigma \): the root is \( \sigma \), and the children of a node \( \tau \) in \( T_k(\sigma) \) are the \( k \)-polytopes in the cutting \( T(\tau) \). Associated with each node \( \tau \) in \( T_k(\sigma) \), and for each facet \( \mu \) of \( \tau \) there is a tree \( T_{k-1}(\mu) \) corresponding to the computation of the contour on \( \mu \). For \( \tau \), let \( N_\tau \) be the number of facets of \( D \) inside \( \tau \), and let \( I_\tau \) be the number of interior points \( \tau \). Thus, \( N_{\tau_0} = O(f) \) and \( I_{\tau_0} = O(n) \). The conflict list \( S_{\tau_0} \) has size \( O(N_\tau + I_\tau) \). The sum of \( N_\tau \) over a level of \( T_k(\sigma) \) is \( O(N_\sigma) \), and the sum of \( I_\tau \) over a level of \( T_k(\sigma) \) is \( O(I_\sigma) \) (since a site interior to \( \sigma \) and not touching any contour inside can be interior to only one descendant of \( \sigma \)). Since the cuttings have constant size, the sum of \( N_\mu \) over sides of simplices \( \tau \) in a level of \( T_k(\sigma) \) is \( O(N_\sigma) \). The sum of \( I_\mu \) over the same range is \( O(N_\sigma + I_\sigma) \). Since the depth of each tree is \( O(\log n) \), one obtains \( O((n + f) \log^d n) \) as a global bound for all conflict lists. A lemma of Edelsbrunner and Shi [17] refined in [8] allows to substitute \( \log^d n \) with \( \log d f \).

**Patching up the algorithm in low dimensions.** We indicate how to patch up the algorithm for \( d + 1 = 5 \). First, we deal with the computation of attaching facets. In the top level of the recursion, as pointed out above, computing attaching facets is straightforward. In the second level of the recursion we get around the difficulty as follows: \( D^h(\hat{S}_{\sigma_a}) \) is \( 2 \)-dimensional, so we can afford to compute it everywhere (including spurious faces outside \( \sigma_a \)). Then, having \( bc(\sigma_a) \) and \( D^h(\hat{S}_{\sigma_a}) \), we can identify \( bc(\mu^* \sigma_a) \) and spurious faces simultaneously.

Second, we deal with the point location problem. As pointed out above, in the top dimension is not possible to end up in a gap. In the second level, we can compute all of \( D^h(\hat{S}_{\sigma_a}) \), so that there are no gaps. Since part of this structure is spurious, we need to argue that it still produces correct results. But this is obvious because the spurious portion is outside \( \sigma_a \). This also solves the problem of finding the restriction to \( \sigma_{\mu^{-1}}^{-1}(\mu^*) \) when using recursion: this is not needed in the first level of recursion; and in the lower levels the point location problem is at most \( 2 \)-dimensional and can be solved efficiently.

With the two difficulties patched up we conclude that there is an algorithm with a running time \( O((n + f) \log^d f) \). Using an optimal \( O(n \log f) \) algorithm for the \( 2 \)-dimensional subproblems, one log factor is gained.

**Theorem 2** Let \( 2 \leq d \leq 4 \) and \( \hat{S} \in \mathbb{R}^{d+1} \) be a set of points in general position. Then \( D(\hat{S}) \) can be computed in time \( O((n + f) \log^{d-1} f) \), where \( n \) is the size of \( \hat{S} \) and \( f \) is the number of faces in \( D(\hat{S}) \).

**Higher dimensions.** In higher dimensions, the algorithm can be patched up using linear programming queries as in [8]. The improvement over previous results is only marginal.

**Handling degeneracies for \( d + 1 = 4 \).** Since Voronoi diagrams in dimension 3 are specially important from the point of view of applications, it is interesting to verify that for this case the algorithm can actually handle
degeneracies so that the running time still is \( O((n + f) \log^2 f) \). It is mostly a tedious description of how to handle different degeneracies. Here, we only point out that the analysis for the total size of the subproblems goes through. Consider \( \mathcal{C} = bc(\tau_s) \). We argue that the 2-faces in \( \mathcal{C} \) can be charged for the vertices in \( \mathcal{C} \), all of which are retained in \( \mathcal{T}_s \). Let us assume \( \mathcal{C} \) is connected. \( \mathcal{C} \) may be pinched at some vertices and edges (but no 2-faces). If these are removed by replicating them and moving them away, then \( \mathcal{C} \) becomes a surface of genus bounded by the number of edges. Then, using the Euler’s formula relating the numbers of 0-1- and 2-dimensional faces and the genus, we get that the number of vertices is at most proportional to the number of 2-faces. The replication is only by a constant factor if we assume a cutting into cells of constant complexity.

**Theorem 3** Let \( \hat{S} \in \mathbb{R}^{3+1} \) be a set of points. Then \( \mathcal{D}(\hat{S}) \) can be computed in time \( O((n + f) \log^2 f) \) (even in the case that \( \hat{S} \) is degenerate).

### 3 Algebraic planar Voronoi diagrams

#### 3.1 Definition

We consider a set \( S \) of \( n \) sites in the plane (alternatively on the sphere). \( p \in S \) is just an identifier for the site but it may represent an object \( O(p) \) in the plane. We say that a curve is piecewise algebraic (p.a.) if it consists of a constant number of algebraic pieces, each of bounded degree. We assume that: (A0) \( O(p) \) is a p.a. segment.\(^\text{10}\) More complicated objects, like a polygon of non-constant size, can be decomposed into simpler objects. For each pair \( p, q \in S, p \neq q \), there exists a curve \( b(p, q) \) (equal to \( b(q, p) \)), called the bisector of \( p \) and \( q \), such that: (A1) \( b(p, q) \) is a p.a. simple curve either closed or unbounded in both directions. Thus, \( b(p, q) \) divides the plane into two (closed) regions. One of them is associated with \( p \) and denoted \( h_p(p, q) \) and the other is associated with \( q \) and denoted \( h_q(p, q) \). We make the following nondegeneracy assumptions: (A2) Any two bisectors intersect only at crossing points and any four bisectors have empty intersection. As usual, degeneracies can be handled using symbolic perturbation techniques. The cell of \( p \in S \) is \( V_p(S) = \bigcap_{p \neq q \in S} h_p(p, q) \). \( O(p) \) need not be contained in \( V_p(S) \). We assume that: (A3) For any site \( p \) and any three other sites \( q_1, q_2, q_3 \), \( V_p(\{ p, q_1, q_2, q_3 \}) \) is empty or simply connected. It follows that for any \( R \subseteq S \) and any \( p \in S, V_p(R) \) is empty or simply connected. Because of the nondegeneracy assumption, the Voronoi diagram \( \mathcal{V}(S) \) consists of 2-dimensional cells \( V_p(S) \), called faces; 1-dimensional cells, connected components\(^\text{11}\) of \( V_p(S) = V_p(S) \cap V_q(S) \), called edges; and 0-dimensional cells, connected components of \( V_{p,q,r}(S) = V_p(S) \cap V_q(S) \cap V_r(S) \), called vertices (some of these may be empty). Since the graph of \( \mathcal{V}(S) \) (edges and vertices in \( \mathcal{V}(S) \)) is planar, there are at most \( n \) faces, and the degree of a vertex is greater than 2, we see that the total number of faces, edges and vertices is \( O(n) \). In order to apply the results of geometric sampling, we need one further assumption: (A4) For any \( p \in R \subseteq S \) and vertex \( v \in V_p(R) \) one can draw a unique geodesic (simple) curve \( g(v, p) \) inside \( V_p(R) \) which is a p.a. segment with an endpoint at \( v \) and the other either on some edge of \( V_p(R) \) or at a point inside \( V_p(R) \) on \( O(p) \); two geodesics \( g(v, p) \) and \( g(w, p) \) may intersect only at an endpoint. The geodesics through all the vertices in \( V_p(R) \), together with \( O(p) \), induce a decomposition \( \mathcal{T}_p(R) \) of \( V_p(R) \) into cells, called trapezoids, of constant complexity (at most two Voronoi edges, two geodesics and possibly \( O(p) \) bound each trapezoid). The collection of all trapezoids in \( \mathcal{T}_p(R) \), for all \( p \in R \), form the trapezoidal decomposition of \( \mathcal{V}(R) \), denoted \( \mathcal{T}(R) \). The size of \( \mathcal{T}(R) \) is \( O(|\mathcal{V}(R)|) \). For \( \sigma \in \mathcal{T}(R) \), the conflict list of \( \sigma \), denoted \( S_{tr} \), consists of those sites \( q \in S \) such that \( h_p(p, q) \) does not contain \( \sigma \). Clearly, only the sites in \( S_{tr} \) can affect \( \mathcal{V}(R) \) restricted to \( \sigma \), denoted \( \mathcal{V}_r(S) \). Finally, we assume that: (A5) Some basic operations on the p.a. curves (bisectors, geodesics and objects) can be performed in constant time.

#### 3.2 Algorithm

We follow the basic approach outlined in the introduction, with a refined pruning rule that is the dual version of that in the previous section. But this is somewhat complicated since we allow multiple intersections between bisectors and between bisectors and the boundaries of regions. Also a different approach for determining interior sites is needed and, to obtain optimal \( O(n \log n) \) work, a faster partitioning is used: a cutting of size \( O(n^2) \) rather than \( O(1) \).

**Cutting.** Under the assumptions above, the following

---

\(^{10}\)Note that in some cases, such as the ball intersection application, \( O(p) \) may not exist.

\(^{11}\)Note that it is possible to have more than one component, for example, in the Voronoi diagram of line segments, or in the intersection of unit balls in \( \mathbb{R}^3 \).
Lemma 4 For $0 < \delta < 1$, there are constants $C,C_0 \geq 0$ such that for $0 < \epsilon \leq \epsilon_0$ and $\epsilon_0 \leq \tau \leq \epsilon^\delta$, a sample $R \subseteq S$ with the following properties can be constructed deterministically in time $O(n \log \tau)$: (i) $|R| = \frac{3\tau}{2}\leq \tau$, (ii) $\max_{r \in T(R)} |S_{r}| \leq \left(\frac{n}{\tau}\right)^{1/2}$, (iii) $|T(R)| \leq C\tau$, and (iv) $\sum_{r \in T(R)} |S_{r}| < Cn$. The decomposition $T(R)$ and the conflict lists can be computed within the same time bound. The computation can be performed deterministically in the EREW PRAM model in time $O(\log^2 n)$ and $O(n \log \tau)$ work.

This result is obtained by a derandomization that uses efficient construction of $(1/\tau)$-approximations using partition trees [24] together with the technique of linearization [22]. For this the algebraic restriction is important. This makes the algorithm quite complicated. If the sampling is not derandomized, we obtain a simpler randomized version of the algorithm with a faster $O(\log n)$ running time (still, for the parallel algorithm, linearization and point location in hyperplane arrangements are needed to compute conflict lists efficiently). Sequentially, the $r^{\delta}$ term in (ii) can be replaced by $C \log n$. Using the lemma, it follows that for $(\sigma, S_{\sigma})$, there is a $(1/r^{1-\delta})$-cutting $T(\sigma)$ of size $O(r)$ and with total conflict size $O(n)$.\textsuperscript{12}

Non redundant and new portions. In this context, in general, there is no corresponding geometric (primal) triangulation. Although an abstract triangulation can be defined, it does not seem advantageous to make use of it. Let $\tau$ be a trapezoid and $\mu$ be its boundary. The restricted Voronoi diagram $V_\tau(S)$ consists of the components of $f \cap \tau$ for $f \in V(S)$. Note that a cell in $V(S)$ can originate more than one cell in $V_\tau(S)$, and at most a constant number of them. Similarly for the restriction on the boundary, that is, the contour $C(\tau) = V_\mu(S)$. A 1- or 2-cell in $V_\tau(S)$ which is not incident to 0-cells (vertices) inside $\tau$ is said to be green, and the 0- and 1-cells of its boundary in $C(\tau)$ are also said to be green. A site is said to be redundant, for $\tau$, if all the faces it contributes in $\tau$ are green. The redundant portion of $\tau$ is the union of the faces corresponding to redundant sites; the non redundant portion, denoted $\tau_{sr}$, is the union of the other faces. The non redundant portion of the contour $C_{sr}(\tau)$ is the restriction of $C(\tau)$ to $\tau_{sr}$. The (green) edges that bound the redundant portion are called attaching edges. They generate a subdivision $D(\tau)$ of $\tau$ into the components of the redundant and non redundant portions. Suppose $\tau$ is a trapezoid in the cutting of a trapezoid $\sigma$. The portion $\tau_{nr} = \tau \cap \sigma_{nr}$ is called the new portion of $\tau$. When recursing on $\tau$, the algorithm is to construct $V_{\tau_{nr}}(S)$, as the remaining portion is already known. The corresponding new portion of the contour is denoted $C_{nr}(\tau)$. Point location in $D(\sigma)$ is used to keep the work to the non redundant portion of $\sigma$.

Outline. the cutting result above is used as the basis of a divide-and-conquer algorithm. Let $n_{sr} = |S_{sr}|$ and $n_{10} = |S_{10}|$. Using $\delta = 1/2$ and $r = n_{sr}$ in the sampling lemma above, then $n_{10} \leq n_{sr}^{1-\epsilon}$ where $\epsilon = \epsilon/2$. Let $n_{i}$ be the maximum problem size in the $i$-th level. Then $n_{i} \leq n_{1}^{1/2-i}$. If we guarantee the invariant that the overall problem size in the $i$-th level is $O(n)$, and that in a subproblem $\sigma$, the amount of work (excluding recursion) is $O(n_{sr} \log n_{sr})$ with running time $O(\log^2 n_{sr})$, then the total work is $O(n \log n)$ and the running time $O(\log^2 n)$. Because of the properties of the cutting, for each $\tau \in T(\sigma)$ an amount of work $O(n_{10} \log n_{sr})$ is allowed. Our goal is to show that pruning enforces the invariant within the time and work constraints.

To emphasize the parallelism, the algorithm below is described in rounds. $T_i$ is the set of active trapezoids in round $i$. The steps in the $i$-th round are as follows. Details are given below.

For each $(\sigma, S_{sr}) \in T_i$ do
1. If $|S_{sr}| \leq C$ then finish in time $O(1)$
2. Compute cutting $T(\sigma)$
3. For each $(\tau, S_{10}) \in T(\sigma)$ do
4. Compute corrupted contour $C_{c}(\tau)$
5. Determine new contour $C_{nr}(\tau)$ using $D(\sigma)$
6. Determine $S_{\sigma}$ and $S_{\sigma}^{nc}$
7. Compute $V(S_{\sigma}^{nc})$ and its D-K hierarchy
8. Determine green faces and interior sites
9. Determine non redundant contour $C_{sr}(\tau)$
10. Prune $S_{10}$ into $S_{sr}$
11. Put $(\tau, S_{sr})$ in $T_{i+1}$

Pruning rule. The pruning rule is quite obvious: include in $S_{sr}$ those sites in $S_{sr}$ that are either interior to $\tau$ or incident to an (non redundant) edge in $C_{sr}(\tau)$. This enforces the global bound $O(n)$ on the size of all subproblems in the same round: since each of the $n$ sites can be interior to at most one subproblem, and because each of the $O(n)$ vertices of $V(S)$ is incident to only 3 sites (hence it can be charged in at most 3 subproblems).

\textsuperscript{12}The factor $r^{\delta}$ can be dropped completely if resampling is used, both sequentially and in parallel [11, 18, 2]; that results in a $(1/r)$-cutting of size $O(r)$. 
Computing contours. Consider the boundary \( \mu \) of a trapezoid \( \tau \). In order to obtain the new contour \( C_{\mu}(\tau) \), the algorithm first computes \( C_{\mu}(\tau) = V_{\mu}(S_{\mu}) \), which is a corrupted version of the contour that includes \( C_{\mu}(\tau) \) as well as spurious edges and vertices (i.e., edges and vertices that do not appear in \( V(S) \), but may appear here due to the pruning). Since a bisector can intersect \( \mu \) several times, one cannot use a 1-dimensional version of our divide-and-conquer algorithm to obtain \( C_{\mu}(\tau) \). A different divide-and-conquer approach is needed: to compute \( V_{\mu}(T) \), \( T \) is divided into two nearly equal parts \( T_1 \) and \( T_2 \). \( V_{\mu}(T_1) \) and \( V_{\mu}(T_2) \) are computed recursively, and then merged; see [30, pages 134–135]. The computation takes time \( O(n_{\mu} \log n_{\mu}) \) sequentially, and time \( O((\log^2 n_{\mu}) \log n_{\mu}) \) and work \( O(n_{\mu} \log^2 n_{\mu}) \) in the EREW PRAM. Note that \( |V_{\mu}(S_{\mu})| = O(n_{\mu}) \), since \( |V(S_{\mu})| \) is \( O(n_{\mu}) \) and \( \mu \) intersects an edge of \( V(S_{\mu}) \) at most a constant number of times. In the previous level of the recursion, for \( \sigma \), a point location for the subdivision \( D(\sigma) \) has been computed. It supports queries in time \( O(\log n_{\sigma}) \). A vertex in \( C_{\sigma}(\tau) \) is spurious if it is in the redundant portion of \( D(\sigma) \). Thus, spurious vertices are identified in time \( O(\log n_{\sigma}) \) using work \( O(n_{\mu} \log n_{\sigma}) \).

The contour sites \( S_{\mu}^c \subseteq S_{\mu} \) are those sites that touch \( C_{\mu}(\tau) \); the remaining \( S_{\mu}^{nc} = S_{\mu} - S_{\mu}^c \) are possible interior sites. To determine the non redundant contour \( C_{\mu}(\tau) \), it remains to determine the green edges and vertices in \( C_{\mu}(\tau) \).

Quasi-green edges and interior sites. Vertices in \( C_{\mu}(\tau) \) that are on the same bisector are easily identified by sorting all vertices on \( C_{\mu}(\tau) \) according to their incident sites. Two such vertices which are consecutive on a bisector \( b \) inside \( \tau \) are identified by sorting along \( b \). The portion \( e \) of \( b \) between these vertices is called a quasi-edge. \( e \) need not be an edge of \( V_\mu(S) \) as the Voronoi cell of some interior site may intersect it inside \( \tau \). If \( e \) is an edge of \( V_\mu(S) \), then it is green. The vertices of a quasi-green edge are also called quasi-green. Thus, identifying green from quasi-green vertices in \( C_{\mu}(\tau) \) is not immediate. There is also difficulty in computing interior sites because we do not have a test based directly on the knowledge of \( C_{\mu}(\tau) \). Fortunately, as discussed below, the Voronoi diagram of the contour sites resolves both difficulties. The corresponding approach for half-space intersections in \( \mathbb{R}^3 \) was used by Reif and Sen [27].

Voronoi diagram of the contour sites. We want to compute \( V(S_{\mu}) \) in time \( O(\log^2 n_{\mu}) \) and \( O(n_{\mu} \log n_{\mu}) \) work in the EREW PRAM. We know that the boundary of \( \tau \) touches the cells of all the sites in \( S_{\mu}^c \), by definition. Thus, if we use on \( S_{\mu}^c \) the same algorithm we are devising, but starting with the decomposition of the plane induced by (the boundary of) \( \tau \), then interior sites will never appear. Therefore, that this algorithm computes the diagram \( V(S_{\mu}^c) \) within the required bounds will follow from the analysis of the overall algorithm. Note that for our purposes we actually need \( V(S_{\mu}^c) \) both inside and outside of \( \tau \).

Dobkin-Kirkpatrick (D-K) hierarchy. Let \( T = S_{\mu}^c \). A D-K hierarchy [13] for \( V(T) \) is a sequence \( V(T_i) \), \( i = 0, \ldots, k \), where \( T_0 = T, T_i \subseteq T_{i+1}, T_k = O(1) \) and \( T_{i+1} = O(\log T_i) \) for some constant \( 0 < \alpha < 1 \) so that \( k = O(\log T) \). \( T_{i+1} \) is obtained from \( T_i \) by removing sites whose cells are not adjacent and each with a number of adjacent cells \( O(1) \). That this is possible follows from the planarity of the graph of \( V(T) \). The D-K hierarchy can be computed easily in time \( O(T) \) sequentially, and time \( O(\log^2 T) \) and work \( O(T \log T) \) in the EREW PRAM (better performance is possible but not needed here).

Green edges and interior sites. Let \( p \in S_{\mu}^{nc} \). The D-K hierarchy is used to detect whether the cell \( V_\mu(S_{\mu}^c \cup \{p\}) \) is nonempty and if so whether it is interior to \( \tau \). This is done by locating a vertex in \( V(S_{\mu}^c \cup \{p\}) \) that is incident to \( V_\mu(S_{\mu}^c \cup \{p\}) \) if it exists: start by locating such a vertex \( v_k \) in \( V(T_k \cup \{p\}) \) in time \( O(1) \) if it exists, else halt with failure; from \( v_{i+1} \) in \( V(T_{i+1} \cup \{p\}) \) one can obtain \( v_i \) in \( V(T_i \cup \{p\}) \) in time \( O(1) \) if it exists by the properties of the D-K decomposition (either \( v_i = v_{i+1} \) because \( v_{i+1} \) is already a vertex in \( V(T_i \cup \{p\}) \), or only \( O(1) \) edges need to be checked), else halt with failure. So in time \( O(\log T) \) the vertex is obtained if it exists. If the vertex exists and is inside \( \tau \), then \( p \) is interior to \( \tau \). A quasi-edge \( e \) is an edge of \( V_\mu(S) \) (i.e., a green edge), and the corresponding vertices on the contour are green, iff none of the vertices witnessing interior sites for \( \tau \) lie on \( e \).

Attaching edges and point location structure for \( D(\tau) \). Once green edges inside \( \tau \) are determined, green faces inside and green edges on the contour are easily determined. Thus, \( C_{\mu}(\tau) \) has been determined. Then, the attaching edges are also easily determined from \( C_{\mu}(\tau) \). The structure of the subdivision \( D(\tau) \) is particularly simple, since it has no vertices inside; specifically, the dual is a tree. A data structure for point location in \( D(\tau) \) with the required performance can be constructed using standard techniques.

Conclusion. This completes the description of the al-
algorithm, and we state the result in the following.

**Theorem 5** An algebraic planar Voronoi diagram of \( n \) sites can be computed deterministically in the EREW PRAM model using time \( O(\log^2 n) \) and optimal work \( O(\log n) \).

### 3.3 Applications

The result for Voronoi diagrams of line segments in the plane follows directly. The result for ball intersections follows either by a projection onto the plane, or by doing everything on a 2-sphere. In the final version, we plan to include a more complete list of applications. The work of Alt and Schwarzkopf [1] seems relevant to broaden the scope of our algorithm.

### References


