Convex Hull and Voronoi Diagram of Additively Weighted Points

Jean-Daniel Boissonnat and Christophe Delage

INRIA Sophia-Antipolis, 2004, route des lucioles,
06902 Sophia-Antipolis cedex, France
{Jean-Daniel.Boissonnat, Christophe.Delage}@sophia.inria.fr
http://www-sop.inria.fr/

Abstract. We provide a complete description of dynamic algorithms for constructing convex hulls and Voronoi diagrams of additively weighted points of $\mathbb{R}^d$. We present simple algorithms and provide a description of the predicates. The algorithms have been implemented in $\mathbb{R}^3$ and experimental results are reported. Our implementation follows the CGAL design and, in particular, is made both robust and efficient through the use of filtered exact arithmetic.

1 Introduction

In this paper, we provide a complete description of dynamic algorithms for constructing convex hulls and Voronoi diagrams of additively weighted points of $\mathbb{R}^d$. The algorithms have been implemented in $\mathbb{R}^3$ and experimental results are reported.

Our motivation comes from the fact that weighted points can be considered as hyperspheres when the weights are positive and is twofold. On one hand, spheres are non linear objects and, besides the combinatorial and algorithmic questions, numerical and robustness issues deserve a careful investigation, which has not been fully done yet. On the other hand, spheres are objects of major concern in various fields, most notably structural biology, and effective implementations of basic geometric algorithms for spheres are needed.

We first revisit the problem of computing the convex hull of n weighted points of $\mathbb{R}^d$. This problem has already been solved optimally [1,2]. We present a simpler fully dynamic algorithm and provide a complete description of all the predicates for any d.

We then consider the construction of additively weighted Voronoi diagrams. It is known that the construction of such diagrams reduces to intersecting a power diagram in $\mathbb{R}^{d+1}$ with half-cones [3]. We are not aware of robust implementations of this algorithm. Other algorithms have been recently designed and implemented in the planar case [4,5]. In $\mathbb{R}^3$, we are only aware of two prototype implementations, one by Will [6] for computing a single cell, and one by Kim et al. [7] that computes the entire diagram. None of these implementations is provably robust. Moreover, the algorithm by Kim et al. assumes that the graph of the edges of each cell is connected, which is not true in general.

We apply our result on the construction of the convex hull of additively weighted points to the construction of a Voronoi cell in the Voronoi diagram of n additively weighted points. The construction, which makes use of inversion, is close to the algorithm of [2]. The main contribution of this work is to provide a full analysis of the predicates involved, a thorough treatment of the degenerate cases, and a CGAL implementation. Our predicates, when specialized to the planar case ($d = 2$), are simpler and of lower degree than the best predicates known so far [8,9]. Our implementation follows the CGAL design and, in particular, is made both robust and efficient through the use of filtered exact arithmetic.

The paper is organized as follows. In section 2, we establish a new correspondence between convex hulls of additively weighted points in $\mathbb{R}^d$ and power diagrams of spheres of $\mathbb{R}^d$, from which we deduce an algorithm to construct such hulls. In section 3, we recall a similar correspondence for a cell in the Voronoi diagram of additively weighted points and present an algorithm for constructing such a cell. In section 4, we describe the predicates. In section 5, we show how to handle the degenerate cases. In section 6, we report on experimental results. Finally, we conclude in section 7.

In the sequel, a weighted point, or *site* for short, of $\mathbb{R}^d$ is a pair $s = (p, w)$ where p is a point of $\mathbb{R}^d$, and w is a real number, we refer to p and w as the *center* and the *weight* of the site, respectively. When w is positive, we also call a weighted point a hypersphere. Given a set n hyperspheres $\Sigma = \{\sigma_1, \ldots, \sigma_n\}$ of $\mathbb{R}^d$, $\sigma_i = (c_i, r_i)$, the *power* of a point $x \in \mathbb{R}^d$ to σ_i is $\mathrm{d}_P(\sigma_i, x) = (x - c_i)^2 - r_i^2$ and the *power diagram* of Σ, noted $\mathcal{P}(\Sigma)$, is the subdivision of $\mathbb{R}^d$ consisting of the n cells $P(\sigma_1), \ldots, P(\sigma_n)$ where $P(\sigma_i) = \{x \in \mathbb{R}^d, \mathrm{d}_P(\sigma_i, x) \leqslant \mathrm{d}_P(\sigma_j, x), j = 1, \ldots, n\}$. We write $P(\sigma_1, \ldots, \sigma_k) = P(\sigma_1) \cap \ldots \cap P(\sigma_k)$. When non empty, $P(\sigma_1, \ldots, \sigma_k)$ is a face of $\mathcal{P}(\Sigma)$, of dimension $d - k + 1$ if the hyperspheres are in general position.

2 Convex Hull of Additively Weighted Points

Let $\mathcal{S} = \{s_1, \ldots, s_n\}$ be a set of weighted points of $\mathbb{R}^d$. We write $s_i = (p_i, w_i)$, $i = 0, \ldots, n$. We consider first the case where all the weights w_i are non negative, i.e. the sites are hyperspheres. The *convex hull* of $\mathcal{S}$, $\mathrm{CH}(\mathcal{S})$, is the smallest closed convex subset of $\mathbb{R}^d$ containing all the hyperspheres of $\mathcal{S}$. A *supporting hyperplane* H of $\mathcal{S}$ is a hyperplane tangent to at least one of the hyperspheres of $\mathcal{S}$, and such that all the hyperspheres of $\mathcal{S}$ lie in the same half-space limited by H. A *facet of* $\mathrm{CH}(\mathcal{S})$ *of circularity* k, $0 \leqslant k < d$, is the portion of $\partial\mathrm{CH}(\mathcal{S})$ that consists of the points whose supporting hyperplanes are tangent to a same subset of $d - k$ hyperspheres. For $d = 3$, faces of circularity 0 are planar faces tangent to three hyperspheres, faces of circularity 1 are conical patches tangent to two hyperspheres, and faces of circularity 2 are spherical patches contained in some s_i.

From Power Diagrams to Convex Hulls of Hyperspheres. Let Π be a supporting hyperplane tangent to k hyperspheres, and m be the unit normal vector of Π pointing away from $\mathcal{S}$. As there is exactly one supporting hyperplane that has a given oriented normal, m defines Π uniquely. Π is a supporting hyperplane tangent to $s_1, \ldots, s_k$ if and only if:

$$m \cdot (p_i - p_1) = w_1 - w_i,\ 1 \leqslant i \leqslant k \tag{1}$$

$$m \cdot (p_i - p_1) < w_1 - w_i,\ k < i \leqslant n \tag{2}$$

We rewrite (2) as follows:

$$\begin{array}{rl} & m \cdot (p_i - p_1) < w_1 - w_i \\ \Longleftrightarrow & -m \cdot p_1 - w_1 < -m \cdot p_i - w_i \\ \Longleftrightarrow & (m - p_1)^2 - (p_1^2 + 2w_1) < (m - p_i)^2 - (p_i^2 + 2w_i) \ , \end{array}$$

(1) can be rewritten the same way. Thus, denoting $r_i^2 = p_i^2 + 2w_i$, $\sigma_i = (p_i, r_i)$ and $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$, this is equivalent to m being in the open $(d-k+1)$-face of $P(\sigma_1, \ldots, \sigma_k)$ in the power diagram $\mathcal{P}(\Sigma)$. As m belongs to the unit hypersphere $\mathbb{S} = \{x \in \mathbb{R}^d : \|x\| = 1\}$, we have proven

Lemma 1. *The k-faces of $\mathcal{P}(\Sigma) \cap \mathbb{S}$ are in 1-1 correspondence with the facets of circularity k of $\partial\mathrm{CH}(\mathcal{S})$.*

The above construction works also when some or all w_i are negative. Although a geometric interpretation in terms of convex hull is then missing, the result of our construction is called the convex hull of the weighted points, or AWCH for short, by analogy to the case of positive weights.

We now present a static algorithm and an incremental algorithm for contrsucting a AWCH. The affine hull of a face f is denoted aff(f). We say that a k-face f' is a *sub-face* of a $(k+1)$-face f (and conversely that f is a *super-face* of f') when $f' \subseteq f$. For a face f and a sub-face f' of f, $H(f, f')$ denotes the halfspace of aff(f) bounded by aff(f') that contains f. For instance, when f is a 1-face (a line segment), f' is one of its endpoints, and $H(f, f')$ is the ray issued from f' that contains f. We assume that the s_i are in general position so that we do not have any degeneracy. Degeneracies will be considered in section 5.

Static Algorithm. The algorithm first constructs the power diagram $\mathcal{P}(\Sigma)$ and then determines, for each face f of $\mathcal{P}(\Sigma)$, whether f intersects $\mathbb{S}$ or not. The result is stored in *tag*$[f]$:

- *tag*$[f] = \varnothing$ if and only if aff(f) is outside $\mathbb{S}$,
- *tag*$[f] = \ominus$ if and only if f does not intersect $\mathbb{S}$ but aff(f) intersects $\mathbb{S}$,
- *tag*$[f] = \oplus$ if and only if f intersects $\mathbb{S}$,
- *tag*$[f] = \odot$ if and only if f is inside $\mathbb{S}$.

Assuming we know *tag*$[f']$ for each sub-face f' of f, we compute *tag*$[f]$ as follows:

1. If aff(f) does not intersect $\mathbb{S}$, then *tag*$[f] = \varnothing$,
2. else, if for each sub-face f' of f, *tag*$[f'] = \odot$, then, by convexity of f, *tag*$[f] = \odot$,

3. else, if there is a sub-face f' of f such that $tag[f'] = \odot$ or $tag[f'] = \oplus$, then, by connexity of f, $tag[f] = \oplus$.
4. else, if for each sub-face f' of f, $tag[f'] = \varnothing$, and $\mathrm{aff}(f) \cap \mathbb{S} \subseteq H(f, f')$, then f intersects $\mathbb{S}$ and $tag[f] = \oplus$,
5. else, $tag[f] = \ominus$.

Assume w.l.o.g. that $f = P(\sigma_1, \dots, \sigma_k)$ and $f' = P(\sigma_1, \dots, \sigma_{k+1})$. This algorithm needs to evaluate the two following geometric predicates.

k-RadicalIntersection(f) determines whether $\mathrm{aff}(f)$ is outside $\mathbb{S}$ or not.
k-RadicalSide(f, f') determines whether $\mathrm{aff}(f) \cap \mathbb{S} \subset H(f, f')$, assuming that $\mathrm{aff}(f)$ intersects $\mathbb{S}$ and $\mathrm{aff}(f')$ is outside $\mathbb{S}$.

To computes the tags, we proceed by induction on the dimension of the faces. This takes a time proportional to the size of the diagram. It follows that the time complexity of the algorithm is upbounded by the time complexity of a power diagram algorithm. Hence, our algorithm computes the additively weighted convex hull of n sites in $\mathcal{O}\left(n \log n + n^{\lceil \frac{d}{2} \rceil}\right)$ time.

Incremental Algorithm. We now present an incremental alogrithm for computing the additively weighted convex hull. We use an incremental algorithm for constructing the power diagram, and we attach to each face f the number $num[f]$ of its sub-faces that are tagged $\oplus$. Updating the power diagram when inserting a new hypersphere in the power diagram amounts to creating some new faces, deleting some faces, and replacing some faces by smaller ones. In this last case, a face f is replaced by a smaller face $\bar{f} \subset f$ that is incident to the cell of the new hypersphere. $\bar{f}$ is called a *cut face*. Notice that a cut face of dimension less than d has exactly one new subface. We denote by m the number of deleted faces plus the number of new faces.

1. For a new face f, $num[f]$ is easily computed by looking at all its sub-faces. This can be done in time proportional to m.
2. We now update $num[\bar{f}]$ for a cut face $\bar{f}$. We set $num[\bar{f}] = num[f]$. Then $num[\bar{f}]$ is decremented by the number of the sub-faces of f that are deleted, and updated according to the tag of the new and cut sub-faces of $\bar{f}$. This can be done in the following way. When the tag of a face f' changes, or f' is deleted, we update $num[f]$ for each *super*-face f of f'. Updating $num[\bar{f}]$ therefore takes $\mathcal{O}(m)$ time.
3. We update $tag[\bar{f}]$ for a cut k-face $\bar{f}$, $k > 1$. We compute $tag[\bar{f}]$ from $num[\bar{f}]$ and $tag[f']$, where f' is the new sub-face f' of $\bar{f}$. As $\bar{f}$ is a cut face, $tag[\bar{f}]$ differs from $tag[f]$ only when $tag[f] = \oplus$. If $num[\bar{f}]$ is positive, then $tag[\bar{f}] = \oplus$. Now, if $num[\bar{f}]$ is 0, only the relative interior of $\bar{f}$ can intersect $\mathbb{S}$. Hence
   – if $tag[f'] = \varnothing$, then we update $tag[\bar{f}]$ according to the outcome of k-RadicalSide($\bar{f}, f'$)
   – otherwise, as the set of the sub-faces of $\bar{f}$ is connected ($\bar{f}$ is of dimension at least 2), $\bar{f}$ has the same tag as f'.

   For a cut 1-face $\bar{f}$, we compute $tag[\bar{f}]$ directly. Updating the tag of a cut face takes a constant time.

The incremental algorithm for constructing an additively weighted convex hull has therefore the same complexity as the incremental algorithm that computes the associated power diagram. When the sites are inserted in random order, the expected time complexity of our algorithm is therefore $\mathcal{O}\left(n \log n + n^{\lceil \frac{d}{2} \rceil}\right)$.

Practical Complexity. Under realistic assumptions, our algorithms perfom better than in the worst case. First, according to our experiments (see section 6), the number of hyperspheres with a non empty cell in the power diagram is usually proportional to the number of points h on the additively weighted convex hull. In that case, the running time for n insertions is $\mathcal{O}\left(n \log h + h^{\lceil \frac{d}{2} \rceil}\right)$. Moreover, h is typically much smaller than n. It is known that the convex hull of a set of n points uniformly distributed inside a sphere of $\mathbb{R}^3$ has $\mathcal{O}(\sqrt{n})$ points on its convex hull. The same result holds trivially for spheres with the same radius. In $\mathbb{R}^3$, assuming that the number of cells in the power diagram is proportional to h and that h is $\mathcal{O}(\sqrt{n})$, the complexity of our algorithm is $\mathcal{O}(n \log n)$.

3 Additively Weighted Voronoi Diagram

The *additively weighted distance*, denoted d_+, from a point m of $\mathbb{R}^d$ to a site $s_i = (p_i, w_i)$ is $\mathrm{d}_+(s_i, m) = \|p_i - m\| - w_i$. Considering the set $\mathcal{S} = \{s_1, \dots, s_n\}$ of sites, the *additively weighted Voronoi cell* of s_i, $V(s_i)$ is:

$$V(s_i) = \left\{m \in \mathbb{R}^d \mid \forall j,\ \mathrm{d}_+(s_i, m) \leqslant \mathrm{d}_+(s_j, m)\right\} \ .$$

It is possible that $V(s_i) = \varnothing$: this happens when $\forall m \in \mathbb{R}^d,\ \exists j,\ \mathrm{d}_+(s_j, m) \leqslant \mathrm{d}_+(s_i, m)$. In that case, we say that s_i is *hidden by* s_j. When dealing with hyperspheres (*i.e.* $w_i, w_j > 0$), s_i is hidden by s_j when $s_i \subseteq s_j$. The *additively weighted Voronoi diagram*, or AWVD for short, of $\mathcal{S}$, noted $\mathcal{V}(\mathcal{S})$, is the cell complex whose d-cells are the $V(s_i)$.

The construction of a single cell of the diagram reduces to computing an additively weighted convex hull, via an inversion. More precisely, the cell of s_1 in $\mathcal{V}(\mathcal{S})$ is combinatorially equivalent to the additively weighted convex hull of $\mathcal{S}' = \{s'_1, \dots, s'_n\}$, where s'_1 is centered at the origin and has weight 0, s'_i is centered at $\frac{p_i - p_1}{\alpha_i}$ and has weight $\frac{w_i - w_1}{\alpha_i}$, $\alpha_i = (p_i - p_1)^2 - (w_i - w_1)^2$, $i = 2 \dots n$. This scheme only works when s_1 is not hidden by, nor does it hide any other site s_i. See [2] for details.

Using the construction of a single cell as a subroutine, we can compute the whole diagram in a fully dynamic manner. All the Voronoi cells are stored in a data structure with pointers between the corresponding elements. In this data structure, $\Gamma(s)$ denotes the set of neighbors of s and $\Gamma_s(s')$ the set of neighbors of s that are also neighbors of s'. Two sites s and s' are called *neighbors* if $V(s)$ and $V(s')$ share a $(d-1)$-face. We add to the data structure an *infinite cell*, which is actually the plain additively weighted convex hull of the sites. That way, we handle unbounded faces, and sites lying on the convex hull seamlessly.

In order to keep track of the hidden sites, we keep, for each site s, a list *hidden*$[s]$ of the sites s hides. We now describe the three main ingredients needed to update an additively weighted Voronoi diagram.

Localization. Given a point m of $\mathbb{R}^d$ and a starting site s, this procedure, called LOCATE(m, s), returns the cell of the diagram in which m lies. This is done by means of a simple walk: if a neighbor s' is closer to m than s, jump to s' and we iterate; otherwise, m belongs to the cell of s and we stop. This localization algorithm requires only one predicate: given two sites s_1 and s_2, determine if a query point m is closer to s_1 than to s_2. This predicate is called SIDEOFBISECTOR.

Insertion. The insertion procedure needs to decide whether a site s_1 hides another site s_2 : we call this predicate ISTRIVIAL(s_1, s_2). To avoid ambiguities when considering diagrams of different sets of sites, we introduce the following notation. Given a set of sites $\mathcal{S}$ and $s, s' \in \mathcal{S}$, we denote $V_{\mathcal{S}}(s)$ the cell of s in the additively weighted Voronoi diagram of $\mathcal{S}$, and $V_{\mathcal{S}}(s, s') = V_{\mathcal{S}}(s) \cap V_{\mathcal{S}}(s')$. A site $s \notin \mathcal{S}$ is *in conflict with* $s' \in \mathcal{S}$ if and only if $V_{\mathcal{S}}(s') \neq V_{\mathcal{S}\cup\{s\}}(s')$. Notice that only the non-hidden sites of $\mathcal{S}$ may be in conflict with s. The *conflict graph* of a site $s \notin \mathcal{S}$ is $G = (X, E)$ where $X \subseteq \mathcal{S}$ is the set of sites in conflict with s, and $xy \in E$ if and only if $V_{\mathcal{S}}(x, y) \cap V_{\mathcal{S}\cup\{s\}}(s) \neq \varnothing$. In other words, the conflict graph of s is the dual of the restriction of $\mathcal{V}(\mathcal{S})$ to $V_{\mathcal{S}\cup\{s\}}(s)$.

Lemma 2. *The conflict graph of s is connected.*

Proof. Given two sites x and y in the conflict graph G of some s, we take p_x in $V_{\mathcal{S}}(x) \cap V_{\mathcal{S}\cup\{s\}}(s)$ and p_y in $V_{\mathcal{S}}(y) \cap V_{\mathcal{S}\cup\{s\}}(s)$. As $V_{\mathcal{S}\cup\{s\}}(s)$ is arc connected, we can follow a path from p_x to p_y in $V_{\mathcal{S}\cup\{s\}}(s)$, and each time we cross a $(d-1)$-face of the diagram, we follow the corresponding edge of G. This gives a path from x to y in G. ☐

The first insertion (*i.e.* the insertion in an empty diagram) is easy: we just create a new cell covering all $\mathbb{R}^d$. Once there is a least one site in the diagram, the insertion procedure of a new site s is the following.

1. Locate the center of s, let s' be the site such that the center of s lies in $V(s')$.
2. If s is hidden by s', then add s to *hidden*$[s']$.
3. Else, s' is a vertex in the conflict graph G of s, so we walk on G, starting from s', and for each s'' in G:
   - if s'' is hidden by s, add s'' to *hidden*$[s]$,
   - else, insert s in $V(s'')$ and s'' in $V(s)$.

Removal. Removing a site s from a diagram is straightforward. Firstly, remove s from the cells adjacent to $V_+(s)$, Secondly, let $\{s_1, \ldots, s_k\}$ be the neighbors of s; for all $1 \leqslant i, j \leqslant k$, $i \neq j$, insert s_i into the cell of s_j to rebuild the hole made by the removal of s. And finally, insert all the sites s was hiding.

Complexity. The localization algorithm described here takes time linear in the size of the diagram, which can be improved to randomized logarithmic time by using a hierarchical data structure as in [4].

The construction of the Voronoi diagram of n sites performs $\mathcal{O}(n)$ localizations, and constructs $\mathcal{O}(n)$ additively weighted convex hulls of $\mathcal{O}(n)$ sites. The overall time to construct the Voronoi diagram of n sites is therefore:

$$\mathcal{O}\left(n \log n + n\left(n \log n + n^{\lceil \frac{d}{2} \rceil}\right)\right) = \mathcal{O}\left(n^2 \log n + n^{\lceil \frac{d}{2} \rceil + 1}\right) \ .$$

Which gives, for $d = 3$, $\mathcal{O}\left(n^3\right)$. This bound can be improved under the following assumptions:

1. $\mathcal{O}(\sqrt{n})$ sites appear on the additively weighted convex hull of $\mathcal{S}$,
2. the sites have $\mathcal{O}(1)$ neighbors,
3. the underlying power diagram of every Voronoi cell has $\mathcal{O}(s)$ non-hidden points, where s is the number of neighbors of the cell.

Those assumptions are not too restrictive, and happen to be satisfied on a variety of input data (see section 6.) Assumptions 1 and 3 implies that the construction of the infinite cell (*i.e.* the AWCH) take $\mathcal{O}(n \log n)$ time. Assumptions 2 and 3 implies that we construct $\mathcal{O}(n)$ finite cells of size $\mathcal{O}(1)$, and that it takes $\mathcal{O}(n)$ time. This leads to an expected running time of $\mathcal{O}(n \log n)$, for constructing the additively weighted Voronoi diagram.

4 Predicates

We consider a set $\mathcal{S} = \{s_1, \ldots, s_n\}$ of sites, where s_i is centered at p_i, and has weight w_i. If each input data is a b-bit integer, the size of each monomial occuring in a predicate is upper bounded by $2^{(b+1)d}$. Moreover, let v be the number of variables that occur in a predicate; for the predicates considered in this

Table 1. Predicate degree summary

Algorithm	AWCH			AWVD		
Dimension	2	3	$d > 3$	2	3	$d > 3$
IsTrivial				2	2	2
SideOfBisector				4	4	4
Orientation	2	3	d	4	5	$d+2$
PowerTest	3	4	$d+1$	5	6	$d+3$
1-RadicalIntersection	2	2	2	6	6	6
2-RadicalIntersection	4	8	8	8	16	16
3-RadicalIntersection		6	12		10	20
k-RadicalIntersection, $1 < k < d$			$4k$			$4k+8$
d-RadicalIntersection			$2d$			$2d+4$
1-RadicalSide	1	1	1	3	3	3
2-RadicalSide	3	3	3	7	7	7
3-RadicalSide		5	5		9	9
k-RadicalSide, $1 < k \leqslant d$			$2k-1$			$2k+3$
Maximum degree	4	8	$4d-4$	8	16	$4d+4$

paper, v is a constant. It follows that a predicate of degree d requires precision $p \leqslant d(b+1+\log v)$. Here, the predicates are polynomials in the unknowns p_i, w_i, and the algebraic degree of each of them is given. In addition to the predicates mentioned in section 2 and 3, we need the two well-known predicates that are needed to construct the power diagram: Orientation and PowerTest.

Predicates IsTrivial and SideOfBisector are detailed in [10] for $d = 2$, and are straightforward to extend to arbitary dimension. IsTrivial is of degree 2, and SideOfBisector is of degree 4. Basic linear algebra provides explicit formulas for the other predicates. The maximum degree of the predicates for the AWCH is 4 in 2D, 8 in 3D, and in general, $4d-4$ in dimension d. The maximum degree of the predicates for the AWVD is 8 in 2D, 16 in 3D, and in general, $4d+4$ in dimension d. See Table 1. This compares very well to the predicates of the algorithm for the additively weighted Voronoi diagram of [4], detailed in [8], which have a maximal degree of 16 for $d = 2$.

5 Degenerate Cases

Additively Weighted Convex Hull. Here, we show how to handle degeneracies in the algorithm for the convex hull of additively weighted points. We call a case *degenerate* when some predicate returns 0, instead of "positive" or "negative". A simple way of dealing with these cases is to carefully choose a non-zero sign to be returned by a predicate when it evaluates to 0.

– Predicates Orientation or PowerTest return zero when Σ is not in general position. Any standard perturbation scheme will work for us.
– When predicate k-RadicalIntersection returns 0, some $(d-k)$-flat is tangent to $\mathbb{S}$ (it intersects $\mathbb{S}$ but not the open ball bounded by $\mathbb{S}$.) We can consider that this $(d-k)$-flat lies outside $\mathbb{S}$.
– Predicate k-RadicalSide(f, f') returns 0 when the projection of the origin on aff(f) is on aff(f'). In that case, both aff(f) and aff(f') intersects $\mathbb{S}$, or both do not intersect $\mathbb{S}$. As predicate k-RadicalSide is only called when aff(f) intersects $\mathbb{S}$ and aff(f') does not, this predicate is never called on degenerate inputs.

Additively Weighted Voronoi Diagram. In the case of the additively weighted Voronoi diagram, the previous perturbation scheme does not work. Indeed, as we compute the Voronoi cells separately, we not only need to resolve degeneracies in each cell, but also to ensure that consistent decisions are taken when we compute the neighboring cells. A set of sites $\mathcal{S}$ is called *degenerate* when there exists $k+1$ sites of $\mathcal{S}$ $s_0, \dots, s_k$, $1 \leqslant k < d$, such that $V(s_0, \dots, s_k)$ is not empty and is of dimension strictly less than $d-k$. When an input is non-degenerate, any small enough perturbation will let the combinatorial structure of the Voronoi diagram unchanged. For a face f, we define $L(f) = \{s \in S \ : \ f \subseteq V(s)\}$. If a degeneracy $\{s_0, \dots, s_k\}$ is minimal (*i.e.* if $\{s_0, \dots, s_k\} \setminus \{s_i\}$ is not degenerate for $0 \leqslant i \leqslant k$) then perturbing the weight of any site in $L(V(s_0, \dots, s_k)$ will remove the degeneracy.

Given a degenerate input $\{s_0, \ldots, s_k\}$, finding a minimal degeneracy is easy: w.l.o.g. we check if $\{s_0, \ldots, s_{k-1}\}$ is still degenerate. As $V(s_0, \ldots, s_k) \neq \varnothing$, $V(s_0, \ldots, s_{k-1})$ has dimension at least 1. A degeneracy of dimension $m \geqslant 1$ in the intersection between the unit hypersphere and a power diagram can only appear if two $(m+1)$-faces of the power diagram are equal, which can be tested with the Orientation and PowerTest predicates.

Now, we can handle the degeneracies in our algorithm by means of symbolic perturbations. When faced with a predicate that returns zero on $\{s_0, \ldots, s_k\}$, we find a minimal degeneracy $\{s_0, \ldots, s_{k'}\}$ and perturb one site, in $L(V(s_0, \ldots, s_{k'}))$, say s_i, *i.e.* we replace w_i by $w_i + \epsilon$. The predicates are now polynomials in ϵ and we need to evaluate the sign of the non-zero coefficient of smallest degree. Notice that this scheme does not increase the degree of the predicates since the perturbation is linear. It can occur that some predicate returns zero, and the Voronoi diagram is not degenerate, and thus some site gets perturbed unnecessarily.

To ensure consistency from one cell to another, we just need to choose the site to perturb in a way that is independent of the site whose cell is under construction when we detect the degeneracy. One way to do that is to choose the smallest site according to some global ordering (for instance, the lexicographical order on the centers.)

6 Experimental Results

We have implemented both our algorithms for constructing the convex hull and the Voronoi diagram of weighted points in $\mathbb{R}^3$. The implementations use CGAL 3.1, mainly its 3D regular triangulations (see [11]), which are the duals of the power diagrams. While not yet fully optimized, this implementation already follows the CGAL standard of genericity and robustness. The predicates are dynamically filtered to avoid problems of precision in degenerate, or near-degenerate, cases. We plan to have the code included in the CGAL library soon. The running times are obtained on a Athlon running at 1333MHz, with 133MHz DDR-SDRAM memory and 256KB of L2 cache.

On Fig. 1, the degenerate input is a set of sites randomly chosen in a cube, with their weight equal to their height, so that all the sites are tangent to the lower face of the cube and the non-degenerate input is a set of sites uniformly distributed inside a sphere, the weights uniformly distributed in an interval.

On Fig. 2, the input comes from a direct application of our algorithm. The sites have their centers on a surface and the weights are of the form $-\frac{\mathrm{lfs}(x)}{k}$ where $\mathrm{lfs}(x)$ is an approximation of the local feature size of the surface at x, and k is a parameter. This kind of diagram has been used to efficiently compute a sizing field, for 3D meshing (see [12] for details). On Fig. 2, $k = 1$ on the left and $k = 0.3$ on the right.

Both algorithms are incremental, and as such, their running time is likely to depend on the insertion order. Fig. 1 and 2 show three insertion orders: sites with small weights first, sites with large weights first, and random. In all cases, the

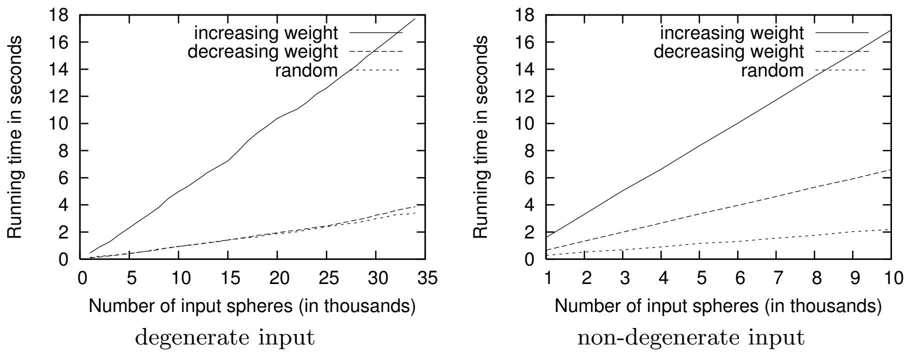

Fig. 1. Additively Weighted convex hull benchmarks, all using filtered predicates, for various input, and insertion order

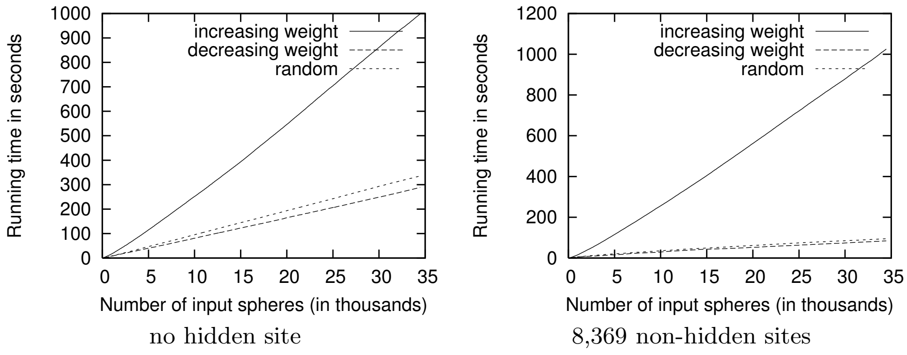

Fig. 2. Additively weighted Voronoi diagram benchmarks using filtered predicates for various input sizes, numbers of hidden sites, and insertion orders

algorithms are faster when the sites are inserted in order of decreasing weights. The reason is that a site with a larger weight tends to have more neighbors, and thus, tends to take longer to insert. The difference is even greater when there are many hidden spheres.

A screenshot is shown in Figure 3, where one cell is represented by meshing its boundary. Our implementation computes all the edges of the cell (*i.e.* facets of circularity 1 in the underlying convex hull), and sample them. Then, each face of the cell (*i.e.* each facet of circularity 2 in the convex hull) is approximated using the meshing algorithm of [13]. We plan to have the code included in the CGAL library soon.

In our experiments, we observed a remarkable phenomenon: almost all the spheres that do not contribute to the additively weighted convex hull are hidden (*i.e.* have empty cells) in the underlying power diagram. This also occur when the AWCH are cells of an additively weighted Voronoi diagram. In no Voronoi cell the examples shown here, the number of cells in the power diagram is more

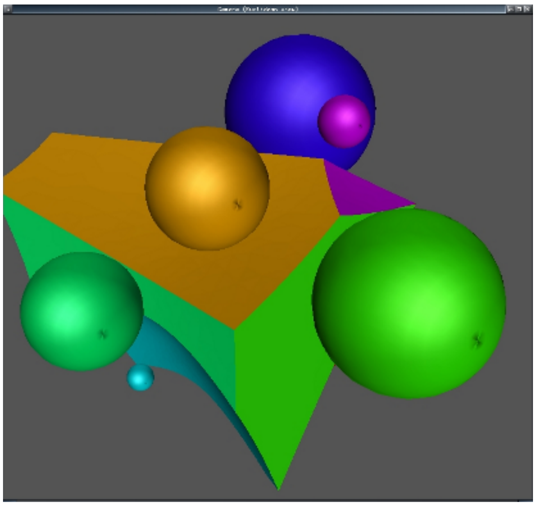

Fig. 3. A screenshot of one cell of a 3D additively weighted Voronoi diagram

than seven times the number of neighbors of the Voronoi cell. Moreover, for only 1% of the Voronoi cells, the number of cells in the underlying power diagram is more than twice the number of neighbors in the Voronoi diagram. Although this observation does not hold in general —it is possible to construct n spheres such that only $\mathcal{O}(1)$ of them contribute to the convex hull, while all of them appear in the power diagram— this makes our algorithms efficient in practice.

7 Conclusion

We have presented fully robust implementations of two algorithms for constructing the convex hull and the Voronoi diagram of additively weighted points (and hyperspheres). To the best of our knowledge, no certified algorithms existed previously.

This work does not settle the main open question in this area : what is the combinatorial complexity of the Voronoi diagram of n additively weighted points in $\mathbb{R}^d$? Tight bounds are only known for $d = 2$ and odd dimensions. We hope that experimenting with our code may provide new insights such as the one mentionned in section 6 that eventually will help improving the combinatorial bounds.

References

1. Boissonnat, J.D., Cérézo, A., Devillers, O., Duquesne, J., Yvinec, M.: An algorithm for constructing the convex hull of a set of spheres in dimension d. Comput. Geom. Theory Appl. **6** (1996) 123–130

2. Boissonnat, J.D., Karavelas, M.: On the combinatorial complexity of Euclidean Voronoi cells and convex hulls of d-dimensional spheres. In: Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA). (2003) 305–312
3. Aurenhammer, F., Imai, H.: Geometric relations among Voronoi diagrams. Geom. Dedicata **27** (1988) 65–75
4. Karavelas, M., Yvinec, M.: Dynamic additively weighted voronoi diagrams in 2d. In: Proc. 10th European Symposium on Algorithms. (2002) 586–598
5. Kim, D.S., Kim, D., Sugihara, K.: Updating the topology of the dynamic voronoi diagram for spheres in euclidean d-dimensional space. Computer-Aided Design **18** (2001) 541–562
6. Will, H.M.: Fast and efficient computation of additively weighted Voronoi cells for applications in molecular biology. In: Proc. 6th Scand. Workshop Algorithm Theory. Volume 1432 of Lecture Notes Comput. Sci., Springer-Verlag (1998) 310–321
7. Kim, D.S., Cho, Y., Kim, D., Bhak, J., Lee, S.H.: Euclidean voronoi diagram of 3d spheres and applications to protein structure analysis. In Sugihara, K., ed.: 1st International Symposium on Voronoi Diagrams in Science and Engineering. (2004)
8. Karavelas, M.I., Emiris, I.Z.: Root comparison techniques applied to computing the additively weighted Voronoi diagram. In: Proc. 14th ACM-SIAM Sympos. Discrete Algorithms (SODA). (2003) 320–329
9. Anton, F.: Voronoi diagrams of semi-algebraic sets. Ph.d. thesis, University of British Columbia (2004)
10. Karavelas, M.I., Emiris, I.Z.: Predicates for the planar additively weighted Voronoi diagram. Technical Report ECG-TR-122201-01, INRIA Sophia-Antipolis (2002)
11. : The CGAL Manual. (2004) Release 3.1.
12. Alliez, P., Cohen-Steiner, D., Yvinec, M., Desbrun, M.: Variational tetrahedral meshing. In: SIGGRAPH. (2005)
13. Boissonnat, J.D., Oudot, S.: Provably good surface sampling and approximation. In: Proc. 1st Symp. on Geometry Processing. (2003) 9–18