An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams

Martin Held
Universität Salzburg, FB Computerwissenschaften, Austria
held@cs.sbg.ac.at

Stefan de Lorenzo
Universität Salzburg, FB Computerwissenschaften, Austria
slorenzo@cs.sbg.ac.at

Abstract

We present a simple wavefront-like approach for computing multiplicatively weighted Voronoi diagrams of points and straight-line segments in the Euclidean plane. If the input sites may be assumed to be randomly weighted points then the use of a so-called overlay arrangement [Har-Peled&Raichel, Discrete Comput. Geom. 53:547–568, 2015] allows to achieve an expected runtime complexity of $\mathcal{O}(n \log^4 n)$, while still maintaining the simplicity of our approach. We implemented the full algorithm for weighted points as input sites, based on CGAL. The results of an experimental evaluation of our implementation suggest $\mathcal{O}(n \log^2 n)$ as a practical bound on the runtime. Our algorithm can be extended to handle also additive weights in addition to multiplicative weights, and it yields a truly simple $\mathcal{O}(n \log n)$ solution for solving the one-dimensional version of this problem.

2012 ACM Subject Classification Theory of computation → Computational geometry

Keywords and phrases Voronoi Diagram, multiplicative weight, additive weight, arc expansion, overlay arrangement, implementation, experiments, CGAL, exact arithmetic

Digital Object Identifier 10.4230/LIPIcs.ESA.2020.56

Related Version A full version of this paper is available at [9], `https://arxiv.org/abs/2006.14298`.

Supplementary Material The source code of our implementation is available at GitHub and can be used freely under the GPL(v3) license: `https://github.com/cgalab/wevo`.

Funding Work supported by Austrian Science Fund (FWF): Grant P31013-N31.

1 Introduction

The multiplicatively weighted Voronoi diagram (MWVD) was introduced by Boots [4]. Aurenhammer and Edelsbrunner [2] present a worst-case optimal incremental algorithm for constructing the MWVD of a set of n points in $\mathcal{O}(n^2)$ time and space. They define spheres on the bisector circles (that are assumed to be situated in the xy-plane) and convert them into half-planes in $\mathbb{R}^3$ using a spherical inversion. Afterwards, these half-planes are intersected. Thus, every Voronoi region is associated with a polyhedron. Finally, the intersection of every such polyhedron with a sphere that corresponds to the xy-plane is inverted back to $\mathbb{R}^2$. We are not aware of an implementation of their algorithm, though. (And it seems difficult to implement.) In any case, the linear-time repeated searches for weighted nearest points indicate that its complexity is $\Theta(n^2)$ even if the combinatorial complexity of the resulting Voronoi diagram is $o(n^2)$. Later Aurenhammer uses divide&conquer to obtain an $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space algorithm for the one-dimensional weighted Voronoi diagram [1].

Har-Peled and Raichel [8] show that a bound of $\mathcal{O}(n \log^2 n)$ holds on the expected combinatorial complexity of a MWVD if all weights are chosen randomly. They sketch how to compute MWVDs in expected time $\mathcal{O}(n \log^3 n)$. Their approach is also difficult to implement because it uses the algorithm by Aurenhammer and Edelsbrunner [2] as a subroutine.

Vyatkina and Barequet [13] present a wavefront-based strategy to compute the MWVD of a set of n lines in the plane in $\mathcal{O}(n^2 \log n)$ time. The Voronoi nodes are computed based on edge and break-through events. An edge event takes place when an wavefront edge disappears. A break-through event happens whenever a new wavefront edge appears.

Since the pioneering work of Hoff et al. [10] it has been well known that discretized versions of Voronoi diagrams can be computed using the GPU framebuffer. More recently, Bonfiglioli et al. [3] presented a refinement of this rendering-based approach. It is obvious that their approach could also be extended to computing approximate MWVDs. However, the output of such an algorithm is just a set of discrete pixels instead of a continuous skeletal structure. Its precision is limited by the resolution of the framebuffer and by the numerical precision of the depth buffer.

2 Our Contribution

Our basic algorithm allows us to compute MWVDs in worst-case $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space. A refined version makes use of the result by Har-Peled and Raichel [8]: We use their overlay arrangement to keep the expected runtime complexity bounded by $\mathcal{O}(n \log^4 n)$ if the point sites are weighted randomly. Hence, for the price of a multiplicative factor of $\log n$ we get an algorithm that is easier to implement. Our experiments suggest that this bound is too pessimistic in practice and that one can expect the actual runtime to be bounded by $\mathcal{O}(n \log^2 n)$. However, our experiments also show that one may get a quadratic runtime if the weights are not chosen randomly. Our algorithm does not require the input sites to have different multiplicative weights, and it can be extended to additive weights and to (disjoint) straight-line segments as input sites. Furthermore, it yields a truly simple $\mathcal{O}(n \log n)$ solution for computing MWVDs in one dimension, where all input points lie on a line.

Our implementation is based on exact arithmetic and the Computational Geometry Algorithms Library (CGAL) [12]. It is publicly available on GitHub under `https://github.com/cgalab/wevo`. To the best of our knowledge, this is the first full implementation of an algorithm for computing MWVDs that achieves a decent expected runtime complexity.

3 Preliminaries

Let $S := \{s_1, s_2, \ldots, s_n\}$ denote a set of n distinct weighted points in $\mathbb{R}^2$ that are indexed such that $w(s_i) \leq w(s_j)$ for $1 \leq i < j \leq n$, where $w(s_i) \in \mathbb{R}^+$ is the weight associated with s_i. It is common to regard the weighted distance $d_w(p, s_i)$ from an arbitrary point p in $\mathbb{R}^2$ to s_i as the standard Euclidean distance $d(p, s_i)$ from p to s_i divided by the weight of s_i, i.e., $d_w(p, s_i) := \frac{d(p,s_i)}{w(s_i)}$. The *(weighted) Voronoi region* $\mathcal{VR}_w(s_i, S)$ of s_i relative to S is the set of all points of the plane such that no site s_j in $S \setminus \{s_i\}$ is closer to p than s_i, that is, $\mathcal{VR}_w(s_i, S) := \big\{p \in \mathbb{R}^2 : d_w(p, s_i) \leq d_w(p, s_j) \text{ for all } j \in \{1, 2, \ldots, n\}\big\}$. Then the multiplicatively weighted Voronoi diagram (MWVD), $\mathcal{VD}_w(S)$, of S is defined as $\mathcal{VD}_w(S) := \bigcup_{s_i \in S} \partial\, \mathcal{VR}_w(s_i, S)$.

A connected component of a Voronoi region is called a *face*. For two distinct sites s_i and s_j of S, the *bisector* $b_{i,j}$ of s_i and s_j models the set of points of the plane that are at the same weighted distance from s_i and s_j. Hence, a non-empty intersection of two Voronoi regions is a subset of the bisector of the two defining sites. Following common terminology, a connected component of such a set is called a *(Voronoi) edge* of $\mathcal{VD}_w(S)$. An end-point of an edge is called a *(Voronoi) node*. It is known that the bisector between two unequally weighted sites forms a circle[1]. An example of a MWVD is shown in Figure 1.

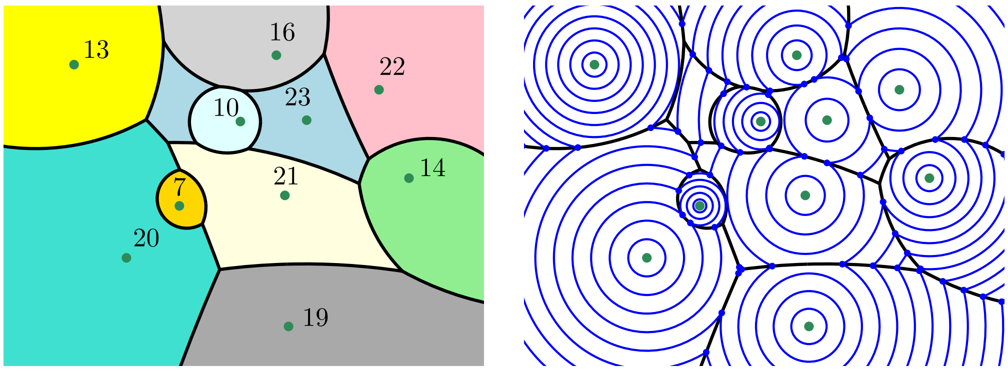


Figure 1 Left: The numbers next to the points indicate their weights and the corresponding MWVD is shown. Right: Wavefronts (in blue) for equally-spaced points in time.

The wavefront $\mathcal{WF}(S,t)$ emanated by S at time $t \geq 0$ is the set of all points p of the plane whose minimal weighted distance from S equals t. More formally,

$$\mathcal{WF}(S,t) := \left\{ p \in \mathbb{R}^2 : \min_{s_i \in S} d_w(p, s_i) = t \right\}.$$

The wavefront consists of circular arcs which we call *wavefront arcs*. A common end-point of two consecutive wavefront arcs is called *wavefront vertex*; see the blue dots in Figure 1.

4 Offset Circles

For the sake of descriptional simplicity, we start with assuming that no point in the plane has the same weighted distance to more than three sites of S. For $t \geq 0$, the *offset circle* $c_i(t)$ of the i-th site s_i is given by a circle centered at s_i with radius $t \cdot w(s_i)$. We find it convenient to regard $c_i(t)$ as a function of either time or distance since at time t every point on $c_i(t)$ is at Euclidean distance $t \cdot w(s_i)$ from s_i, i.e., at weighted distance t. We specify a point of $c_i(t)$ relative to s_i by its polar angle α and its (weighted) polar radius t and denote it by $p_i(\alpha, t)$.

For $1 \leq i < j \leq n$, consider two sites $s_i, s_j \in S$ and assume that $w(s_i) \neq w(s_j)$. Then there exists a unique closed time interval $[t_{ij}^{min}, t_{ij}^{max}]$ during which the respective offset circles of s_i, s_j intersect. We say that the two offset circles *collide* at their mutual *collision time* t_{ij}^{min}, and s_j starts to *dominate* s_i at the domination time t_{ij}^{max}. For all other times t within this interval the two offset circles $c_i(t)$ and $c_j(t)$ intersect in two disjoint points $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$. These *(moving) vertices* trace out the bisector between s_i and s_j; see Figure 2. Since $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ are defined by the same pair of offset circles we refer to $v_{i,j}^l(t)$ as the *vertex married to* $v_{i,j}^r(t)$, and vice versa. Every other pair of moving vertices defined by two different pairs of intersecting offset circles is called *unmarried*. To simplify the notation, we will drop the parameter t if we do not need to refer to a specific time. Similarly, we drop the superscripts l and r if no distinction between married and unmarried vertices is necessary.

[1] Apollonius of Perga defined a circle as a set of points that have a specific distance ratio to two foci.

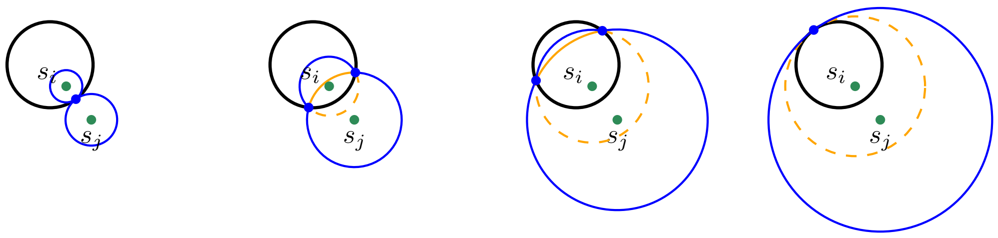


Figure 2 Two married vertices (highlighted by the blue dots) trace out the bisector b_{ij} (in black).

5 A Simple Event-Based Construction Scheme

In this section we describe a simulation of a propagation of the wavefront $\mathcal{WF}(S,t)$ to compute $\mathcal{VD}_w(S)$. Since the wavefront is given by a subset of the arcs of the arrangement of all offset circles, one could attempt to study the evolution of all arcs of that arrangement over time. However, it is sufficient to restrict our attention to a subset of arcs of that arrangement. We note that our wavefront can be seen as a kinetic data structure [7].

Clearly, the arc along $c_i(t)$ which is inside $c_j(t)$ will not belong to $\mathcal{WF}(\{s_i,s_j\},t^*)$ for any $t^* > t$. We will make use of this observation to define *inactive* and *active arcs* that are situated along the offset circles.

▶ **Definition 1** (Active point). *A point p on the offset circle $c_i(t)$ is called* inactive *at time t (relative to S) if there exists $j > i$, with $1 \le i < j \le n$, such that p lies strictly inside of $c_j(t)$. Otherwise, p is* active *(relative to S) at time t. A vertex $v_{i,j}(t)$ is an* active vertex *if it is an active point on both $c_i(t)$ and $c_j(t)$ at time t; otherwise, it is an* inactive vertex.

▶ **Lemma 2.** *If $p_i(\alpha,t)$ is inactive at time t then $p_i(\alpha,t')$ will be inactive for all times $t' \ge t$.*

An inactive point $p_i(\alpha,t)$ cannot be part of the wavefront $\mathcal{WF}(S,t)$. Lemma 2 ensures that none of its future incarnations $p_i(\alpha,t')$ can become part of the wavefront $\mathcal{WF}(S,t')$.

▶ **Definition 3** (Active arc). *For $1 \le i \le n$ and $t \ge 0$, an* active arc *of the offset circle $c_i(t)$ at time t is a maximal connected set of points on $c_i(t)$ that are active at time t. The closure of a maximal connected set of inactive points of $c_i(t)$ forms an* inactive arc *of $c_i(t)$ at time t.*

Every end-point of an active arc of $c_i(t)$ is given by the intersection of $c_i(t)$ with some other offset circle $c_j(t)$, i.e., by a moving vertex $v_{i,j}(t)$. This vertex is active, too.

▶ **Definition 4** (Arc arrangement). *The* arc arrangement (AA) *of S at time t, $\mathcal{A}(S,t)$, is the arrangement induced by all active arcs of all offset circles of S at time t.*

As time t increases, the offset circles expand. This causes the vertices of $\mathcal{A}(S,t)$ to move, but it will also result in topological changes of the arc arrangement.

▶ **Definition 5** (Collision event). *Let $p_i(\alpha,t_{ij}^{min}) = p_j(\alpha+\pi,t_{ij}^{min})$ be the point of intersection of the offset circles of s_i and s_j at the collision time t_{ij}^{min}, for some fixed angle α. A* collision event *occurs between these two offset circles at time t_{ij}^{min} if the points $p_i(\alpha,t)$ and $p_j(\alpha+\pi,t)$ have been active for all times $0 \le t \le t_{ij}^{min}$.*

At the time of a collision a new pair of married vertices $v_{i,j}^l(t)$ and $v_{i,j}^r(t)$ is created. Of course, we have $v_{i,j}^l(t_{ij}^{min}) = v_{i,j}^r(t_{ij}^{min}) = p_i(\alpha,t_{ij}^{min})$.

▶ **Definition 6** (Domination event). *Let $p_i(\alpha, t_{ij}^{max}) = p_j(\alpha, t_{ij}^{max})$ be the point of intersection of the offset circles of s_i and s_j at the domination time t_{ij}^{max}, for some fixed angle α. A* domination event *occurs between these two offset circles at time t_{ij}^{max} if the points $p_i(\alpha, t)$ and $p_j(\alpha, t)$ have been active for all times $0 \le t \le t_{ij}^{max}$.*

At the time of a domination event the married vertices $v_{i,j}^l(t_{ij}^{max})$ and $v_{i,j}^r(t_{ij}^{max})$ coincide and are removed.

▶ **Definition 7** (Arc event). *An* arc event *e occurs at time t_e when an active arc a_i shrinks to zero length because two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e on $c_i(t_e)$.*

Lemma 2 implies that $p_i(\alpha, t)$ has been active for all times $t \le t_e$ if $p_i(\alpha, t_e) = p_e$. At the time of an arc event two unmarried vertices trade their places along an offset circle. Now suppose that the two unmarried vertices $v_{i,j}(t_e)$ and $v_{i,k}(t_e)$ meet in a point p_e along $c_i(t_e)$ at the time t_e of an arc event, thereby causing an active arc of $c_i(t_e)$ to shrink to zero length. Hence, the offset circles of s_i, s_j and s_k intersect at the point p_e at time t_e. If $c_j(t)$ and $c_k(t)$ did not intersect for $t < t_e$ then we also get a collision event between $c_j(t)$ and $c_k(t)$ at time t_e, see Figure 3a. (This configuration can occur for any relative order of the weights $w(s_i), w(s_j), w(s_k)$.) Otherwise, one or both of the married vertices $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ must also coincide with p_e. If both coincide with p_e then we also get a domination event between $c_j(t)$ and $c_k(t)$ at time t_e and we have $w(s_j) < w(s_k)$, see Figure 3b. The scenarios remaining for the case that only one of $v_{j,k}^l(t_e)$ and $v_{j,k}^r(t_e)$ coincides with p_e are detailed in the following lemma.

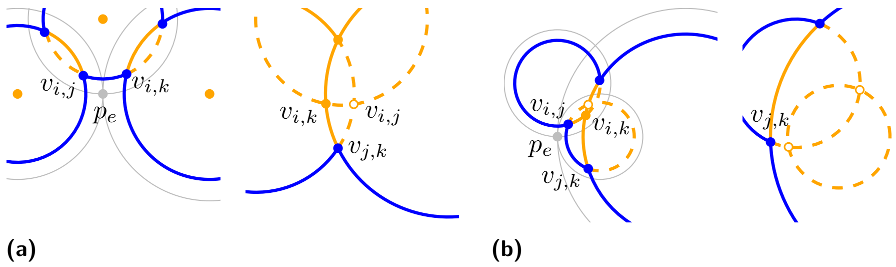
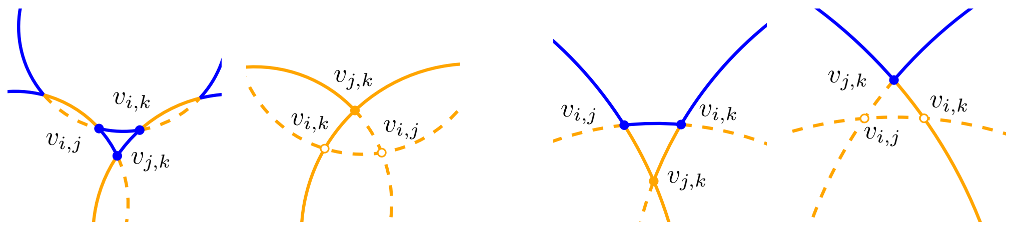


Figure 3 (a) The configuration shortly before (left) and after (right) a collision event as well as an arc event occur simultaneously at the same point p_e. In the left figure the offset arcs at the time of the event are shown in gray. Arcs and vertices that are on $\mathcal{WF}(\{s_i, s_j, s_k\}, t)$ are highlighted in blue. Other active arcs and vertices are depicted by solid orange lines and filled disks, while inactive arcs and vertices are depicted by dashed orange lines and circles. (b) The configuration shortly before and after a domination event and an arc event occur simultaneously at the same point p_e.

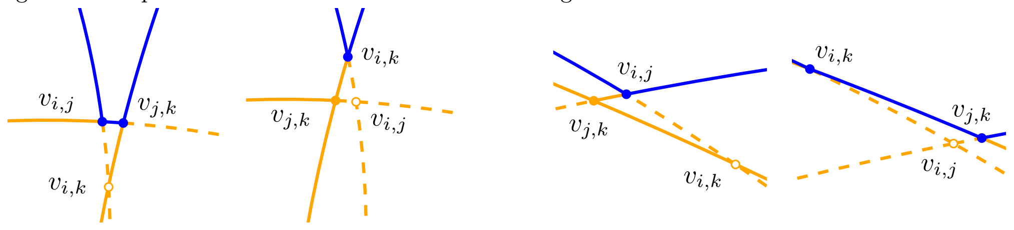
▶ **Lemma 8.** *Let $i < j < k$ and consider an arc event such that exactly the three vertices $v_{i,j}(t_e)$, $v_{i,k}(t_e)$, and $v_{j,k}(t_e)$ coincide at time t_e. Then either*

- *all three vertices were active before the event, see Figure 4a, or*
- *$v_{i,j}$ and $v_{j,k}$ were active and $v_{i,k}$ was inactive before the event, see Figure 4b, or*
- *$v_{i,k}$ and $v_{j,k}$ were active and $v_{i,j}$ was inactive before the event, see Figure 4c.*

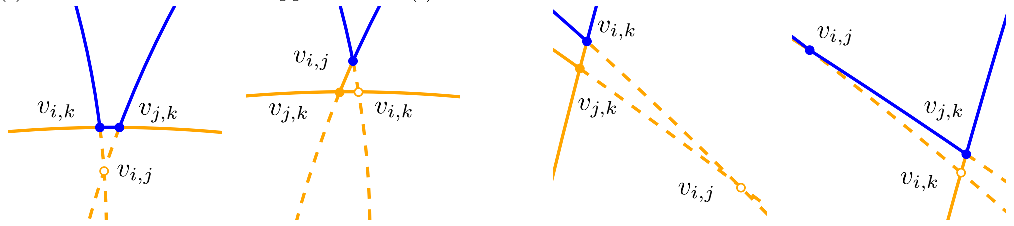
We now describe an event-handling scheme that allows us to trace out $\mathcal{VD}_w(S)$ by simulating the expansion of the arcs of $\mathcal{A}(S, t)$ as t increases, see Figure 5. We refer to this process as *arc expansion*.



(a) The two possible configurations shortly before (shown in the left figures) and after (shown in the right figures) one active arc disappears on $c_i(t)$ if no collision or domination event occurs at the same point. We get the collapse of all three arcs of an active-arc triangle.



(b) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_j(t)$ and another active arc appears on $c_k(t)$.



(c) The two possible configurations shortly before (left) and after (right) one active arc disappears on $c_k(t)$ and another active arc appears on $c_j(t)$.

Figure 4 The six different configurations that can occur for arc events for $1 \leq i < j < k \leq n$.

For each site we maintain a search data structure to keep track of all active arcs during the arc expansion. This *active offset* o_i of s_i holds the set of all arcs of $c_i(t)$ which are active at time t sorted in counter-clockwise angular order around s_i, and supports the following basic operations in time logarithmic in the number of arcs stored:

- It supports the insertion and deletion of active arcs as well as the lookup of their corresponding vertices.
- It supports point-location queries, allowing us to identify that active arc within o_i which contains a query point p on $c_i(t)$.

Every active offset contains at most $2(n-1)$ vertices and, thus, $O(n)$ active arcs. Hence, each such operation on an active offset takes $\mathcal{O}(\log n)$ time in the worst case.

Checking and handling the configurations shown in Figures 3a to 4 can be done by using only basic operations within the respective active offsets. The events themselves are stored in a priority queue $\mathcal{Q}$ ordered by the time of their occurrence. If two events take place simultaneously at the same point then collision events are prioritized higher than arc events, and arc events have to be handled before domination events. Four auxiliary operations are utilized that allow a more compact description of this process. Each one takes $\mathcal{O}(\log n)$ time.

- The *collapse-operation* takes place from $v_{i,x}$ to $v_{j,k}$ within an active offset o_x, with $x \in \{j,k\}$, in which $v_{i,x}$ and $v_{j,k}$ bound an active arc a_x that is already part of o_x; see Figure 6a. It determines the neighboring active arc a'_x of a_x that is bounded (on one side) by $v_{i,x}$, deletes a_x from o_x, and replaces $v_{i,x}$ by $v_{j,k}$ in a'_x.

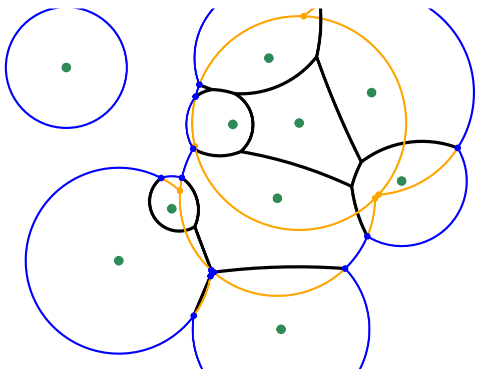
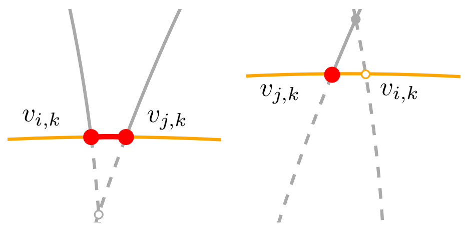
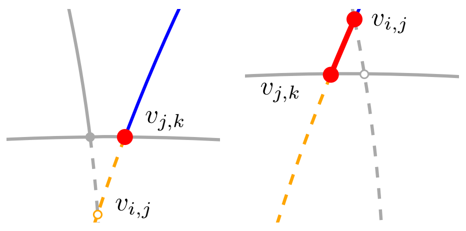


Figure 5 A snapshot of the arc expansion for the input shown in Figure 1. Active arcs that are currently not part of the wavefront are drawn in orange.

- The counterpart of the collapse-operation is the *expand-operation*; see Figure 6b. It happens from $v_{j,k}$ to $v_{i,x}$ in which $v_{j,k}$ bounds an active arc a'_x within o_x. The expansion will either move along a currently inactive or an already active portion of the offset circle of s_x. In the latter case, $v_{j,k}$ is replaced by $v_{i,x}$ in a'_x. In any case, we insert the respective active arc that is bounded by $v_{i,x}$ and $v_{j,k}$ into o_x.
- A *split-operation* involves two active offsets o_i and o_j as well as a point p_e which is situated within the active arcs $a_i := (v_{i,s}, v_{i,e})$ and $a_j := (v_{j,s'}, v_{j,e'})$ within o_i and o_j, respectively; see Figure 7a. Two married vertices $v^l_{i,j}$ and $v^r_{i,j}$ are created. Afterwards a_i and a_j are removed from o_i and o_j, respectively. Two new active arcs $(v_{i,s}, v^l_{i,j})$ and $(v^r_{i,j}, v_{i,e})$ are created and inserted into o_i. Furthermore, the three active arcs $(v_{j,e}, v^r_{i,j})$, $(v^r_{i,j}, v^l_{i,j})$, and $(v^l_{i,j}, v_{j,e})$ are inserted into o_j. If a_i and a_j were wavefront arcs then the newly created married vertices coincide with wavefront vertices and the newly inserted active arcs except $(v^r_{i,j}, v^l_{i,j})$ are marked as wavefront arcs.
- During a *merge-operation*, exactly two offset circles interact; see Figure 7b. The active arcs a_i and a_j bounded by the two corresponding married vertices $v^r_{i,j}$ and $v^l_{i,j}$ are removed from o_i and o_j, respectively. Additionally, the active arcs $(v_{j,s}, v^r_{i,j})$ and $(v^l_{i,j}, v_{j,e})$ that were adjacent to a_j within o_j are removed. Finally, a new active arc $a'_j := (v_{j,s}, v_{j,e})$ is inserted into o_j. If a_j was a wavefront arc then a'_j is also marked as a wavefront arc.



(a)



(b)

Figure 6 (a) A collapse-operation from $v_{i,k}$ to $v_{j,k}$ takes place within o_k. (b) An expand-operation happens within o_j from $v_{j,k}$ to $v_{i,j}$.

Domination events and arc events are easy to detect. The point and time of a collision is trivial to compute for any pair of offset circles, too. Unfortunately there is no obvious way to identify those pairs of circles for which this intersection will happen within portions of these offset circles which will still be active at the time of the collision. Hence, for the rest of

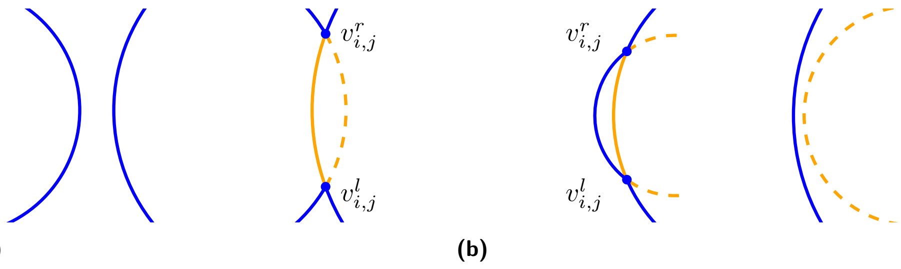


Figure 7 (a) A split-operation happens when at the time of a collision event. (b) A merge-operation happens at the time of a domination event.

this section we assume that all collisions among all pairs of offset circles are computed prior to the actual arc expansion. Lemma 9 verifies that our algorithm correctly simulates the arc expansion.

▶ **Lemma 9.** *For time $t > 0$, the arc arrangement $\mathcal{A}(S,t)$ can be obtained from $\mathcal{A}(S,0)$ by modifying it according to all collision events, domination events and arc events that occur till time t, in the order in which they appear.*

If the maximum weight of all sites is associated with only one site then there will be a time t when the offset circle of this site dominates all other offset circles, i.e., when $\mathcal{WF}(S,t)$ contains only this offset circle as one active arc. Obviously, at this time no further event can occur and the arc expansion stops. If multiple sites have the same maximum weight then $\mathcal{Q}$ can only be empty once $\mathcal{WF}(S,t)$ contains only one loop of active arcs which all lie on offset circles of these sites and if all wavefront vertices move along rays to infinity.

▶ **Lemma 10.** *An active arc or active vertex within an active offset is identified and marked as a wavefront arc (wavefront vertex, resp.) at time $t \geq 0$ if and only if it lies on $\mathcal{WF}(S,t)$.*

If we allow points in $\mathbb{R}^2$ to have the same weighted distance to more than three sites then we need to modify our strategy. In particular, we need to take care of constellations in which more than three arc events happen simultaneously at the same point. In such a case it is necessary to carefully choose the sequence in which the corresponding arc events are handled. More precisely, an arc event may only be handled (without corrupting the state of the active offsets) whenever the respective active vertices are considered neighboring within the active offsets. If the active vertices that participate in an arc event are not currently neighboring then we can always find an arc event whose active vertices are neighboring that happens simultaneously at the same location by walking along the corresponding active offsets. By dealing with the arc events in this specific order, we generate multiple coinciding Voronoi nodes of degree three. Domination events that occur simultaneously at the same point p_e are processed in increasing order of the weights. Note that this order can already be established at the time when an event is inserted into $\mathcal{Q}$, at no additional computational cost. Simultaneous multiple collision events at the same point p_e either involve arcs that are not active or coincide with arc events. These arc events automatically establish a sorted order of the active arcs around p_e, thus allowing us to avoid an explicit (and time-consuming) sorting.

▶ **Lemma 11.** *During the arc expansion $\mathcal{O}(n^2)$ collision and domination events are computed.*

We know that collision events create and domination events remove active vertices (and make them inactive for good). A collapse of an entire active-arc triangle causes two vertices to become inactive. During every other arc event at least one active vertex becomes inactive, but at the same time one inactive vertex may become active again. In order to bound the number of arc events it is essential to determine how many vertices can be active and how often a vertex can undergo a *reactivation*, i.e., change its status from inactive to active. (Note that Lemma 2 is not applicable to a moving vertex since its polar angle does not stay constant.) We now argue that the total number of reactivations of inactive vertices is bounded by the number of different vertices that ever were active during the arc expansion.

▶ **Lemma 12.** *Every reactivation of a moving vertex during an arc event forces another moving vertex to become inactive and remain inactive for the rest of the arc expansion.*

▶ **Lemma 13.** *Let h be the number of different vertices that ever were active during the arc expansion. Then $\mathcal{O}(h)$ arc events can take place during the arc expansion.*

▶ **Theorem 14.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites can be computed in $\mathcal{O}(n^2 \log n)$ time and $\mathcal{O}(n^2)$ space.*

Additionally, in the full version [9] we argue that the one-dimensional MWVD can be computed efficiently using a wavefront-based strategy.

▶ **Theorem 15.** *The multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n weighted point sites in one dimension can be computed in $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n)$ space.*

6 Reducing the Number of Collisions Computed

Experiments quickly indicate that the vast majority of pairwise collisions computed a priori never ends up on pairs of active arcs. Furthermore, the resulting Voronoi diagrams show a quadratic combinatorial complexity only for contrived input data. We make use of the following results to determine all collision events in near-linear expected time. Throughout this section, we assume that for each site $s_i \in S$ the corresponding weight $w(s_i)$ is independently sampled from some probability distribution.

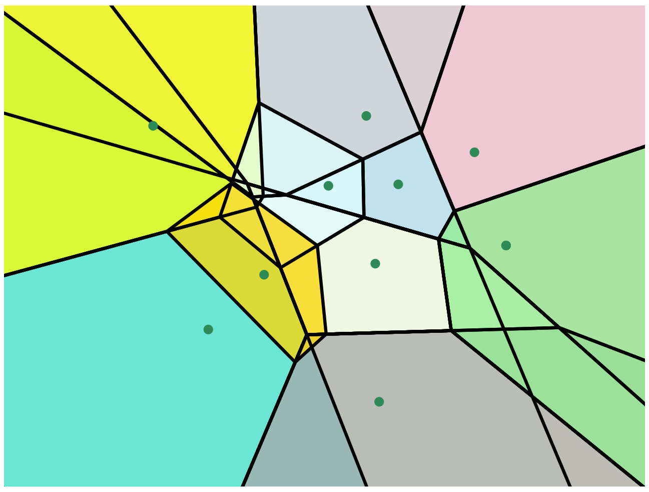


Figure 8 The overlay arrangement is generated by inserting the sites ordered by decreasing weights.

▶ **Definition 16** (Candidate Set). *Consider an arbitrary (but fixed) point $q \in \mathbb{R}^2$, and let s be its nearest neighbor in S under the weighted distance. Let $s' \in S \setminus \{s\}$ be another site. Since s is the nearest neighbor of q we know that either s has a higher weight than s' or a* *smaller Euclidean distance to q than s'. Thus, one can define a* candidate set *for a weighted nearest neighbor of q which consists of all sites $s \in S$ such that all other sites in S either have a smaller weight or a larger Euclidean distance to q.*

▶ **Lemma 17** (Har-Peled and Raichel [8])**.** *For all points $q \in \mathbb{R}^2$, the candidate set for q among S is of size $\mathcal{O}(\log n)$ with high probability.*

▶ **Lemma 18** (Har-Peled and Raichel [8])**.** *Let K_i denote the Voronoi cell of s_i in the unweighted Voronoi diagram of the i-th suffix $S_i := \{s_i, \ldots, s_n\}$. Let $\mathcal{OA}$ denote the arrangement formed by the overlay of the regions $K_1, \ldots, K_n$. Then, for every face f of $\mathcal{OA}$, the candidate set is the same for all points in f.*

Figure 8 shows a sample overlay arrangement. Kaplan et al. [11] prove that this overlay arrangement has an expected complexity of $\mathcal{O}(n \log n)$. Note that their result is applicable since inserting the points in sorted order of their randomly chosen weights corresponds to a randomized insertion. These results allow us to derive better complexity bounds.

▶ **Theorem 19** (Kaplan et al. [11])**.** *The expected combinatorial complexity of the overlay of the minimization diagrams that arises during a randomized incremental construction of the lower envelope of n hyperplanes in $\mathbb{R}^d$, for $d \geq 2$, is $\mathcal{O}(n^{\lfloor d/2 \rfloor})$, for d even, and $O(n^{\lfloor d/2 \rfloor} \log n)$, for d odd. The bounds for d even and for $d = 3$ are tight in the worst case.*

▶ **Lemma 20.** *If a collision event occurs between the offset circles of two sites $s_i, s_j \in S$ then there exists at least one candidate set which includes both s_i and s_j.*

▶ **Theorem 21.** *All collision events can be determined in $\mathcal{O}(n \log^3 n)$ expected time by computing the overlay arrangement $\mathcal{OA}$ of a set S of n input sites.*

Thus, the number h of vertices created during the arc expansion can be expected to be bounded by $\mathcal{O}(n \log^3 n)$. Lemma 13 tells us that the number of arc events is in $\mathcal{O}(h)$. Therefore, $\mathcal{O}(n \log^3 n)$ events happen in total.

▶ **Theorem 22.** *A wavefront-based approach allows to compute the multiplicatively weighted Voronoi diagram $\mathcal{VD}_w(S)$ of a set S of n (randomly) weighted point sites in expected $\mathcal{O}(n \log^4 n)$ time and expected $\mathcal{O}(n \log^3 n)$ space.*

7 Extensions

Consider a set S' of n disjoint weighted straight-line segments in $\mathbb{R}^2$. A wavefront propagation among weighted line segments requires us to refine our notion of "collision". We call an intersection of two offset circles a *non-piercing collision event* if it marks the initial contact of the two offset circles. That is, it occurs when the first pair of moving vertices appear. We call an intersection of two offset circles a *piercing collision event* if it takes place when two already intersecting offset circles intersect in a third point for the first time; see Figure 9. In this case, a second pair of moving vertices appear.

Hence, a minor modification of our event-based construction scheme is sufficient to extend it to weighted straight-line segments; see Figure 10. We only need to check whether a piercing collision event that happens at a point p_e at time t_e currently is part of $\mathcal{WF}(S', t_e)$. In such a case the two new vertices as well as the corresponding active arc between them need to be flagged as part of $\mathcal{WF}(S', t_e)$.

An extension to additive weights can be integrated easily into our scheme by simply giving every offset circle a head-start of $w_a(s_i)$ at time $t = 0$, where $w_a(s_i) \geq 0$ denotes the real-valued additive weight that is associated with s_i.

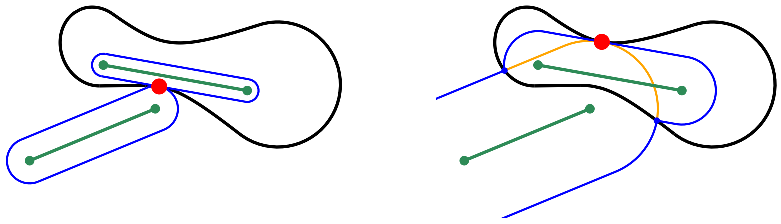


Figure 9 An example of a non-piercing (left) as well as a piercing collision event (right).

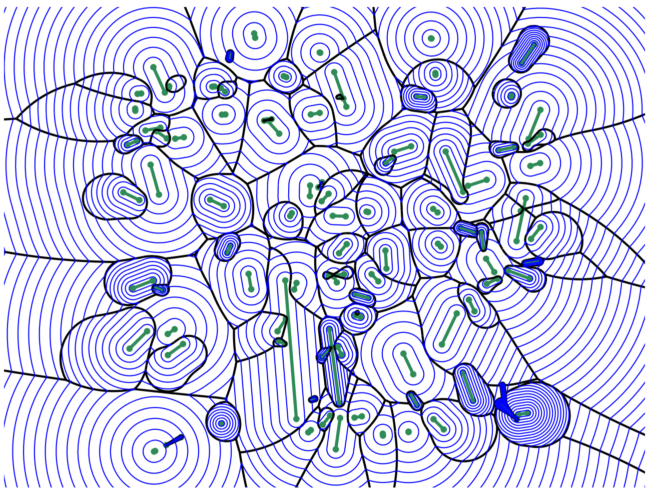


Figure 10 The MWVD of a set of weighted points and weighted straight-line segments together with a family of wavefronts for equally-spaced points in time.

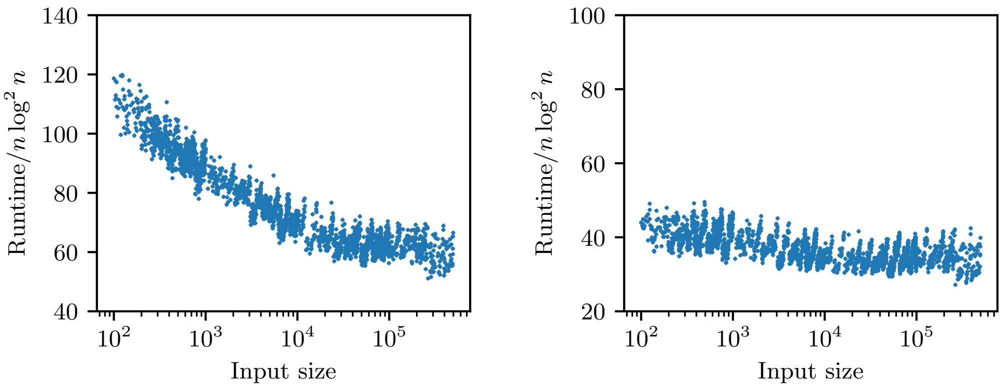
8 Experimental Evaluation

We implemented our full algorithm for multiplicatively weighted points as input sites[2], based on CGAL and exact arithmetic[3]. In particular, we use CGAL's `Arrangement_2` package for computing the overlay arrangement and its `Voronoi_diagram_2` package for computing unweighted Voronoi diagrams. The computation of the MWVD itself utilizes CGAL's `Exact_circular_kernel_2` package which is based on the `Gmpq` number type. The obvious advantage of using exact number types is that events are guaranteed to be processed in the right order even if they occur nearly simultaneously at nearly the same place. One of the main drawbacks of exact number types is their memory consumption which is significantly (and sometimes unpredictably) higher than when standard floating-point numbers are used.

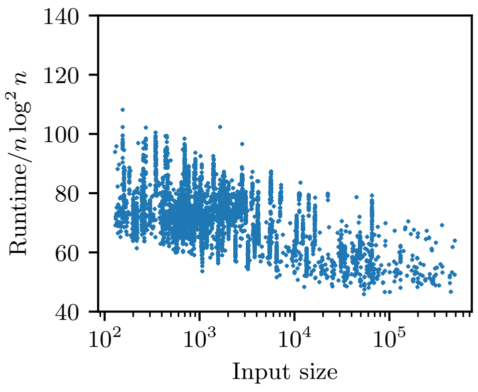
We used our implementation for an experimental evaluation and ran our code on over 8000 inputs ranging from 256 vertices to 500 000 vertices. For all inputs all weights were chosen uniformly at random from the interval $[0, 1]$. All tests were carried out with CGAL 5.0 on an Intel Core i9-7900X processor clocked at 3.3 GHz.

[2] We do also have a prototype implementation that handles both weighted points and weighted straight-line segments. It was used to generate Figure 10.

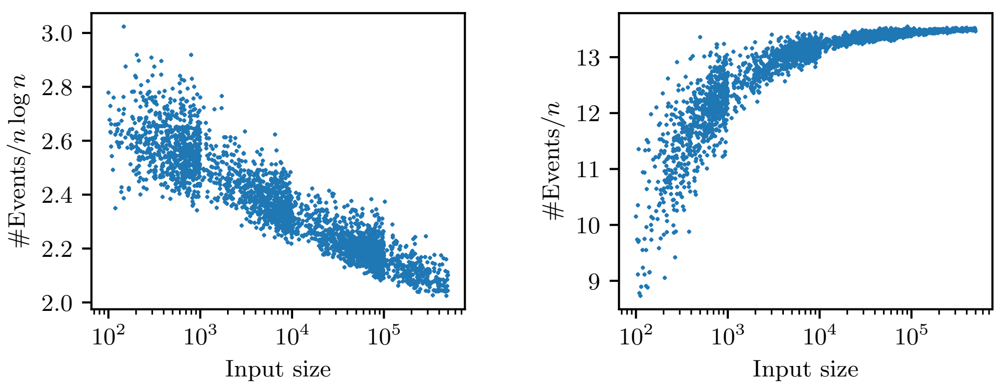
[3] We have not spent enough time on fine-tuning an implementation based on conventional floating-point arithmetic. The obvious crux is that inaccurately determined event times (and locations) may corrupt the state of the arc arrangement and, thus, cause a variety of errors during the subsequent arc expansion.



(a) Left: The overall runtime results for inputs with randomly generated weights and point coordinates. Right: The runtime consumed by the computation of the corresponding overlay arrangements. All runtimes were divided by $n\log^2 n$.



(b) The overall runtime results for inputs with randomly generated weights and vertices of real-world polygons and polygons of the Salzburg database of polygonal data [5, 6] taken as input points. The runtimes were divided by $n\log^2 n$.



(c) The left plot shows the total number of (valid and invalid) collision events (divided by $n\log n$); the right plot shows the number of arc events (divided by n) processed during the arc expansion. All point coordinates and weights were generated randomly.

Figure 11 Experimental evaluation.

In any case, the number of events is smaller than predicted by the theoretical analysis. This is also reflected by our runtime statistics: In Figures 11a and 11b the runtime that was consumed by the computation of a MWVD is plotted. We ran our tests on two different input classes: The point locations were either generated randomly, i.e., they were chosen according to either a uniform or a normal distribution, or obtained by taking the vertices of real-world polygons or polygons of the brand-new Salzburg database of polygonal data [5, 6]. Summarizing, our tests suggest an overall runtime of $\mathcal{O}(n \log^2 n)$ for both input classes. In particular, the actual geometric distribution of the sites does not have a significant impact on the runtime if the weights are chosen randomly: For real-world, irregularly distributed sites the runtimes are scattered more wildly than in the case of uniformly distributed sites, but they do not increase. The numbers of collision events and arc events that occurred during the arc expansion are plotted in Figure 11c. Our tests suggest that we can expect to see at most $3n \log n$ collision events and at most most $14n$ arc events to occur. Note that the number of arc events forms an upper bound on the number of Voronoi nodes of the final MWVD. That is, random weights seem to result in a linear combinatorial complexity of the MWVD.

It is natural to ask how much these results depend on the randomness of the weights. To probe this question we set up a second series of experiments: We sampled points uniformly within a square with side-length $\sqrt{2}$ and then tested different weights. Let $d(s)$ be the distance of the site $s \in S$ from the center of the square, and let $r(s)$ be a number uniformly distributed within the interval $[0, 1]$. Of course, $0 \le d(s) \le 1$. Then we assign ${}^{\alpha \cdot d(s) + \beta \cdot r(s)}/_{(\alpha+\beta)}$ as weight to s, with α and β being the same arbitrary but fixed non-negative numbers for all sites of S. Figure 12 shows the results obtained for the same sets of points and the (α, β)-pairs $(1, 0)$, $(9, 1)$, $(7, 3)$, $(1, 1)$ and $(0, 1)$. This test makes it evident that the bounds on the complexities need not hold if the weights are not chosen randomly, even for a uniform distribution of the sites. Rather, this may lead to a linear number of candidates per candidate set and a quadratic runtime complexity, as shown in Figure 12.

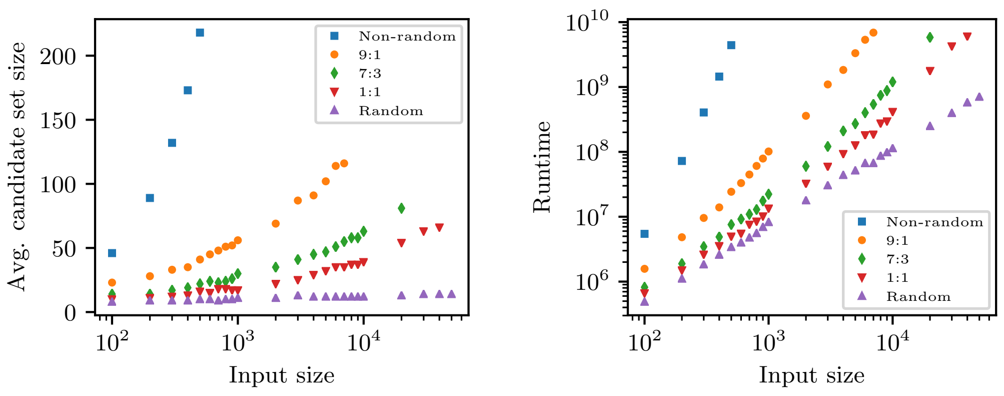


Figure 12 The plots show how the average number of candidates (left) and the total runtime (right) depend on the weights assigned to the sites. Each marker on the x-axes indicates the number n of input sites uniformly distributed within a square.

9 Conclusion

We present a wavefront-like approach for computing the MWVD of points and straight-line segments. Results by Kaplan et al. [11] and Har-Peled and Raichel [8] allow to predict an $\mathcal{O}(n \log^4 n)$ expected time complexity for point sites with random weights. We also discuss a robust, practical implementation which is based on CGAL and exact arithmetic. Extensive tests of our code indicate an average runtime of $\mathcal{O}(n \log^2 n)$ if the sites are weighted randomly. To the best of our knowledge, there does not exist any other code for computing MWVDs that is comparatively fast. A simple modification of our arc expansion scheme makes it possible to handle both additive and multiplicative weights simultaneously. Our code is publicly available on GitHub under `https://github.com/cgalab/wevo`. Figure 13 shows several examples of MWVDs computed by our implementation.

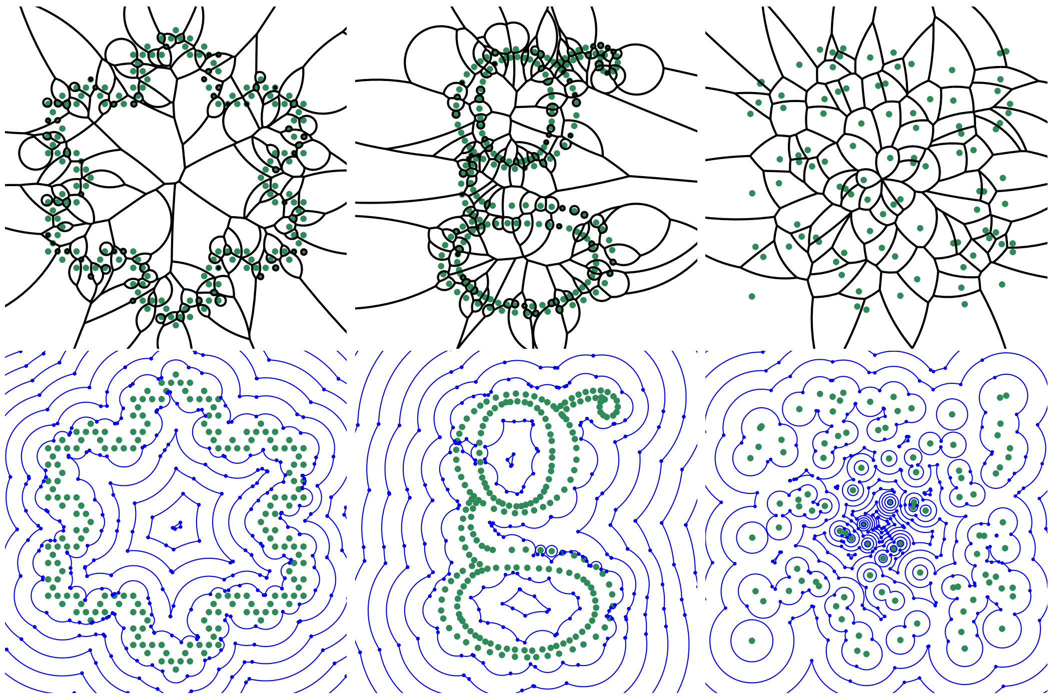


Figure 13 Several examples of MWVDs are shown in the top figures. The bottom figures illustrate a series of uniformly distributed wavefronts that have been derived from the corresponding MWVDs.

References

1 Franz Aurenhammer. The One-Dimensional Weighted Voronoi Diagram. *Information Processing Letters*, 22(3):119–123, 1986. doi:10.1016/0020-0190(86)90055-4.

2 Franz Aurenhammer and Herbert Edelsbrunner. An Optimal Algorithm for Constructing the Weighted Voronoi Diagram in the Plane. *Pattern Recognition*, 17(2):251–257, 1984. doi:10.1016/0031-3203(84)90064-5.

3 Rudi Bonfiglioli, Wouter van Toll, and Roland Geraerts. GPGPU-Accelerated Construction of High-Resolution Generalized Voronoi Diagrams and Navigation Meshes. In *Proceedings of the Seventh International Conference on Motion in Games*, pages 26–30, 2014. doi:10.1145/2668084.2668093.

4 Barry N. Boots. Weighting Thiessen Polygons. *Economic Geography*, 56(3):248–259, 1980. doi:10.2307/142716.

5 Günther Eder, Martin Held, Steinþór Jasonarson, Philipp Mayer, and Peter Palfrader. On Generating Polygons: Introducing the Salzburg Database. In *Proceedings of the 36th European Workshop on Computational Geometry*, pages 75:1–75:7, March 2020.

6 Computational Geometry and Applications Lab Salzburg. Salzburg Database of Geometric Inputs. `https://sbgdb.cs.sbg.ac.at/`, 2020.

7 Leonidas Guibas. Kinetic Data Structures. In Dinesh P. Mehta and Sartaj Sahni, editors, *Handbook of Data Structures and Applications*, pages 23.1–23.18. Chapman and Hall/CRC, 2001. ISBN 9781584884354.

8 Sariel Har-Peled and Benjamin Raichel. On the Complexity of Randomly Weighted Multiplicative Voronoi Diagrams. *Discrete & Computational Geometry*, 53(3):547–568, 2015. `doi:10.1007/s00454-015-9675-0`.

9 Martin Held and Stefan de Lorenzo. An Efficient, Practical Algorithm and Implementation for Computing Multiplicatively Weighted Voronoi Diagrams, 2020. `arXiv:2006.14298`.

10 Kenneth E. Hoff III, John Keyser, Ming Lin, Dinesh Manocha, and Tim Culver. Fast Computation of Generalized Voronoi Diagrams using Graphics Hardware. In *Proceedings of the the 26th Annual International Conference on Computer Graphics and Interactive Techniques*, pages 277–286. ACM Press/Addison-Wesley Publishing Co., 1999. `doi:10.1145/311535.311567`.

11 Haim Kaplan, Edgar Ramos, and Micha Sharir. The Overlay of Minimization Diagrams in a Randomized Incremental Construction. *Discrete & Computational Geometry*, 45(3):371–382, 2011. `doi:10.1007/s00454-010-9324-6`.

12 The CGAL Project. *CGAL User and Reference Manual*. CGAL Editorial Board, 5.0 edition, 2019. URL: `https://doc.cgal.org/5.0/Manual/packages.html`.

13 Kira Vyatkina and Gill Barequet. On Multiplicatively Weighted Voronoi Diagrams for Lines in the Plane. *Transactions on Computational Science*, 13:44–71, 2011. `doi:10.1007/978-3-642-22619-9_3`.