Weierstraß-Institut
für Angewandte Analysis und Stochastik

Leibniz-Institut im Forschungsverbund Berlin e. V.

Preprint ISSN 2198-5855

On the computation of high dimensional Voronoi diagrams

Martin Heida

submitted: August 29, 2023

Weierstrass Institute
Mohrenstr. 39
10117 Berlin
Germany
E-Mail: martin.heida@wias-berlin.de

No. 3041
Berlin 2023

2020 *Mathematics Subject Classification.* 65N50, 65Y20, 68Q25, 52-08, 52Bxx, 52-04.

Key words and phrases. Raycast, mesh generation, geometry, HighVoronoi.

The work was financed by DFG through the SPP2256 "Variational Methods for Predicting Complex Phenomena in Engineering Structures and Materials", project HE 8716/1-1, project ID:441154659.

Edited by
Weierstraß-Institut für Angewandte Analysis und Stochastik (WIAS)
Leibniz-Institut im Forschungsverbund Berlin e. V.
Mohrenstraße 39
10117 Berlin
Germany

Fax: +49 30 20372-303
E-Mail: preprint@wias-berlin.de
World Wide Web: http://www.wias-berlin.de/

On the computation of high dimensional Voronoi diagrams

Martin Heida

Abstract

We investigate a recently implemented new algorithm for the computation of a Voronoi diagram in high dimensions and generalize it to N nodes in general or non-general position using a geometric characterization of edges merging in a given vertex. We provide a mathematical proof that the algorithm is exact, convergent and has computational costs of $O(E\,\mathbb{NN}(N))$, where E is the number of edges and $\mathbb{NN}(N)$ is the computational cost to calculate the nearest neighbor among N points. We also provide data from performance tests in the recently developed Julia package „HighVoronoi.jl".

Contents

DOI 10.20347/WIAS.PREPRINT.3041 Berlin 2023

1 Introduction

In computational geometry, the Voronoi diagram and Delaunay triangulation stand as fundamental cornerstones, with a broad variety of applications. As such we find for the Voronoi diagram applications in molecular modeling [6, 5], political science [10] topological data analysis [13] or mesh generation for finite volume methods [7]. The Delaunay triangulation is among other applications widely used in finite element analysis.

Beyond these applications, Voronoi diagrams and Delaunay triangulations have and inherent mathematical elegance which make them very interesting objects and very suitable to provide insights into the spatial partitioning of points, defining regions of influence and revealing underlying patterns within datasets.

In this work, we focus on the Voronoi diagram and its computation for *any given set of* N *points* in the Euclidian geometry, as long as there are at least d linear independent points in $\mathbb{R}^d$. By this we mean that the method also applies to generators in non-general position.

In a mathematical language, given N *generators (nodes)* $(x_i)_{i=1,...,N}$ in $\mathbb{R}^d$ satisfying

$$\forall i, j \in \{1, \dots, N\}\,, \qquad x_i \neq x_j\,, \tag{1}$$

the *Voronoi diagram* introduced in [15] is a partitioning of the space into cells $(x_i)_{i=1,...,N}$ according to the following definition.

Definition 1.1 (Voronoi cells and vertices)**.** Let $X = \{x_1, ..., x_N\}$ be a set of *nodes (generators)* where $x_i \in \mathbb{R}^d$. The Voronoi diagram of X consists of N closed cells

$$(C_i)_{i=1,...,n}\,, \qquad C_i = \left\{x \in \mathbb{R}^d \,:\, \forall j \neq i \; |x - x_i| \leq |x - x_j|\right\}\,.$$

Introducing for every $x \in \mathbb{R}^d$ the local quantities

$$\mathcal{N}(x) := \{i \in \{1, \dots, n\} \,:\, x \in C_i\}\,, \qquad N(x) := \#\mathcal{N}(x)\,,$$

a point $\nu \in \mathbb{R}^d$ is called *vertex* if there exists $\delta > 0$ such that for every $x \in \mathbb{B}_\delta(\nu) \setminus \{\nu\}$ it holds $N(x) < N(\nu)$. The set of vertices is denoted $\mathcal{V}(X)$ and we also denote locally $\mathcal{V}_i := \mathcal{V}(X) \cap C_i$ the vertices of C_i.

Our definition of a vertex appears different from the usual definitions in literature but we will see that for generators in general position, this is equivalent with the classical definition:

Definition 1.2 (Generators in general position)**.** A set of generators is in general position if for $d+k$ mutually different cells $(C_{i_j})_{j=1,\dots d+k}$ it holds

1 If $k > 1$ then $\bigcap_j C_{i_j} = \emptyset$.

2 If $k = 1$ then $\bigcap_j C_{i_j} = \emptyset$ or $\bigcap_j C_{i_j} = \{x_0\}$ and x_0 is called the vertex of $(C_{i_j})_{j=1,\dots d+1}$.

A single vertex ν is in general position if $N(\nu) = d+1$.

Remark. If the nodes are in general position it implies that every vertex is in general position. The nodes are in general position if the circumcircle of every vertex has exactly $d+1$ generators on the boundary sphere.

The Delaunay triangulation was introduced independently in [4], and is characterized to connect points in such a way that no point is inside the circumcircle (circumball in higher dimensions) of any triangle (tetrahedron) in the triangulation. The Delaunay triangulation is the dual graph of the Voronoi diagram and is well defined iff the generators are in general position. The derivation of the Delaunay triangulation then goes basically with $O(N)$.

If the generators are not in general position, we find that the circumcircles of vertices have more than $d+1$ generators on there circumcircle. This implies that the triangulation of these local generators is not unique and it is difficult to calculate a good triangulation, i.e. in such a way that facets of tetrahedrons belonging to neighboring vertices properly interlap. On the other hand, the Voronoi diagram is always unique.

While the significance of Voronoi diagrams and Delaunay triangulations is undeniable, the methods to compute these structures have evolved over time. The most common approach to Voronoi diagrams are the computation of the Delaunay triangulation and deriving the Voronoi diagram as the dual. There are also direct methods such as the Fortune algorithm, but often working only in low dimensions. One of the early algorithms is the Bowyer-Watson algorithm [3, 16]. It works by subsequently adding points and comparing with the circumcircle. It has complexity $O(N\ln(N))$ in general but having $O(N^2)$ complexity for some rare cases. It works, however, for nodes in general positions only. This is a special case of so called incremental algorithms, that are known to yield $O(N\ln N)$ for nodes in general position.

Another method is to project the nodes into $\mathbb{R}^{d+1}$ using the map $(x_1,\dots,x_d) \to (x_1,\dots,x_d,\|x\|^2)$. Then the convex hull of this new set of points will correspond to tetrahedrons of the Delaunay triangulation of the original points. This method is limited in its applications to nodes in general position: Suppose we consider as nodes $\mathbb{Z}^2$. Then the facets in R^3 will contain four squares around the origin but all other original squares will be split into triangles. From here, one has a hard time to recover the original Voronoi cells. The search for the convex hull is typically done using the quickhull algorithm, which grows polynomially in high dimensions with $O(N\, f_{\mathcal{V}}|\mathcal{V}|^{-1})$ where $f_{\mathcal{V}}$ goes with $\exp(\mathrm{floor}(d/2)\ln|\mathcal{V}|)$ [1].

Finally, devide-and-conquer algorithms split the sets of points incrementally by planes into equal parts, until tetrahedrons are left.

This paper presents a novel approach that redefines the computation of Voronoi diagrams. The approach was introduced first in the `VoronoiGraph.jl` library [14] for nodes in general position and is an extension of a previously introduced idea of raycasting [12, 13]. The algorithm as such is based on a nearest neighbor search combined with a sound mathematical characterization of the edges emerging

Dimension	2	3	4	5	6	7
Vertices / Cell	6.6	29.2	191	1250	9720	91900
Neighbors / Cell	6.6	16.6	43	95	210	480
Approx. Variance	20%	15%	15%	15%	15%	10%

Table 1: Data for the average amount of vertices and neighbors for each cell when the nodes are generated i.i.d and are in general position. The data is generated as the average over 10 representative cells using the following Julia code:

```
for dim in 2:7
    println( HighVoronoi.VoronoiStatistics( dim, 10; geodata=false ) )
end
```

Dimension	2	3	4	5	6	7
Vertices / Cell	5.6	14	42	87	160	500
Neighbors / Cell	5.6	9	15	18	19	30
Approx. Variance	15%	30%	50%	30%	30%	60%

Table 2: Data for the average amount of vertices and neighbors for each cell when the nodes are generated by two nodes in a unit cell which is then copied periodically. In particular, the nodes are in non-general position. The data is generated as the average over 10 representative cells using the following Julia code:

```
for dim in 2:7
    println( HighVoronoi.VoronoiStatistics( dim, 10; periodic=2,
    ↪  geodata=false ) )
end
```

in a given vertex. While this characterization was implicitly used in `VoronoiGraph.jl` for nodes in general position, its universal mathematical formulation for arbitrary grids given below seems to be new, or at least unexploited.

We mention at this place that the usage of nodes in non-general position can be very benefiting in Finite Volume methods due to the reduced number of neighbors that comes with it. We illustrate this as follows: In a cubic mesh in $\mathbb{R}^d$, every cell has $2d$ neighbors. However, if the nodes are in general position, we observe that the number of neighbors and vertices increases dramatically with higher dimension, see Table 1. We can thus use quasi-periodic meshes that benefit from a low number of generators per vertex and a low number of neighbors per cell, see Table 2.

Also a new feature in this work is the inclusion of the boundary of a convex polygonal domain. The algorithm we construct works by „traveling along edges from vertex to vertex" and as such immediately has the potential of mesh refinement or the inclusion of periodic boundary conditions. All of these features were implemented in the new `HighVoronoi.jl` Julia package [9].

We will see that the computational effort grows linearly in the number of edges E multiplied with the effort to compute nearest neighbors. While for iid. distributed nodes this turns out to be $O(\ln N)$ in the KD-Tree search algorithm [8], for nodes in non-general position this performance unfortunately decreases to $O(N)$ for the nearest neighbor search. We suggest a solution to this issue in the outlook.

Outline

In Section 2 we provide the mathematics on which the algorithms are based. The theory culminates in the following four major results: Lemma 2.12 states that the well known fact that the Voronoi diagram is connected via edges, no matter whether the nodes are in general or non-general position. Lemma 2.11 states that it is always and in every cell possible to find an initial vertex as starting point for the travel. Lemma 2.9 provides the insight how we can travel from one vertex along a known edge to the neighboring vertex and Lemma 2.16 provides an efficient way to identify all edges emerging at a vertex with generators in general or non-general position.

In Section 3 we provide a fully exhaustive and terminating algorithm to calculate the Voronoi diagram based on the theory from Section 2. We also provide a proof for these properties, see Theorem 3.2. Theorem 3.5 provides the complexity estimate for the algorithm. This also leads to estimates for special cases.

In Section 4 we provide an algorithm suitable to calculate a refined version of a given Voronoi diagram when adding a set of further points. We prove in Theorem 4.1 that the resulting mesh is complete. Section 5 exploits the properties of the algorithm to introduce a fast way to calculate quasi-periodic grids.

In Section 6 we have a look at the robustness of the algorithm, i.e. how likely it is to find a corrupted vertex, edge or interface.

In Section 7 we discuss performance tests based on the Julia implementation `HighVoronoi.jl` and in Section 8 we give a wrap-up of our findings and provide an outlook.

2 The underlying mathematics

In what follows, we write $\mathbb{S}^{d-1}$ for the unit sphere in $\mathbb{R}^d$ and $\mathbb{B}_R(x)$ for a ball with radius R around x.

2.1 Properties of $\mathcal{N}(\,\cdot\,)$

Corollary. *Every Voronoi cell is convex and closed.*

Proof. This well known result is a direct consequence of the triangle inequality and the definition of a Voronoi cell. □

Corollary. *In the setting of Definition 1.1 the set V of vertices consists of isolated points. For every $\nu \in \mathcal{V}(X)$ it holds $N(\nu) \geq d+1$.*

Proof. The first statement is evident from the definition of a vertex. For the second statement, note that given $x_i, x_j \in C$ the set of equidistant points to $x_i, x_j \in C$ is a plane and it takes at least d planes to intersect in order to define a single point. □

The last corollary is important as it limits the amount of nodes needed to define a vertex from below by $d+1$. This lower bound is at the same time the upper bound for a huge class of Voronoi diagrams.

Lemma 2.1. *Let the points X be randomly distributed with a density function that is absolutely continuous w.r.t. the Lebesgue measure, i.e. the density is $\rho \mathrm{d}\mathcal{L}$. Then the probability of a given vertex ν to have $N(\nu) > d+1$ is zero.*

Proof. Suppose ν is a vertex and $d+1$ generators are known. Then these nodes all lie on a sphere of radius R around ν. The probability to find another node inside the spherical hull $\mathbb{B}_{R+\varepsilon}(\nu) \setminus \mathbb{B}_R(\nu)$ is then proportional to $|X| \int_{\mathbb{B}_{R+\varepsilon}(\nu)\setminus\mathbb{B}_R(\nu)} \rho \mathrm{d}\mathcal{L} \to 0$ as $\varepsilon \to 0$. □

Finally, it has to be mentioned that $\mathcal{N}(x)$ is upper semicontinuous in the following sense:

Lemma 2.2 ($\mathcal{N}(\cdot)$ and $N(\cdot)$ are upper semicontinuous)**.** *For every $y \in \mathbb{R}^d$ there exists $\delta > 0$ such that for every $\tilde{y} \in \mathbb{B}_\delta(y)$ it holds $\mathcal{N}(\tilde{y}) \subseteq \mathcal{N}(y)$. In particular, $N(x)$ is upper semicontinuous:*

$$\limsup_{x\to y} N(x) \leq N(y)\,. \tag{2}$$

Proof. Assume the first statement is wrong. Then for every $k \in \mathbb{N}$ there exists $y_k \in \mathbb{B}_{\frac{1}{k}}(y)$ and $x_{j_k} \in X$ with $x_{j_k} \in \mathcal{N}(y_k)$ but $x_{j_k} \notin \mathcal{N}(y)$. Since X is finite we can assume that $x_{j_k} = x_j$ is constant (subsequence argument!). But $x_j \in \mathcal{N}(y_k)$ implies $y_k \in C_j$. Since C_j is closed and $y_k \to y$ this implies $y \in C_j$ and x_j in $\mathcal{N}(y)$, a contradiction. This now leads straight forward to $N(\tilde{y}) \leq N(y)$ and (2). □

2.2 Vertices, neighbors, adjacents and edges

Every vertex ν satisfies $N(\nu) = d+k$, $k \geq 1$, and it is the only element of the intersection of the cells generated by $\{x_{\sigma_1}, \dots, x_{\sigma_{d+k}}\} \subset X$, which we store as

$$\boldsymbol{\sigma}_\nu = [\sigma_1, \dots, \sigma_{d+k}]\,, \quad \text{where} \quad \sigma_1 < \sigma_2 < \dots < \sigma_{d+k}.$$

Definition 2.1. We say that two nodes x_i, x_j are *adjacent* if they share a vertex, i.e. if $\mathcal{V}_i \cap \mathcal{V}_j \neq \emptyset$. We say that two nodes x_i, x_j are *neighbors* if they share a positive interfacial area, i.e. if $\partial C_i \cap \partial C_j$ has positive $(d-1)$-dimensional measure.

Corollary. *x_i, x_j are neighbors if and only if they share d linearly independent vertices.*

Proof. This follows since the intersection of both cells is convex and it needs d points to span a $(d-1)$-dimensional hyperplane. □

Definition 2.2. An *edge* of the Voronoi diagram generated by X is a 1-dimensional set of positive or infinite 1-dimensional measure for which there exist at least d elements $\{x_{j_1}, \dots, x_{j_d}\} \subset X$ such that

$$\eta = \bigcap_{k=1,\dots,d} C_{j_k}\,. \tag{3}$$

We write $\eta = (x_{j_1}, \dots, x_{j_d})$. We denote $X_\eta := \{x_i \in X \,:\, \eta \subset C_i\}$ and say that η is an edge of $x_i \in X_\eta$.

Lemma 2.3. *1. Let η be an edge. Then for every $x_0 \in X_\eta$ the set $X_\eta - x_0$ contains $d-1$ linear independent points and there exists $\mathbf{n} \in \mathbb{S}^{d-1}$ such that X_η is contained in the plane characterized by $(x - x_0)\cdot \mathbf{n} = 0$ for arbitrary $x_0 \in X_\eta$.*

2. Let $t \mapsto x_t := x_0 + t\mathbf{n}$ be the straight infinte line containing η. Then $x_t \in \eta$ if and only if every $x_i \in X_\eta$ is in $\mathcal{N}(x_t)$.

Proof. 1. Since all cells are polytopes, their intersections are polytopes. If η is one dimensional and characterized by equal distance to all elements of X_η, it has to be contained in a straight line and X_η has to lie in a plane orthogonal to that line. This in turn implies that $X_\eta - x_0$ span a $d-1$ dimensional space since otherwise η would be at least two-dimensional.

2. Let $t \mapsto x_t := x_0 + t\mathbf{n}$ represent the straight line discovered in 1. Then for every x_t and every $x_i \in X_\eta$ it holds: if for $x \in X \setminus X_\eta$ it holds $|x - x_t| < |x_i - x_t|$ then for every $x_j \in X_\eta$ it also holds $|x - x_t| < |x_i - x_t| = |x_j - x_t|$. Hence $x_i \in \mathcal{N}(x_t)$ if and only if $X_\eta \subset \mathcal{N}(x_t)$. □

If $x_i \in X$ and x_j is a neighbor of x_i, we denote P_{ij} the hyperplane with base $b_{ij} = \frac{1}{2}(x_i + x_j)$ and normal $\mathbf{n}_{ij} = \frac{x_j - x_i}{|x_j - x_i|}$.

Lemma 2.4. *If η is an edge of C_o then X_η contains at least $d-1$ neighbors $\tilde{x}_1, \dots \tilde{x}_{d-1}$ such that $\tilde{x}_j - x_o$ are linearly independent.*

Proof. By a scaling with a constant factor, a rotation and mirror operation, assume $\mathbf{n}$ from Lemma 2.3 satisfies $\mathbf{n} = \mathbf{e}_d = (0, \dots, 0, 1)$, $X_\eta \subset \mathbb{R}^{d-1} \times \{0\}$ and $x_o = r\mathbf{e}_1 = (r, 0 \dots, 0)$. Let also $\nu = \mathbf{n} \in \eta$ such that for some $\varepsilon > 0$ we find $\nu + (-\varepsilon, \varepsilon)\mathbf{n} \subset \eta$.

Now we write $X_\eta = (x_{\eta_i})_i$ and we study the planes P_{oi} which have their normal vector $\mathbf{n}_{oi}$ in parallel to $\mathbb{R}^{d-1} \times \{0\} \simeq \mathbb{R}^{d-1}$ and we consider their orthogonal projections $\tilde{P}_{oi}$ onto R^{d-1}. The equations $\mathbf{n}_{oi} \cdot x < 0$ describe a convex cone-like set $\mathcal{C}$ (but note that it might be rather a „bar" in one or more directions) with apex 0 at least $d-1$ different flat parts of the surface, due to the $d-1$ linear independent elements of X_η.

Since ν has positive distance to the vertices of η and the straight line connecting x_i and ν lies in C_o, we find that for sufficiently small $t > 0$ it holds $\nu_t := \nu + t\mathbf{e}_1 \in C_o$.

Now we consider the plane $P_\nu \simeq \mathbb{R}^{d-1}$ through ν orthogonal to $\mathbf{n}$ with $\nu_t \in P_\nu$ and observe that $\tilde{P}_{oi}$ form the same cone-like set $\mathcal{C}$ in this plane with $\nu_t \in \mathcal{C}$. Adding the extra dimension $\mathbf{n}$, we see that each plane of $\mathcal{C}$ corresponds locally to a set of positive $d-1$ dimensional measure close to ν, which is by the same time part of the boundary of C_o. Thus we have found at least d_1 neighbors of x_o. □

Lemma 2.5. *η is an edge of C_i in the sense of Definition 2.2 if and only if there exist $d-1$ neighbors $\tilde{\mathcal{N}} := \{x_{j_1}, \dots, x_{j_{d-1}}\}$ of x_i and $\mathbf{n} \in \mathbb{S}^{d-1}$ orthogonal to a hyperplane that contains x_i and $\tilde{\mathcal{N}}$ and one of the following conditions holds:*

1. *There exists $x \in C_i$ with $x_{j_1}, \dots, x_{j_{d-1}} \in \mathcal{N}(x)$ and $\eta = x + \mathbb{R}\mathbf{n}$.*
2. *for some $\nu_0 \in \mathcal{V}_i$ with $x_{j_1}, \dots, x_{j_{d-1}} \in \mathcal{N}(\nu_0)$ it holds $\eta = \nu_0 + [0, \infty)\mathbf{n}$.*
3. *There exists $t_1 > 0$ and $\nu_0, \nu_1 \in \mathcal{V}_i$ with $x_{j_1}, \dots, x_{j_{d-1}} \in \mathcal{N}(\nu_0) \cap \mathcal{N}(\nu_1)$ such that $\eta = \nu_0 + [0, t_1]\mathbf{n}$ and $\nu_1 = \nu_0 + t_1\mathbf{n}$.*

Interpretation. *The lemma expresses the following fact: If the nodes in X all together span a $d-1$ dimensional space, then the edges will be orthogonal to this space but of infinite extend (1.). There can also be edges emerging on a given vertex with no "counterpart", i.e. the edge tends to infinity (2.). Lastly, an edge can connect two vertices (3.).*

Definition 2.3 (Neighbored vertices $\nu_0 \sim \nu_1$)**.** We call $\nu_0, \nu_1 \in \mathcal{V}_i$ *neighbored vertices* and write $\nu_0 \sim \nu_1$ if ν_0, ν_1 satisfy 3..

Proof of Lemma 2.5. If $x \in X$ and $\tilde{\mathcal{N}} = \{x_{j_1}, \dots, x_{j_{d-1}}\}$ are neighbors of x this implies that $\{x_i\} \cup \tilde{\mathcal{N}}$ are d linear independent vectors and thus up to a sign $\pm$ the normal vector $\mathbf{n}$ of the hyperplane P_0 containing $\{x_i\} \cup \tilde{\mathcal{N}}$ is unique. Furthermore, the planes P_{ij_k} are orthogonal to P_0 and intersect in a straight line $\tilde{\eta} = x + \mathbb{R}\mathbf{n}$ which is therefore also orthogonal to P_0. It remains to show that $\eta := \tilde{\eta} \cap C_i \neq \emptyset$ is equivalent with 1., 2. or 3..

First we observe that η is convex because C_i is convex and hence η having positive one dimensional measure is equivalent with η being a ray of positive finite or infinite length. In particular, there exists $x \in \eta$ and $-\infty \leq t_- < t_+ \leq +\infty$ such that $\eta = x + [t_-, t_+]\mathbf{n}$.

It remains to show that $t_- > -\infty$ or $t_+ < +\infty$ implies $x + t_\pm \mathbf{n} \in \mathcal{V}_i$ respectively. Without restriction consider the case $t_+ < +\infty$ with the other case handled similarly. Recall that $\mathcal{N}(\cdot)$ and $N(\cdot)$ are upper semicontinuous and for $t < t_+$ but $|t - t_+|$ small enough it holds that $\mathcal{N}(x + t_+\mathbf{n}) \supseteq \mathcal{N}(x + t\mathbf{n})$. By Lemma 2.3 we find for every $\tilde{t} > t_+$ that $x_i \notin \mathcal{N}(x + \tilde{t}\mathbf{n})$, i.e. there exists $x_{\tilde{t}} \in X$ with $|x_{\tilde{t}} - (x + \tilde{t}\mathbf{n})| < |x_i - (x + \tilde{t}\mathbf{n})|$. But also for $|t_+ - \tilde{t}|$ small enough we find $\mathcal{N}(x + \tilde{t}\mathbf{n}) \subseteq \mathcal{N}(x + t_+\mathbf{n})$ This means that $N(x + t_+\mathbf{n})$ has a local maximum and $x + t_+\mathbf{n}$ is a vertex. □

Lemma 2.6. *Let $x_i \in X$ and let $\nu_0, \nu_1 \in \mathcal{V}_i$ share $d - 1$ neighbors of C_i. Then there exists an edge η of C_i in the representation 3. of Lemma 2.5 with vertices ν_0 and ν_1.*

Proof. Let $x_{j_1}, \dots, x_{j_{d-1}}$ be the $d - 1$ neighbors of C_i that are also neighbors of ν_0 and ν_1. Then $\nu_0, \nu_1 \in C_i \cap \bigcap_{k=1,\dots,d-1} P_{ij_k}$ and we conclude with Lemma 2.5. □

Lemma 2.7 (Characterization of edges emerging from a given vertex)**.** *Let $x_i \in X$ and $\nu \in \mathcal{V}_i$. Let Y be all the adjacents of x_i which share the vertex ν. Then the following are equivalent:*

1. *x_i has an edge of the form of Lemma 2.5 characterization 2. or 3. with $\nu_0 = \nu$ and given $\mathbf{n}$*
2. *there exists $\tilde{Y} = \{y_1, \dots, y_{d-1}\} \subset Y$ such that $\tilde{Y} \cup \{x_i\}$ are linear independent and such that $\mathbf{n}$ is orthogonal to the hyperplane containing x_i and $\tilde{Y}$. Furthermore, there is no $y \in Y \setminus \tilde{Y}$ with $(y - x_i) \cdot \mathbf{n} > 0$.*

Interpretation. *The meaning of this lemma is as follows: the cell C_i has a vertex ν generated by x_i and other nodes $y_1, \dots, y_K$. Then x_i and the other nodes lie on a sphere of radius $R > 0$ around ν. The above Lemma tells us now that $\eta = [0, t_1]\mathbf{n} + \nu$ is an edge of C_i emerging at ν if and only if all nodes generating this edge lie on a $d - 1$ dimensional hyperplane (not surprising) and all other nodes lie on only one side of this hyperplane. One can draw the following conclusion: The above hyperplanes corresponding to edges of C_i originating at ν form the boundary of a closed convex polytope that contains all nodes generating ν.*

For the case of a general Voronoi grid (i.e. when ν is generated by $d + 1$ nodes), this commonly known to be the simplex defined by x_i and its neighbors in ν.

Proof. Suppose 1. holds with neighbors $\tilde{Y} = \{y_1, \dots, y_{d-1}\} \subset Y$ and $\mathbf{n}$ orthogonal to the hyperplane containing x_i and $\tilde{Y}$. It remains to show that there is no $y \in Y \setminus \tilde{Y}$ with $(y - x_i) \cdot \mathbf{n} > 0$. We give a proof by contradiction.

Suppose such y exists and consider $\tilde{y} \in \tilde{Y}$. Since $(x_i - \tilde{y}) \cdot \mathbf{n} = 0$ it follows

$$(y - \nu) \cdot \mathbf{n} = (y - x_i) \cdot \mathbf{n} + (x_i - \nu) \cdot \mathbf{n} > (\tilde{y} - x_i) \cdot \mathbf{n} + (x_i - \nu) \cdot \mathbf{n} = (\tilde{y} - \nu) \cdot \mathbf{n}\,.$$

A) B)

Figure 1: Illustration of Lemma 2.7.2, i.e. the difference between A) valid and B) invalid directions $\mathbf{n}$ for an edge emerging at a vertex ν. The gray dots indicate elements of Y

We denote $y_n = (y-\nu)\cdot\mathbf{n}$ with $y_\perp = \sqrt{|y-\nu|^2 - y_n^2}$ and similarly $\tilde{y}_n$, $\tilde{y}_\perp$ for $\tilde{y}$. Then moving to $\nu_t := \nu + t\mathbf{n}$, $t>0$, implies

$$\begin{aligned}|y-\nu-t\mathbf{n}|^2 &= y_\perp^2 + (y_n-t)^2 = y_\perp^2 + y_n^2 - 2y_nt + t^2\\ &= |y-\nu|^2 - 2y_nt + t^2 = |\tilde{y}-\nu|^2 - 2y_nt + t^2\\ &< |\tilde{y}-\nu|^2 - 2\tilde{y}_nt + t^2 = |\tilde{y}-\nu-t\mathbf{n}|^2\,.\end{aligned}$$

But this would imply that y lies closer to $\nu+t\mathbf{n}$ than $\tilde{y}$ and hence for every $t>0$ it holds that $\nu+t\mathbf{n}$ is not a part of the edge, a contradiction.

Now suppose 2. holds, let P be the hyperplane containing x_i and $\tilde{Y}$ and assume $0\in P$. Since ν has the same distance to every $y\in\tilde{Y}$ and to x_i the same holds for $\nu+t\mathbf{n}$, $t\in\mathbb{R}$ and we may assume w.l.o.g. that the line $\nu+t\mathbf{n}$ hits P in $0\in P$ for some $t_0>0$. Since $(y-x_i)\cdot\mathbf{n}\le 0$ and $|x_i-\nu| = |y-\nu|$ for every $y\in Y\setminus\tilde{Y}$ we conclude that for $\tilde{t}>0$ small enough it holds $|y-\nu-t\mathbf{n}|^2 > |x_i-\nu-t\mathbf{n}|^2$ for every such y and $t\in(0,\tilde{t})$. Every other $x\in X\setminus Y$ satisfies $|x-\nu|>|x_i-\nu|$ and since the distance function is Lipschitz continuous we can modify $\tilde{t}$ such that

$$\forall x\in X\setminus Y,\ t\in(0,\tilde{t})\ :\qquad |x-\nu-t\mathbf{n}| > |x_i-\nu-t\mathbf{n}|.$$

Depending on whether equality can be obtained for some $t\ge\tilde{t}$ and some $x\in X$, we found an edge of either form 2. or 3.. □

2.3 The Raycast Lemma

We start with the following lemma, which we believe to most probably exist somewhere in the literature but we are not aware of a proper reference. Its proof is indeed rather simple and we provide it for completeness.

Lemma 2.8 (Circumcircle-Recovery-Lemma, see Figure 2)**.** *Let $\mathbb{B}_1$ be the open unit ball around $\nu=0$, let $\mathbf{n}\in\mathbb{S}^{d-1}$ be a vector in the unit sphere and let $x,z\in\mathbb{S}^{d-1}\setminus\{\mathbf{n}\}$ such that $x\cdot\mathbf{n} < z\cdot\mathbf{n}$. Writing $x_\mathbf{n} := \mathbf{n}\,(x\cdot\mathbf{n})$ and $\overline{\mathbb{B}}_0 := \overline{\mathbb{B}_{|x-x_\mathbf{n}|}(x_\mathbf{n})}$ the following holds:*
If $z'\in\mathbb{B}_1(0)\setminus\overline{\mathbb{B}}_0$ there exists $t\in(0,|x_\mathbf{n}|)$ and $R<1$ such that $\nu' := t\mathbf{n}$ satisfies

$$|z-\nu'| > R = |x-\nu'| = |z'-\nu'|\,. \tag{4}$$

Furthermore, for $0<s<|x_\mathbf{n}|$ and every $z'\in\mathbb{B}_{|x-s\mathbf{n}|}(s\mathbf{n})\setminus\mathbb{B}_1(\nu)$ there exists $0<t<s$ with $|t\mathbf{n}-z'| = |t\mathbf{n}-x|$.

Interpretation. *With regard to Figure 2 it means that if z' is located in the very big white ball but not inside the gray ball OR inside the gray ball but not inside the big white ball, then there exists ν' on the*

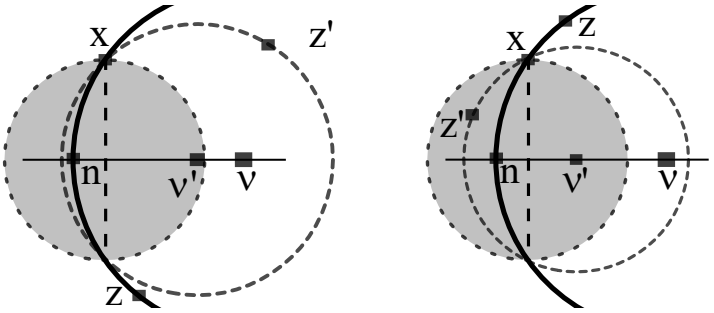

Figure 2: The two different scenarios handled in Lemma 2.8: The left picture shows case $z' \in \mathbb{B}_1(\nu) \setminus \overline{\mathbb{B}_{|x-\mathbf{n}x\cdot\mathbf{n}|}(\mathbf{n}x\cdot\mathbf{n})}$, i.e. z' in this case does not lie within the gray ball. The right picture shows the second case, where z' lies within the grey ball but outside the large original ball going through x and z. In both cases, ν' lies between ν and the vertical dotted line marking $x\cdot\mathbf{n}$.

line segment connecting ν and $x_\mathbf{n}$ such that there exists a sphere around ν' with both x and z' lying on this very same sphere. Furthermore, every z lying on the big white sphere but not inside the gray ball will definitely lie outside of the new sphere around ν'.

The lemma is named „recovery" since it can be applied in an iterative manner to recover the circumcircles in a Voronoi diagram, as we will see below.

Proof. Since $|z'|<1$ there exists a point $t\mathbf{n}$ with $|x-t\mathbf{n}| = |z'-t\mathbf{n}|$. This point minimizes

$$t \mapsto \left((x-t\mathbf{n})^2 - (z'-t\mathbf{n})^2\right)^2 = (1+|z'|^2 - 2(x-z')\cdot\mathbf{n}t)^2$$

and a short calculation shows that $t = \frac{1-|z'|^2}{2(x-z')\cdot\mathbf{n}} > 0$ is the only solution. We next observe that $t < x\cdot\mathbf{n}$ is equivalent with

$$\begin{aligned} & 2(x\cdot\mathbf{n})^2 - 2(x\cdot\mathbf{n})(z'\cdot\mathbf{n}) > 1-|z'|^2 \\ \Leftrightarrow \quad & (x\cdot\mathbf{n}-z'\cdot\mathbf{n})^2 - (z'\cdot\mathbf{n})^2 > 1-(x\cdot\mathbf{n})^2 \\ \Leftrightarrow \quad & |z'-\mathbf{n}\,x\cdot\mathbf{n}|^2 > |x-\mathbf{n}\,x\cdot\mathbf{n}|^2 \end{aligned}$$

and the last line is equivalent with $z' \notin \overline{B}_0$.

It remains to prove (4). Let $x = x_\mathbf{n} + \xi_x$, $z = z_\mathbf{n} + \xi_z$, where $\mathbf{n}\cdot\xi_i = 0$ and where $|x_\mathbf{n}| > |z_\mathbf{n}|$. It holds

$$(x-t\mathbf{n})^2 < (z-t\mathbf{n})^2 \qquad \Leftrightarrow \qquad |\xi_x|^2 + (|x_\mathbf{n}|-t)^2 < |\xi_z|^2 + (|z_\mathbf{n}|-t)^2$$

Using $|\xi_i|^2 + |i_\mathbf{n}|^2 = 1$, $i = x, z$, the latter condition is equivalent with $|x_\mathbf{n}| > |z_\mathbf{n}|$ and hence the first claim follows.

For the second claim we argue in a similar way. □

Lemma 2.9 (Raycast Lemma)**.** *Let $\nu = 0$, $\mathbf{n}\in\mathbb{S}^{d-1}$, $r>0$ and let every $x\in X$ with $x\cdot\mathbf{n} = 0$ satisfy $|x-\nu|\geq r$. If $x_1\in X$ with $x_1\cdot\mathbf{n}=0$ and $|x_1-\nu| = r$ consider the following iteration:*

1. *chose z_1 as nearest neighbor of $\nu_1 = \nu$ satisfying $z_1\cdot\mathbf{n} > 0$. If z_1 does not exists, set $t_1 = \infty$ and terminate*
2. *Given $k\geq 1$ search $\nu_k' = \nu + t_k\mathbf{n}$ through the equation*

$$|\nu_k' - x_1|^2 = |z_k - \nu_k'|^2 \qquad \Leftrightarrow \qquad |\nu - x_1|^2 - 2t_k\mathbf{n}\cdot(\nu-x_1) = |\nu-z_k|^2 - 2t_k\mathbf{n}\cdot(\nu-z_k)\,. \quad (5)$$

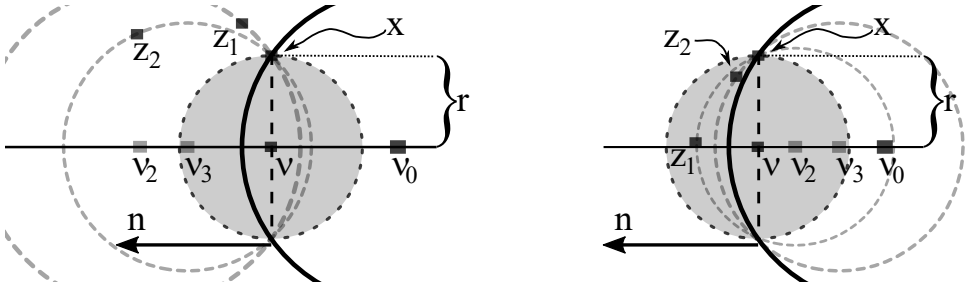

Figure 3: The idea of the Raycast Algorithm from Lemma 2.9. In application, we will start with a vertex ν_0 and take the generators of an edge emerging ν_0 with direction $\mathbf{n}$ and generated by nodes which correspond to x. The algorithm then starts with $\nu = \nu_1$ and terminates in ν_3 on both pictures.
The new vertex will lie on the opposite site of plane through all x orthogonal to $\mathbf{n}$. In the first case, the new generator will lie outside the gray ball and the new vertex will be on the opposite side of the plane. In the second case, the new vertex will lie inside the gray ball and the new vertex will lie on the same side of the plane as ν_0.

3 *Set $\nu_{k+1} = \nu'_k$ and search for its nearest neighbor z_{k+1} of ν_{k+1} which lies within a ball of radius $|\nu'_k - z_k|$. If $|\nu'_k - z_k| = |\nu'_k - z_{k+1}|$ set $\tilde{\nu} = \nu_{k+1}$, $x = z_k$ and stop, otherwise continue with step 2. for $k+1$.*

Then the algorithm terminates either in Step 1 with $t_1 = \infty$ or in Step 3 with $\tilde{\nu} \neq \nu$ and some $x \in X$ satisfying $x \cdot \mathbf{n} \neq 0$. If $\tilde{X} \subset X$ is the list of generators such that $\tilde{x} \in \tilde{X}$ if and only if $\tilde{x} \cdot \mathbf{n} = 0$ and $|\tilde{x} - \nu| = r$ there exists $\tilde{r} > r$ such that for every $\tilde{x} \in \tilde{X}$ it holds $|\tilde{x} - \tilde{\nu}| = |x - \tilde{\nu}| = \tilde{r}$ and $\mathcal{N}(\tilde{\nu}) = \tilde{X} \cup \{x\}$.

Proof. Assume that the algorithm does not stop at Step 1.

In the first case, $\overline{\mathbb{B}_{|x_1|}(0)} \cap X = \emptyset$ and hence $z_k \notin \overline{\mathbb{B}_{|x_1|}(0)}$ but $z_k \cdot \mathbf{n} > 0$. In this case the first part of Lemma 2.8 yields that for every k and $|z_k - \nu'_k| = |x_1 - \nu'_k| < |x_1 - \nu'_{k-1}|$ is a decreasing sequence with $\nu_{k+1} \cdot \mathbf{n} > 0$. The algorithm terminates due to finiteness of X.

In the second case we face the situation $z_1 \in \overline{\mathbb{B}_{|x_1|}(0)}$. The second part of Lemma 2.8 yields $\nu_2 \cdot \mathbf{n} < 0$ and also subsequently $z_k \in \overline{\mathbb{B}_{|x_1|}(0)}$ and $\nu_{k+1} \cdot \mathbf{n} < 0$. The algorithm also in this case will terminate after finite steps.

The remaining properties hold by the termination criteria. □

2.4 Iterative search for vertices along edges

Definition 2.4 ((lower dimensional) Facets)**.** Let $\tilde{X} \subset X$ such that such that $E(\tilde{X}) = \bigcap_{x_j \in \tilde{X}} C_j$ satisfies $\mathcal{H}^k(E(\tilde{X})) > 0$ and in case $k < d$ also $\mathcal{H}^{k+1}(E(\tilde{X})) = 0$. Then we call $E(\tilde{X})$ a k-facet.

Remark. The two extreme cases are 0-facets, which are vertices defined by $d+1$ elements, and d-facets which are defined by only one point and are hence the full Voronoi cell of the given point.

Lemma 2.10. *Let $\tilde{X}, \tilde{Y} \subset X$ and such that both $E(\tilde{X})$ and $E(\tilde{Y})$ are k facets and such that $E(\tilde{X}) \cap E(\tilde{Y})$ has positive k dimensional Hausdorff measure. Then $E_0 := E(\tilde{X}) = E(\tilde{Y}) = E(\tilde{X} \cup \tilde{Y})$ and the k-dimensional plane defined by E_0 is orthogonal to a $d-k$ dimensional plane containing $\tilde{X} \cup \tilde{Y}$.*

Proof. The case $k = 1$ is a direct consequence of Lemma 2.7 which tells us that edges are closed convex intervals on finite lines and we further on focus on the case $k > 1$.

Since $E(\tilde{X}), E(\tilde{Y})$ are closed, $E(\tilde{X}) \neq E(\tilde{Y})$ implies that $E(\tilde{X}) \setminus E(\tilde{Y}) \cup E(\tilde{Y}) \setminus E(\tilde{X})$ is not countable. However, they are both k dimensional plane objects and since $E(\tilde{X}) \cap E(\tilde{Y})$ has positive measure their supporting planes must coincide.

Let $x \in E(\tilde{Y}) \cap E(\tilde{X})$ and let w.l.o.g. $y \in E(\tilde{Y}) \setminus E(\tilde{X})$. Denoting $\mathbf{n} := y - x$ we find for $\eta(t) = x + t\mathbf{n}$ that $\eta(0) = x$, $\eta(1) = y$ and there is some t_0 with $\eta(t) \in E(\tilde{Y}) \cap E(\tilde{X})$ for $t < t_0$ and $\eta(t) \in E(\tilde{Y}) \setminus E(\tilde{X})$ for $t > t_0$. Similar to the proof of Lemma 2.5 we can argue from here that $\eta(t_0)$ is a vertex. But since x and y can be chosen from a k-dimensional set, the set of vertices $\eta(t_0)$ forms a $k-1$ dimensional set, a contradiction to the characterization of vertices.

Now let $\tilde{Z} = \tilde{X} \cup \tilde{Y}$. Since every $x \in E_0$ and every $z_1, z_2 \in \tilde{Z}$ satisfy $|x - z_1| = |x - z_2|$ it holds that $\tilde{Z}$ lies within a $d-k$-dimensional plane orthogonal to E_0. □

Lemma 2.11 (Descent-Lemma)**.** *Let X contain $d+1$ linearly independent points. Then for every k-facet $E(\tilde{X})$ with $k > 0$ the following holds: For $\mathcal{H}^k$ almost every $\nu \in E(\tilde{X})$ there exists $x_j \in X$, $\alpha \in \mathbb{R}$ and $\mathbf{n} || E(\tilde{X})$ such that*

$$\tilde{\nu} = \nu + \alpha \mathbf{n} \in E(\tilde{Y}), \qquad \tilde{Y} = \tilde{X} \cup \{x_j\}$$

and $E(\tilde{Y})$ is a $k-1$ facet.

Interpretation. *One of the major problems of our algorithm is recovering an initial vertex from where to start building the grid. The above lemma states that we can start in x_i, then walk straight to the nearest neighbor and hit the interface in the midpoint of the line segment. However, from there we can take almost surely any orthogonal direction, knowing that the Raycast algorithm will yield a third element of X and a point $\nu \in C_i$ that has equal distance to all three elements. The set of points having equal distance to all three elements has dimension $d-2$. We can iterate this algorithm until we found a vertex of dimension 0.*

Proof. If $k > 1$ Lemma 2.10 shows that we can assume that $\tilde{X}$ is so large that it contains all sets capable to generate $E(\tilde{X})$. If $k = 1$ it is clear that all generators of a given 1-facet have to lie on one single plane orthogonal to the facet.

Now let $\nu \in E(\tilde{X})$ and $\mathbf{n}$ be orthogonal to the $d-k$ dimensional plane that contains $\tilde{X}$. However, $\mathbf{n}$ defines a full $d-1$ dimensional plane, which is not uniquely defined in case $k < d-1$ and we first assume we can chose $\mathbf{n}$ such that there exists $x' \in X$ on the opposite side of the plane than ν. But then we can apply the Raycast algorithm of Lemma 2.9 to find $\tilde{\nu}$ as claimed.

If the above choice of a plane and $\mathbf{n}$ is not possible, we observe in a first step of the Raycast, that $t_1 = \infty$. We make use of this fact to „walk" along the ray to some ν' the other side of the plane and start the Raycast in ν'. □

Lemma 2.12 (Connectivity Lemma)**.** *Let $\nu, \tilde{\nu} \in \mathcal{V}_i$. Then there exist vertices N vertices $\nu_1 = \nu, \dots \nu_N = \tilde{\nu} \in \mathcal{V}_i$ such that for every $j = 1, \dots, N-1$ it holds $\nu_j \sim \nu_{j+1}$. In particular, there exists a path from ν to $\tilde{\nu}$ among edges of $\mathcal{V}_i$.*

Proof. We take a straight line γ_0 connecting ν and $\tilde{\nu}$. We project this line within a $d-1$ dimensional plane to the boundary of $\mathcal{V}_i$ obtaining a new pathγ_1 as a family of straight lines within a sequence $E_1, \dots, E_K$ of $d-1$ facets which are interfaces with neighboring cells and $\nu \in E_1$, $\tilde{\nu} \in E_K$. We may now shift the intersection of γ_1 with $E_1 \cap E_2$ using the Descent Lemma to a vertex of $\mathcal{V}_i$ still lying

within $E_1 \cap E_2$. Using this approach we may thus finally assume that γ_1 is a path among $d-1$ facets which skips crosses from facet to facet only in vertices of $\mathcal{V}_i$. We iterate this process going from $d-1$ to $d-2$ to $\dots$ to 1-facets which are edges. □

2.5 Localized convex cone algorithm: edge-iteration over degenerate vertices

We will now discuss how to identify the edges emerging at a vertex and how to iterate over these edges in a cost- effective way. In the first case, the generators are in general position and a vertex is defined by exactly $d+1$ generators.

Lemma 2.13. *Let ν be a vertex defined by $x_1, \dots, x_{d+1}$. Then there are exactly $d+1$ edges emerging at ν each characterized by dropping one of the points x_i. The direction of each such edges is orthogonal to the plane containing the remaining d points.*

Proof. This is a direct consequence of Lemma 2.7. □

Lemma 2.13 implies that $d+1$ edges are found in $d+1$ steps provided the generators are in general position. In case the nodes are *not* in general position, i.e. a vertex ν is generated by $x_1, \dots, x_{d+k}$, $k > 1$, Lemma 2.7 still tells us that we only need to iterate over all subsets $x_{i_1}, \dots, x_{i_d}$ and verify condition 2 of that lemma.

However, consider the case that $N = n^d$, $(x_i)_{i=1,\dots,n^d} = \{1, \dots, n\}^d$. Generators in such a diagram have $2d$ neighbors and $3^d - 1$ adjacents as well as 2^d vertices and each of these vertices has 2^d generators. If we want to verify condition 2 of Lemma 2.7 for all $x_{i_1}, \dots, x_{i_d}$ of a vertex with 2^d generators, this takes $\begin{pmatrix} 2^d \\ d \end{pmatrix} \approx (2^d)^d/d!$ such verifications. In 5 dimensions, this corresponds to 201376 candidates that have to be verified, in 8 dimensions already 409663695276000. Luckily, we can reduce the amount of iterations significantly. For this, we will locally identify some of the tangential planes to a Voronoi cell and extract iteratively for a given vertex the *locally essential* generators. It is then sufficient to iterate only over subsets of these essential generators.

Definition 2.5 (Infinite cone)**.** Let $A \subset \mathbb{R}^d$ be a set and let $x_0 \in \mathbb{R}^d$ be a point. Writing $\overline{\mathrm{conv}} B$ for the closed convex hull of a set $B \subset \mathbb{R}^d$ we call

$$\mathcal{C}(x_0, A) := \overline{\mathrm{conv}}\{t(x - x_0) + x_0 \ : \ t \geq 0, x \in A\}$$

the infinite cone of A with apex x_0.

Lemma 2.14. *Let $d \geq 2$ and let P_d be a convex polytope generated by the finite set of points X_d, i.e. $P_d = \overline{\mathrm{conv}} X_d$. Let $0 \in X_d$ and let $\tilde{X}_d \subset X_d$ be such that $P_d \subset \mathcal{C}(0, \tilde{X}_d)$ but $P_d \not\subset \mathcal{C}(0, \tilde{X}_d \setminus \{\tilde{x}\})$ for every $\tilde{x} \in \tilde{X}_d$. Then $\tilde{X}_d$ is well defined.*

Furthermore, assume that $\mathbb{R}^{d-1} \simeq \mathbb{R}^{d-1} \times \{0\}$ is tangential to P_d and that $P_{d-1} = P_d \cap \mathbb{R}^{d-1}$ has positive measure and let $X_{d-1} := P_d \cap \mathbb{R}^{d-1}$: If $d > 2$ it holds $x \in \tilde{X}_d \cap \mathbb{R}^{d-1}$ if and only if $x \in \tilde{X}_{d-1}$. If $d = 2$ then $\tilde{X}_2$ contains the two elements of $X_2 \setminus 0$ that share the largest angle.

Interpretation. *Let $x_0 = 0 \in X$ with Voronoi cell C_0. If ν is a vertex and X_d are the generators of ν, then Lemma 2.7 tells us that the edges emerging at ν correspond to the surface elements of P_d. Moreover, the source elements of P_d containing x_0 correspond to the edges emerging at ν along C_0. Since the set $\tilde{X}_d$ corresponds to the minimal set of edges that are needed to engulf P_d by a cone with apex x_0, we find a one to one correspondence between the surface elements of $\mathcal{C}(0, \tilde{X}_d)$ and*

the surface elements of P_d containing $x_0 = 0$. If $\tilde{X}_d$ is significantly smaller than X_d, the iteration over subsets of d elements of $\tilde{X}_d$ will be tremendously faster.

For a more concrete example, let $\nu = (0.5, 0.5, 0.5)$, $x_0 = 0$ and $X_3 = \{(a_1, a_2, a_3) : a_i \in \{0, 1\}\}$. Then

$$\tilde{X}_3 = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$$

has only three elements while X_d has 8. This effect increases exponentially in d.
While this example suggests that $\tilde{X}_d$ has always d elements, numerical experiments show that this is not true in general. The author was able to generate in 5d examples of $\tilde{X}_5$ with up to 9 elements.

Proof of Lemma 2.14. Step 1: If $\tilde{x} \in \tilde{X}_d$ then $\tilde{x} \notin \mathcal{C}(0, \tilde{X}_d \setminus \{\tilde{x}\})$. Proving this by contradiction, let $\tilde{x} \in \mathcal{C}(0, \tilde{X}_d \setminus \{\tilde{x}\})$. Then by convexity of $\mathcal{C}(0, \tilde{X}_d \setminus \{\tilde{x}\})$ this leads to $\mathcal{C}(0, \tilde{X}_d \setminus \{\tilde{x}\}) = \mathcal{C}(0, \tilde{X}_d)$.

Step 2: In case $d = 2$ convexity implies that $\tilde{X}_2$ is unique for a given pair (P_2, X_2) and $\tilde{X}_2$ contains the two elements of $X_2 \setminus 0$ that share the largest angle.

Step 3: Assume the uniqueness of $\tilde{X}_{d-1}$ is proven for the projection P_{d-1} and X_{d-1}.

Step 3a: We claim that $\tilde{X}_d \cap P_{d-1} \supset \tilde{X}_{d-1}$:

Since every $x \in P_{d-1}$ is a convex combination of elements in $\tilde{X}_d$ and $P_{d-1} \subset \partial P_d$ it follows that every $x \in P_{d-1}$ is a convex combination of elements in $R^{d-1} \cap \tilde{X}_d$. On the other hand we can start from $X_d \cap P_{d-1}$ to calculate a suitable $\tilde{X}_{d-1}$. Since $\tilde{X}_{d-1}$ is unique, the claim 3a follows.

Step 3b: Let $x_0 = 0$ and let $P_i \simeq \mathbb{R}^{d-1}$ be a family of hyperplanes that are parallel to one of the planes of P_d emerging at x_0. Let $P_{d-1,i} := \partial P_d \cap P_i$ and $\tilde{X}_{d-1,i} := X_d \cap P_i$ be the corresponding subsets of $X_{d-1,i} \subset X_d$. By definition of $\mathcal{C}(x, A)$ and convexity we have

$$\mathcal{C}(0, X_d) = \mathcal{C}(0, \tilde{X}_d) = \mathcal{C}(0, \bigcup_i \mathcal{C}(0, \tilde{X}_{d-1,i})) = \mathcal{C}(0, \bigcup_i \tilde{X}_{d-1,i}) \,.$$

Hence $\tilde{X}_d \subset \hat{X}_d := \bigcup_i \tilde{X}_{d-1,i}$. On the other hand, $\hat{X}_d$ cannot be reduced because of the necessary condition from Step 3a and the uniqueness of $\tilde{X}_{d-1}$. This implies $\tilde{X}_d = \hat{X}_d$ and by the same time uniqueness as well as $x \in \tilde{X}_d \cap \mathbb{R}^{d-1}$ if and only if $x \in \tilde{X}_{d-1}$. □

Lemma 2.15 (Local iterative convex cone algorithm)**.** *Let C_0 be a Voronoi cell around $x_0 = 0$ and $\nu = (x_0, x_1, \ldots, x_N)$ be a vertex of C_0 and consider the following algorithm.*

1 *Set $P_d = X_d$ and mark all nodes as* `checked=false` *and* `blocked=false`

2 *chose an „unchecked" x_i and try to find a flat surface of C_0 that contains x_i and „unblocked" generators.*

3 *if this is not possible, set x_i as* `checked=true` *and* `blocked=true`

4 *else*

  4.1 *if $d > 2$ pass that interface to the same lower dimensional algorithm.*

  4.2 *else mark the outer two nodes (those with the biggest angle between them) as* `blocked=false` *and the others as* `blocked=true`

5 *mark all involved nodes as* `checked=true`

6 *if not all nodes are „checked", restart at 2.*

Then this algorithm returns all elements of $\tilde{X}_d$ as `blocked=false` *and the others as* `blocked=true`*.*

Proof. This is an immediate consequence of Lemma 2.14. □

Lemma 2.16. *Let C_0 be a Voronoi cell around $x_0 = 0$ and $\nu = (x_0, x_1, \dots, x_N)$ be a vertex of C_0 and consider the following algorithm.*

1 *Let X_d be all generators of ν and let $\tilde{X}_d$ be given by the algorithm from Lemma 2.15.*

2 *for each subset $\tilde{\boldsymbol{\eta}} = \{x_1, \dots, x_{d-1}\} \subset \tilde{X}_d$ with exactly $d-1$ elements, verify that*

- *$\{x_1, \dots, x_{d-1}\}$ are linear independent*
- *all elements of X_d lie on the same side of the $d-1$ dimensional plane spanned by $\tilde{\boldsymbol{\eta}}$ and x_0.*

and if both conditions are satisfied, calculate all elements of X_d that lie in the plane given by $\tilde{\boldsymbol{\eta}} \cup \{x_0\}$ and the normal vector $\mathbf{u}$. If these elements have numbers $\boldsymbol{\eta} \subset \{1, \dots, N\}$ store $(\boldsymbol{\eta}, \mathbf{u})$ as edge.

Then this algorithm finds all edges of C_0 emerging at ν.

Proof. X_d are the generators of ν and form a convex polytope around ν. By Lemma 2.15 the first step yields $\tilde{X}_d$ for x_0 with the properties given in Lemma 2.14 which also ensures that every plane of P_d that intersects with x_0 is generated by elements of $\tilde{X}_d$ and x_0. Part 2 of Lemma 2.7 says that every edge of C_0 emerging at ν is characterized as plane on ∂P_d emerging at x_0. Thus the above algorithm will yield all edges of C_0 emerging at ν. □

3 The algorithms and data structures

We will now introduce the concepts of the data structures and the algorithms to generate a full Voronoi mesh from a given set of nodes X. We will provide both the algorithms and proofs that they converge.

3.1 The fundamental data structure

If $X = (x_i)_{i=1,\dots,N}$ we recall the notation of Definition 1.1 and represent a **vertex** $\nu \in \mathcal{V}_i$ as $\nu = (\boldsymbol{\sigma}, \boldsymbol{r})$, where $\boldsymbol{\sigma} = (\sigma_{j_1}, \dots, \sigma_{j_k})$ is a ordered subset of $\{1, \dots, N\}$ with $\mathcal{N}(\nu) = \{x_{\sigma_{j_1}}, \dots, x_{\sigma_{j_k}}\}$ and where $\boldsymbol{r}$ stores the actual coordinates of ν.

In a similar way, we store an **edge** $\eta = (\boldsymbol{\eta}, \mathbf{u})$, where $\boldsymbol{\eta} = (\eta_1, \dots, \eta_k)$ is a sorted list of the numbers of all the generators of the edge and $\mathbf{u}$ is the orientation of the edge. In view of Definition 2.2 or Lemma 2.5 it appears that we should also provide the list of neighbors (which are actually unknown at the instance of computation) or at least one vertex of $\boldsymbol{\eta}$. However, this one vertex will implicitly be known during the calculations.

Given $x_i \in X$ with $\mathcal{V}_i$ and accounting for the fact that every $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i$ satisfies $i \in \boldsymbol{\sigma}$ and that $\boldsymbol{\sigma}$ is ordered we split

$$\mathcal{V}_i = \mathcal{A}_i \cup \mathcal{B}_i, \quad \text{where} \quad \mathcal{A}_i = \{(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i \mid \sigma_1 = i\} \quad \text{and} \quad \mathcal{B}_i = \{(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i \mid \sigma_1 < i\},$$

Furthermore we store in $\mathcal{B}^\infty$ the edges of type 2. in Lemma 2.5. Then our mesh consists of

$$\mathcal{M} = (X, \mathcal{A}, \mathcal{B}, \mathcal{B}^\infty) .$$

3.2 Extended data structure with boundary planes

For $i \in 1, \dots, K$ let $P_i = (b_i, n_i)$ be a pair of a base vector b_i and normal vector n_i and let

$$\Omega := \left\{ x \in \mathbb{R}^d \,|\, \forall i \,:\, (x - b_i) \cdot n_i < 0 \right\} .$$

Then Ω is convex and we assume that $X \subset \Omega$. Note that Ω might be bounded in all directions or partially unbounded in some directions. Our domain in which we work thus has two representations: Ω and $(P_i)_{i=1,\dots,K}$ which we group in the expression $\mathcal{O}$ and our new mesh data structure becomes:

$$\mathcal{O} = \mathcal{O}\left((P_i)_{i=1,\dots,K}\right) = \left((P_i)_{i=1,\dots,K}, \Omega\right) , \qquad \mathcal{M} = (X, \mathcal{O}, \mathcal{A}, \mathcal{B}, \mathcal{B}^\infty) .$$

We take account of the boundaries in the Raycast algorithm using virtual, mirrored points. In particular, we denote for $j \in \{N+1, \dots, N+K\}$ and $i \in \{1, \dots, N\}$

$$x_j^{(i)} := x_i + 2\langle x_i - b_{j-N} \,,\, n_{j-N} \rangle \, n_{j-N} . \tag{6}$$

When working on the cell i, we will denote $x_j = x_j^{(i)}$ whenever this is convenient and no confusion is possible. Furthermore, we extend X by entries x_j, $j > n$ which are either temporarily given as some $x_j^{(i)}$ or otherwise remain undefined. This makes sense in view of the following result.

Lemma 3.1. *Given a set of nodes $X \subset \mathbb{R}^d$ and Ω, $(P_j)_{j=1,\dots,K}$ as above we call $x_i \in X$ a boundary node if $C_i \cap (\mathbb{R}^d \setminus \Omega) \neq \emptyset$ and define $C_{\Omega,i} := C_i \cap \overline{\Omega}$. Given i we extend X to $\tilde{X}_i$ by $x_{j+N} := x_{j+N}^{(i)}$, $j = 1, \dots, K$, according to* (6) *and define $\tilde{C}_i$ as the Voronoi cell of x_i w.r.t. $\tilde{X}_i$. Then it holds $\tilde{C}_i = C_{\Omega,i}$.*

Proof. For every plane P_j such that $P_j \cap \partial C_{\Omega,i}$ has positive $d-1$ dimensional measure, we recall that $x_{j+N}^{(i)}$ is the mirror of x_i at P_j and hence P_j can be equally expressed by the base $\tilde{b}_j := \frac{1}{2}(x_{j+N}^{(i)} + x_i)$ and normal $\tilde{n}_j = \frac{x_{j+N}^{(i)} - x_i}{|x_{j+N}^{(i)} - x_i|}$. But this implies that $P_j \cap \partial C_{\Omega,i}$ is the Voronoi interface between $x_{j+N}^{(i)}$ and x_i. Iterating over all $j \in \{1, \dots, K\}$ gives the claim. □

Remark.

- It should be mentioned that most of the cells will typically not share any of the boundary planes P_i. Hence in the realization of the code, the boundary nodes $x_j^{(i)}$ are calculated on the fly and deleted after the iteration over the i-th cell.
- In case the cell C_i touches the boundary plane P_{j-N} and we locally consider $x_j = x_j^{(i)}$ as part of the mesh, we can w.l.o.g. call $x_j = x_j^{(i)}$ a neighbor of x_i, since all of the above findings for neighbors hold true.

3.3 The exhaustive search algorithm

We will now introduce the main algorithm. It is meant to work „from zero" but can also be run if parts of the mesh are already known (reflected in the assumption $\mathcal{A}_i \neq \emptyset$). The strategy is to iterate over all generators x_i, every vertex $\nu \in \mathcal{V}_i$ and finally over all edges emerging at ν while skipping edges that are already known.

Algorithm 3.1. `Voronoi`$(X, (P_i)_{i=1,\dots,K})$

1 Set up $\mathcal{O}((P_i)_{i=1,\dots,K})$ and create a data structure $\mathcal{M} = (X, \mathcal{O}((P_i)_{i=1,\dots,K}), \mathcal{A}, \mathcal{B}, \mathcal{B}^\infty)$ for storage of results.

2 For each i in $1, \dots, N$ call `ExploreCell`$(\mathcal{M}, i)$

Algorithm 3.2. `ExploreCell`$(\mathcal{M}, i)$

1 Initialize an empty queue list $\mathcal{Q}$ of vertices.

2 Initialize an empty list $\mathcal{E}$ of type (edge=>count)

3 Initialize a `Boolean` list $(\alpha_j)_{j=1,\dots,K}$ of `false` entries.

4 If $\mathcal{V}_i = \mathcal{A}_i \cup \mathcal{B}_i$ is not empty, run through all $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i$: if $\boldsymbol{\sigma}$ contains $\sigma_j > N$ call `ActivateBoundary`$(\mathcal{M}, \alpha, i, \sigma_j)$

5 Cases:

- If $\mathcal{V}_i = \mathcal{A}_i \cup \mathcal{B}_i$ is empty:
  do $(\boldsymbol{\sigma}, \boldsymbol{r})$ =`Descent`$(\mathcal{M}, i)$ and do `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \boldsymbol{\sigma}, \boldsymbol{r}, i)$.
- Else: for every $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i$ do `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \boldsymbol{\sigma}, \boldsymbol{r}, i$,`init=true`).

6 While $\mathcal{Q}$ is not empty

6.1 `pop!` first element $(\boldsymbol{\sigma}, \boldsymbol{r})$ from $\mathcal{Q}$ (take first element of $\mathcal{Q}$ and delete it from the list)

6.2 call `ExploreVertex`$(\mathcal{Q}, \mathcal{E}, \mathcal{M}, i, \boldsymbol{\sigma}, \boldsymbol{r})$

6.3 $(\boldsymbol{\sigma}, \boldsymbol{r}) \notin \mathcal{A}_i$ store vertex $(\boldsymbol{\sigma}, \boldsymbol{r})$ to $\mathcal{A}_i$ and if $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_n)$ store vertex $(\boldsymbol{\sigma}, \boldsymbol{r})$ to every $\mathcal{B}_{\sigma_j}$, $j = 2, \dots, n$, provided $(\boldsymbol{\sigma}, \boldsymbol{r}) \notin \mathcal{B}_j$.

Algorithm 3.3. `ActivateBoundary`$(\mathcal{M}, \alpha, i, \sigma_j)$
If $\alpha_{\sigma_j - N}$=`false` then

1 Calculate $x^{(i)}_{\sigma_j}$ according to (6) and store it in position σ_j of X: $x_{\sigma_j} = x^{(i)}_{\sigma_j}$.

2 Set $\alpha_{\sigma_j - N}$ =`true`.

Algorithm 3.4. `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \boldsymbol{\sigma}, \boldsymbol{r}, i$,[`optional: init=false`])

1 If $(\boldsymbol{\sigma}, \boldsymbol{r}) \notin \mathcal{A}_i \cup \mathcal{Q}$ or `init==true` then push $(\boldsymbol{\sigma}, \boldsymbol{r})$ to $\mathcal{Q}$, else STOP.

2 Call `Edges`$(\boldsymbol{\sigma}, \boldsymbol{r}, \mathcal{M}, i)$ to get the set $E(\boldsymbol{\sigma}, i) = (\boldsymbol{\eta}_j, \_)_j$ of all sorted edges $\boldsymbol{\eta}_j \subset \boldsymbol{\sigma}$.

3 For every $\boldsymbol{\eta}_j = (\eta_1, \dots)$ with smallest entry $\eta_1 = i$:

- If $\mathcal{E}$ has an entry (η, c), increase c by 1
- Else: `push!` an entry $(\eta, 1)$ to $\mathcal{E}$

Algorithm 3.5. `ExploreVertex`$(\mathcal{Q}, \mathcal{E}, \mathcal{M}, i, \boldsymbol{\sigma}, \boldsymbol{r})$

1. Call `Edges`$(\boldsymbol{\sigma}, \boldsymbol{r}, \mathcal{M}, i)$ to obtain a list $E(\boldsymbol{\sigma}, i) = (\boldsymbol{\eta}_j \mathbf{u}_j)_j$.
2. Iterate through all sorted edges $(\boldsymbol{\eta}, \mathbf{u}) \in E(\boldsymbol{\sigma}, i)$, $\boldsymbol{\eta} = (\eta_1, \dots, \eta_d)$ which stem from of $\boldsymbol{\sigma}$ with the smallest element given by $\eta_1 = i$:
   2.1. Load (η, c) from $\mathcal{E}$. If such entry exists and $c \geq 2$ then skip the next step.
   2.2. Call `WalkRay`$(\mathcal{Q}, \mathcal{E}, \mathcal{M}, \alpha, i, \eta, \boldsymbol{r}, \mathbf{u})$.

Algorithm 3.6. `Edges`$(\boldsymbol{\sigma}, \boldsymbol{r}, \mathcal{M}, i)$

- If length(σ)==$d+1$ return all subsets of σ of length d and with miminal entry i and the corresponding directions of the edges as described by Lemma 2.13.
- else use the algorithm given in Lemma 2.16 for $x_0 = x_i$ and $\nu = \boldsymbol{r}$ and return those edges.

Algorithm 3.7. `WalkRay`$(\mathcal{Q}, \mathcal{E}, \mathcal{M}, \alpha, i, \eta, \boldsymbol{r}, \mathbf{n})$

1. resolve $\mathcal{M} = (X, ((P_i)_{i=1,\dots,K}, \Omega), \dots)$. In order to keep in mind that X is extended by „activated" cells, we also write
$$\tilde{X}_i = (X_j)_{j=1,\dots,N} \cup \{x_j^{(i)} \in X \mid j > N, \alpha_{j-N} = \texttt{true}\}$$
2. Call `RayCast`$(\mathcal{M}, \boldsymbol{r}, \mathbf{n}, i)$ and
   2.1. Either obtain a vertex $(\tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{r}})$, defined by points in $\tilde{X}_i$.
      2.1.1. If $\tilde{\boldsymbol{r}} \in \overline{\Omega}$ call `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{r}}, i)$
      2.1.2. Else the line connecting $\boldsymbol{r}$ and $\tilde{\boldsymbol{r}}$ intersects with $\partial\Omega$ at plane P_j in coordinates $\tilde{R}$. We call `ActivateBoundary`$(\mathcal{M}, \alpha, i, N+j)$ to update $\tilde{X}_i$. Afterwards calculate $\tilde{\boldsymbol{\sigma}}$ as the list of all nearest neighbors of $\tilde{R}$ in $\tilde{X}_i$ (`inrange`-search!).
      Call `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \tilde{\boldsymbol{\sigma}}, \tilde{R}, i)$
   2.2. otherwise try to calculate an intersection of $\boldsymbol{r} + t\mathbf{n}$, $t > 0$ with $\partial\Omega$:
      2.2.1. If existent, this point is given through $\tilde{\boldsymbol{r}} = \boldsymbol{r} + t_0\mathbf{n}$, $t_0 > 0$, and P_j with the property $(\tilde{\boldsymbol{r}} - b_j) \cdot n_j = 0$ and for every $k \neq j$ it holds $(\tilde{\boldsymbol{r}} - b_k) \cdot n_k \leq 0$. Call `ActivateBoundary`$(\mathcal{M}, \alpha, i, N+j)$, update $\tilde{X}_i$ and afterwards calculate $\tilde{\boldsymbol{\sigma}}$ as the list of all nearest neighbors of $\tilde{\boldsymbol{r}}$ in $\tilde{X}_i$ (`inrange`-search!).
      Call `QueueVertex`$(\mathcal{Q}, \mathcal{E}, \tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{r}}, i)$
      2.2.2. Otherwise we found an edge of type 2. in Lemma 2.5. we store $(\boldsymbol{r}, \mathbf{n})$ to $\mathcal{B}^\infty$

Theorem 3.2 (Exhaustive search)**.** *Given a set of nodes $X \subset \mathbb{R}^d$ and a domain $\Omega \supset X$ described by planes $(P_j)_{j=1,\dots,K}$ the Algorithm* `Voronoi`$(X, (P_j)_{j=1,\dots,K})$ *is exhaustive in $\mathbb{R}^d$. In particular, if X contains a subset of d linear independent nodes, then almost surely for every $x_i \in X$ all vertices of $\tilde{C}_i$ will be found, where $\tilde{C}_i = C_i \cap \overline{\Omega}$ is defined in Lemma 3.1.*

Proof. Let us first assume that $K = 0$, i.e. there are no planes. The algorithm iterates over all cells. For each cell C_i it checks whether there are known vertices. If not it calls `Descent`, which yields a vertex due to Lemma 2.11. After the first vertex is found, Lemma 2.12 combined with the convergence of `RayCast` by Lemma 2.9 ensures that the algorithm finds all vertices of C_i if we apply `RayCast` to all edges of C_i emerging at ν for every $\nu \in \mathcal{V}_i$. This in turn is ensured by `Edges` due to Lemmas 2.13 and 2.16. However, let us note that every edge connects two vertices and Step 2.1. of `ExploreVertex` skips a given edge if and only if $c \geq 2$, i.e. if and only if the edge has been identified twice – or framed differently: if both vertices have been identified before.

If $K > 0$ then Step 2.2.2. of `WalkRay` will identify if the boundary is crossed by the `RayCast` and will first adjust the vertex to the boundary and also store this boundary implicitly as an additional generator node for future calculations in the current cell. Lemma 3.1 tells us that we are then back in the first case of the proof. Due to the localized nature of the calculations, the proof is complete. □

3.4 Descent and Raycast algorithms

We will now get to the heart of the mesh generator: the Raycast algorithm. It is fully based on the algorithm outlined in Lemma 2.9 and hence it converges.

Algorithm 3.8. `RayCast`$(\mathcal{M}, \boldsymbol{r}, \mathbf{n}, i)$

1. chose z_1 as nearest neighbor of $\nu_1 = \nu$ with $(z_1 - x_i) \cdot \mathbf{n} > 0$
2. Given $k \geq 1$ search $r_k = \nu_k + t_k \mathbf{n}$ through the equation

   $$|r_k - x_i|^2 = |z_k - r_k|^2 \qquad \Leftrightarrow \qquad |\nu_k - x_i|^2 - 2t_k \nu \cdot (\nu_k - x_i) = |\nu_k - z_k|^2 - 2t_k \nu \cdot (\nu_k - z_k)$$

   which ultimately yields t_k.
3. if $t_k = \infty$ return $(\boldsymbol{r}, \mathbf{n})$.
4. otherwise set $\nu_{k+1} = r_k$ and search for its nearest neighbor z_{k+1} which lies within a ball of radius $|r_k - z_k|$. If $|r_k - z_k| = |r_k - z_{k+1}|$ we return $(r_k, \mathcal{N}(r_k))$ otherwise we continue with Step 2. for $k + 1$.

The `Descent` algorithm is almost an implementation of the proof of Lemma 2.11 and relies on the `RayCast` algorithm. However, we still need to show that it converges.

Algorithm 3.9. `Descent`$(\mathcal{M}, i)$

1. let $y_1 = x_i$ and $Y_1 = \{y_1\}$ as well as $\nu_1 = y_1 = x_i$
2. for $j = 1, \ldots d$ do the following:

   2.1. Given $Y_j = \{y_1, \ldots, y_j\}$ let $\mathbf{n}_j \in \mathbb{S}^{d-1}$ be a random unit vector orthogonal to the $j - 1$ vectors $\{(y_1 - y_2), \ldots, (y_1 - y_j)\}$.

   2.2. call `RayCast`$(\mathcal{M}, \nu_j, \mathbf{n}_j, i)$.

   2.3. If `RayCast` successfully returns $(r, \mathcal{N}(r))$ set $Y_{j+1} = \mathcal{N}(r)$ and $\nu_{j+1} = r$.

   2.4. else call `RayCast`$(\mathcal{M}, \nu_j, -\mathbf{n}_j, i)$, take the result $(r, \mathcal{N}(r))$ and set $Y_{j+1} = \mathcal{N}(r)$ and $\nu_{j+1} = r$.

Lemma 3.3. *The* `Descent` *algorithm converges almost surely.*

Remark. Lemma 3.3 is to be read as: Even though there is some randomness involved, the probability that `Descent(...)` fails is zero.

Proof. The Descent Lemma 2.11 and its proof ensure that for almost every ν_k lying in a $d-k+1$ facet of C_i the descent Steps 2.2.–2.4. will almost surely yield ν_{k+1} in a $d-k$-facet. Thus iterating the algorithm finally yields a 0-facet, i.e. a vertex of C_i. □

3.5 Computational complexity of the algorithm

Theorem 3.4. *Let $X = (x_i)_{i=1,\dots,N}$ be a set of nodes in $\mathbb{R}^d$ generating a Voronoi diagram with E edges. Assume that the average complexity to calculate the nearest neighbor is $\mathbb{NN}(X)$. Then the algorithm* `Voronoi`*$(X, (P_i)_{i=1,\dots,K})$ has computational complexity $O(W\,\mathbb{NN}(X))$.*

Proof. This is a direct consequence of the fact that we have to perform `RayCast` at most once along each edge, every `RayCast` is associated with a finite number of nearest neighbor searches and all other operations are bounded by a constant for each edge. □

Theorem 3.5. *Let $X = (x_i)_{i=1,\dots,N}$ be a set of nodes in $\mathbb{R}^d$ such that the resulting Voronoi diagram is regular, i.e. every vertex is given by exactly $d+1$ nodes. Then the algorithm* `Voronoi`*$(X, (P_i)_{i=1,\dots,K})$ has computational complexity $O(N \ln N)$.*

Proof. The number of vertices is proportional to the number of nodes N. The number of edges is proportional to the number of vertices and the computational complexity of nearest neighbor searches is proportional to $O(\ln N)$, e.g. with a kd-tree [2, 8]. This gives the claimed computational complexity. □

Remark. For a Voronoi diagram with nodes in non-general position, one would at a first glance expect $O(n^{2-\frac{1}{d}})$ performance. This is because for each newly calculated vertex we have to perform an inrange search to find all generating nodes. An inrange search in a k-d tree has a computational cost of worst case $O(n^{1-\frac{1}{d}})$ [11]. However, numerical experiments suggest that the inrange search is of minor influence, see Table 4. Instead we will see that for nodes of a (quasi) periodic grid, the computational effort to find the nearest neighbor approaches N. This probably is because nodes are oriented along flat interfaces, rendering the partitioning of space by flat surfaces – as done by the KD-Tree algorithm – useless. However, the upper cost for a nearest neighbor search is N and there is still hope for more fittet nearest neighbor searches in the future.

4 Mesh refinement

Suppose we are given a fully computed Voronoi diagram $\mathcal{M} = (X, \mathcal{O}, \mathcal{A}, \mathcal{B}, \mathcal{B}^\infty)$ and want to add some additional points $Z \subset \Omega$. This is done by the following algorithm

Algorithm 4.1. `Refine`$(\mathcal{M}, Z)$

1 Let $N_z := \#Z$.

2 For every $i \in \{1, \dots, N\}$ and every $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{A}_i \cup \mathcal{B}_i$ do: Replace $\boldsymbol{\sigma} = (\sigma_1, \dots, \sigma_J)$ by $\boldsymbol{\sigma} = (\sigma_1 + N_z, \dots, \sigma_J + N_z)$

3 Prepend Z to X, that is: define $\tilde{X} = Z \cup X$ where

$$\tilde{X} = \{z_1, \dots, z_{N_z}, x_1, \dots, x_N\}\,, \quad \text{i.e.}$$
$$\forall i = 1, \dots, N_z : \ \tilde{x}_i = z_i\,, \quad \forall i = N_z + 1, \dots, N_z + N : \ \tilde{x}_i = x_{i-N_z}\,.$$

4 For $i = 1, \dots N_z$ set $\tilde{\mathcal{A}}_i, \tilde{\mathcal{B}}_i$ as empty sets and for $i = N_z + 1, \dots, N_z + N$ set $\tilde{\mathcal{A}}_i = \mathcal{A}_{i-N_z}$ and $\tilde{\mathcal{B}}_i = \mathcal{B}_{i-N_z}$. Furthermore, set $\tilde{\mathcal{B}}^\infty$ as an empty set. Then set

$$\tilde{\mathcal{M}} = (\tilde{X}, \mathcal{O}, \tilde{\mathcal{A}}, \tilde{\mathcal{B}}, \tilde{\mathcal{B}}^\infty)$$

5 For $i = 1, \dots, N_z$ call `ExploreCell`($\tilde{\mathcal{M}}, i$)

6 Call `Affected`($\tilde{\mathcal{M}}, N_z$) to get the set of indices A of affected nodes

7 For every $i = N_z + 1, \dots, N_z + N$ do the following:
For every $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{A}_i$ such that $\boldsymbol{\sigma} \subset A \cup \{N_z + N + 1, \dots, N_z + N + K\}$ delete $(\boldsymbol{\sigma}, \boldsymbol{r})$ from $\mathcal{A}_i$ and from every $\mathcal{B}_j$, $j > i$.

8 For $i = N_z + 1, \dots, N_z + N$ call `ExploreCell`($\tilde{\mathcal{M}}, i$)

9 For every element $(\boldsymbol{r}, \mathbf{u})$ of $\mathcal{B}^\infty$ check if $\boldsymbol{r}$ is still a valid vertex and $\mathbf{u}$ is still a valid direction. If so, copy to $\tilde{\mathcal{B}}^\infty$

Algorithm 4.2. `Affected`($\tilde{\mathcal{M}}, n_z$)

1 create empty list A of integer indices

2 for $j \in 1, \dots, N_z$ and for every $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{A}_j$ do: If $\boldsymbol{\sigma} = (\sigma_1, \dots \sigma_K)$ store every $\sigma_k > N_z$ to A

3 return A

Theorem 4.1. *For every fully calculated mesh $\mathcal{M}$ and every set of points $Z \subset \Omega \setminus X$ the* `Refine` *algorithm converges to a complete mesh, i.e. in view of Theorem 3.2 if X contains d linear independent nodes, then almost surely for ever $x_i \in \tilde{X}$ all vertices of $\tilde{C}_i$ will be found, where $\tilde{C}_i = C_i \cap \overline{\Omega}$ is defined in Lemma 3.1.*

Proof. Steps 1.–4. only modify existing data in way as to increase the index number of every single node $x \in X$ by n_z. Accordingly, also the indices of the boundary planes and the nodes $x_j^{(i)}$ are increased.

In Step 5 the *new* Voronoi mesh is calculated but only the cells of the first N_z nodes. Following the outline of the proof of Theorem 3.2 this is completely independent from any result calculated by `ExploreCell`($\tilde{\mathcal{M}}, i$) for $i > N_z$.

In Step 6 we determine which of the „old cells" calculated and stored in $\mathcal{M}$ are affected by the inclusion of Z. However, an old cell C_j, $j > N_z$ is affected by the new nodes if and only if one of the cells C_i, $i \le N_z$ has a vertex $(\boldsymbol{\sigma}, \boldsymbol{r}) \in \mathcal{V}_i$ with $j \in \boldsymbol{\sigma}$.

In Step 7 we make use of the fact that an „old" vertex can be affected by the new nodes only if all of its adjacent nodes are affected. Hence we delete all such vertices and keep the others.

In Step 8 we use once more `ExploreCell`($\tilde{\mathcal{M}}, i$) for all $i > N_z$ which will make use of all previously calculated, non-deleted vertices and complete the mesh as pointed out in the proof of Theorem 3.2.

In Step 9 we make sure to copy only reasonable elements of $\mathcal{B}^\infty$ to the new mesh. □

5 Fast quasi-periodic mesh generation

We will now come to a particular feature of our data structure: We can take N_0 nodes $X_0 = (x_i)_{i=1,\dots,N_0}$ within a cuboid $Q \subset \mathbb{R}^d$ oriented along the standard axes and declare a multiplication array $A = [a_1, \dots, a_d]$. If the origin of Q is at 0 and Q has width q_i in dimension i we associate with each $B = [b_1, \dots, b_d]$ with $1 \leq b_i \leq a_i$ a cube $Q_B = Q + ((b_1 - 1)q_1, \dots, (b_d - 1)q_d)$ and copy a version of X_0 into Q_B and call it X_B. The union of all X_B is our new X for which we want to calculate the Voronoi Diagram inside $\bigcup_B Q_B$.

If $B = [b_1, \dots, b_d]$ satisfies for some i that $2 < b_i < a_i$ then we can define $\tilde{B} = [b_1, \dots, b_{i-1}, b_i - 1, b_{i+1}, \dots, b_d]$ and copy the vertices of $X_{\tilde{B}}$ directly to X_B, only adjusting the indices of the generators. This also works with the data on volume and interfaces. On the remaining boundary cells, one can copy most of the data from previous steps and only has to take care of intersections with the boundary.

This trick is implemented in `HighVoronoi` [9] and can be called as follows:

```
dim = 5
scalefactor = 4 # how often do I want to repeat the data in each direction,
↪  it fill be fitted into the unit cube.
use_fast_mode = true
VG2 = VoronoiGeometry( VoronoiNodes(rand(dim,2),periodic_grid=(periodic=[],
↪  dimensions=ones(Float64,dim), scale=scalefactor^(-1)*ones(Float64,dim),
↪  repeat=scalefactor*ones(Int64,dim), fast=use_fast_mode),
↪  integrator=HighVoronoi.VI_GEOMETRY)
```

We can provide a rough estimate on the amount of time saved by this approach as follows: set

$$P_0 = 0 \quad \text{and if } a_k > 2 \quad P_k = P_{k-1}\frac{3}{a_k} + \frac{a_k - 3}{a_k} \quad \text{else } P_k = P_{k-1}$$

then P_d is the fraction of data that can be obtained from copying. Some sample values for P_d are given in the following table:

A	[4,4,4,4]	[4,4,4,5]	[4,4,4,10]	[4,4,4,4,4]	[4,4,4,4,4,4]	[4,4,4,4,4,4,4]
P_d	0.683594	0.746875	0.873438	0.762695	0.822021	0.899887

6 Robustness

In our whole algorithm machinery, there are two major sources of error that have proven to cause complications and have to be accounted by internal corrections:

1 Rounding errors in orthogonalization: There are many occasions in the code when a local orthogonal basis has to be calculated, primarily when we iterate through edges. In 5 dimensions it happens quickly that the accuracy is only of order 10^{-8}, which quickly accumulates to massive deviations from the exact positions of new vertices. However, this can be fixed by repeated orthogonalization, which pushes the error below 10^{-15}.

2 Deviation of a computed vertex from its true position: If a computed vertex deviates to much from its true position this causes tremendous problems in case of non-general position generators, because the edge criterion in Lemma 2.16 is very sensitive to the relative positions of vertex and generators.

However, a good criterion turns out to be the relative total variation of the position of vertices: For each vertex $(\boldsymbol{\sigma}, \boldsymbol{r})$ one may calculate the mean squared distance $d_0^2 := |\boldsymbol{\sigma}|^{-1} \sum_{\sigma_i \in \boldsymbol{\sigma}} |x_{\sigma_i} - \boldsymbol{r}|^2$ and the relative variation: $v := d_0^{-2} \sum_{\sigma_i \in \boldsymbol{\sigma}} (|x_{\sigma_i} - \boldsymbol{r}|^2 - d_0^2)^2$. By this we avoid the costly calculation of squareroots. If the later value is to large, one may correct the position: Note that $\boldsymbol{\sigma}$ has a least $d+1$ generators. Using a sophisticated selection we set up the invertible linear system

$$|x_i - \boldsymbol{r}|^2 = |x_1 - \boldsymbol{r}|^2 \qquad \Leftrightarrow \qquad |x_i|^2 - |x_1|^2 = 2(x_i - x_1) \cdot \boldsymbol{r}\,, \tag{7}$$

and use our current candidate for $\boldsymbol{r}$ as initial guess. This correction, if necessary, can reduce the error significantly. In the `HighVoronoi` package, the distances of the generators coincide with a relative error of less than 10^{-12}, the relative total variation, e.g. in 5D, being often less than 10^{-30}.

6.1 Probability for a fraud vertex

In what follows, a *fraud vertex* is given by $\tilde{\nu} = (\tilde{\boldsymbol{\sigma}}, \tilde{\boldsymbol{r}})$ where $\tilde{\boldsymbol{\sigma}}$ will not be a valid set of indices that could represents a true vertex. That is, we leave aside the question whether $\tilde{\boldsymbol{r}}$ has sufficient accuracy, as this question was handled at the beginning of this whole section.

Assume we are given the domain $\Omega = (0,1)^d$, i.e. the unit cube, with N points distributed randomly and uniformly. Given the vertex $\nu = (\boldsymbol{\sigma}, \boldsymbol{r})$ we denote $X_{\boldsymbol{\sigma}} = (x_{\sigma_1}, \ldots, x_{\sigma_k})$ with $R_0 = |x_{\sigma_1} - \boldsymbol{r}|$. In order for a fraud vertex to occur, the necessary condition is that there exists $x \in X \setminus X_{\boldsymbol{\sigma}}$ that is close enough to $\boldsymbol{r}$ so the algorithm will consider it as a generator of ν. By definition of a vertex, the event $(X \setminus X_{\boldsymbol{\sigma}}) \cap \overline{\mathbb{B}_{R_0}(\boldsymbol{r})} = \emptyset$ has probability 1 and thus the probability of fraud vertex amounts to the probability for some small $\varepsilon > 0$ that

$$(X \setminus X_{\boldsymbol{\sigma}}) \cap \left(\mathbb{B}_{R_0(1+\varepsilon)}(\boldsymbol{r}) \setminus \overline{\mathbb{B}_{R_0}(\boldsymbol{r})} \right) \neq \emptyset$$

We introduce the notation $\mathbb{S}_{\varepsilon,R_0}(\nu) := \mathbb{B}_{R_0(1+\varepsilon)}(\boldsymbol{r}) \setminus \overline{\mathbb{B}_{R_0}(\boldsymbol{r})}$ and imply that on average $R_0 < \sqrt[d]{N^{-1}}$. Given S_{d-1} the area of the unit sphere, we obtain the estimate

$$V_0 = |\mathbb{S}_{\varepsilon,R_0}(\nu)| = \left| \mathbb{B}_{R_0(1+\varepsilon)}(\boldsymbol{r}) \setminus \overline{\mathbb{B}_{R_0}(\boldsymbol{r})} \right| < R_0^{d-1} S_{d-1}\, R_0 \varepsilon < S_{d-1} N^{-1} \varepsilon\,.$$

This value is by order 0 proportional to the probability of any $x \in X \setminus \boldsymbol{\sigma}$ to lie in $\mathbb{S}_{\varepsilon,R_0}(\nu)$. The probability for none of these is thus larger than

$$(1-V_0)^{N-d-1} > (1-V_0)^N = \exp(N \ln(1-V_0)) \approx \exp(-N\, V_0) > \exp(-S_{d-1}\varepsilon)\,.$$

While this implies that the probability to find a wrong $x \in \tilde{\boldsymbol{\sigma}}$ is of the order $1-(1-V_0)^{N-d-1} \approx S_{d-1}\varepsilon$ and hence rather small, the probability to find such an error in at least one vertex of the full Voronoi diagram is rather high: Given $\#\mathcal{V}$ the number of vertices we find the probability of non-failure to be

$$(1-V_0)^{(N-d-1)\#\mathcal{V}} \gtrsim \exp(-S_{d-1}\#\mathcal{V}\varepsilon)\,. \tag{8}$$

6.2 Probability for a fraud edge or fraud interface

We set aside S_{d-1} in formula 8 as this value has its maximum in $d = 8$ and afterwards decreases with d. We find that the failure probability for $\#\mathcal{V}$ vertices with accuracy ε is bounded from above essentially

by $\#\mathcal{V}\varepsilon$. We have seen that this is somewhat the probability that there is at least one fraudulent vertex. The probability that there is a whole fraudulent edge, e.g. an edge formed by two fraudulent vertices that both share a given $x \in X$ even though they should not, is the product of the probability that each of these vertices if fraudulent. It is hence bounded from above by $(S_{d-1}\varepsilon)^2$.

In a grid of nodes of general position, there are d-times more edges than vertices and we conclude that the probability of a fraudulent edge is smaller than $d\, S_{d-1}\#\mathcal{V}\varepsilon^2$. Similar conclusions hold for fraudulent interfaces, which decrease with $\#\mathcal{V}\varepsilon^d$.

7 Implementation and performance tests

The above algorithms are implemented as parts of the Julia package `HighVoronoi.jl` by the author [9]. On top of the sole calculation of the vertices it implements mesh refinement, substitution of prescribed area, prescribed distribution of generators, boundaries and periodic meshes. Further it provides algorithms for volume and area as well as for integration of functions over cells and interfaces and it can pass these to a Finite Volume model generator that can account for Dirichlet-, Neumann- and periodic boundary conditions.

7.1 Vertices from generators in non-general position

We will discuss the effects and benefits of generators in non-general position. First we will discuss the benefits in Subsection 7.1.1. Then we will illustrate our algorithm from Lemma 2.16 that allows to effectively iterate the edges in vertices of non-general position.

7.1.1 Benefits

The most evident benefit of a cubic mesh over a general Voronoi mesh is that every cell has precisely $2d$ neighbors and hence the matrices in Finite Volume (equivalent with finite difference in this case) are extremely sparse. On the other side of the spectrum, Voronoi meshes generated from i.i.d distributed nodes have a very high average number of neighbors in high dimensions, see Table 1. In 7 dimensions, the number of neighbors goes up to a factor of 35 compared to the cubic grid. However, if the algorithm is capable to handle non-general position generators, we can e.g. compute a quasi periodic grid, where two nodes are placed in a unit cell which is then copied in each dimension for a given number of repetition. This results in much lower amount of neighbors per cell, see Table 2, where the factor from a cubic grid in 7D is now in between 2 and 4.

The approach also allows us in principle to generate a fully cubic grid and to refine it locally. Computational effort is then spent only in the places where a high resolution is needed while the rest of the grid is handled in a very fast way.

Finally, it is possible to compute large quasi-periodic grids using copy-and-paste subroutines. This is particularly interesting if one is not only interested in the Voronoi diagram but also in volumes and interface areas, such as needed in Finite Volume methods. Figure 6 shows the performance gain for this fast copy-and-paste algorithm vs. computation for a full mesh of nodes in general position. Also it shows the time vs. number of nodes when increasing a quasi-periodic mesh.

7.1.2 Illustration of Lemma 2.16

We start with a periodic mesh in 5D generated from two nodes by the same principle as in the previous subsection. In the resulting mesh, the first two nodes are $x_1 = (0.0475,\ 0.225,\ 0.01,\ 0.205,\ 0.0425)$ and $x_2 = (0.1475,\ 0.1375,\ 0.245,\ 0.0875,\ 0.085)$. We pick out a vertex which has 10 generators and apply a shift transformation of coordinates such that $x_1 = 0$. In these local coordinates we have the following generators:

node	local coordinates	node	local coordinates	node	local coordinates
9	[0.0, 0.25, 0.0, 0.0, 0.0]	33	[0.0, 0.0, 0.25, 0.0, 0.0]	41	[0.0, 0.25, 0.25, 0.0, 0.0]
138	[0.1, 0.1625, 0.235, 0.1325, 0.0425]	513	[0.0, 0.0, 0.0, 0.0, 0.25]	521	[0.0, 0.25, 0.0, 0.0, 0.25]
545	[0.0, 0.0, 0.25, 0.0, 0.25]	553	[0.0, 0.25, 0.25, 0.0, 0.25]	2050	[-0.095, 0.0, 0.0, 0.0, 0.0]

Note that 2050 is a boundary node, i.e. a mirrored version of x_1. By their odd number we can conclude that all other nodes except node 138 are periodic copies of x_1. We will now follow the algorithm how it will identify the essential generators of the vertex from the point of view of x_1.

Step I: The algorithm will detect that x_1, x_9, x_{33}, x_{41}, x_{513}, x_{521}, x_{545}, x_{553}, x_{2050} lie in one plane. It will pass this to the lower dimensional algorithm in 4D

Step I.a: Identify the $d-2=3$ dimensional plane x_1, x_{33}, x_{513}, x_{545}, x_{2050}

Step I.a.1: The 2-dimensional plane x_1, x_{33}, x_{513}, x_{545} will lead to kicking out x_{545}. x_{33}, x_{513} are identified as part of $\tilde{X}_2$.

Step I.a.2: From x_1, x_{33}, x_{2050} we will identify in total $\tilde{X}_3$=$x_1, x_{33}, x_{513}, x_{2050}$.

Step I.b: Next is the 3 dimensional plane x_1, x_9, x_{33}, x_{41}, x_{2050}

Step I.b.1: The 2 dimensional plane x_1, x_9, x_{33}, x_{41} leads to kicking out x_{41}.

Step II: The points x_1, x_9, x_{33}, x_{41}, x_{138}, x_{513}, x_{2050} all lie on another 4 dimensional plane

Step II.a: 3 dimensional plane x_1, x_9, x_{33}, x_{41}, x_{138}

Step II.a.1: The 2 dimensional plane x_1, x_{33}, x_{41}, x_{138} leads to removal of x_{41} and acceptance of x_{138}.

Hence, the algorithm terminates with essential generators x_9, x_{33}, x_{138}, x_{513} and x_{2050}. The resulting edges can be computed to be the following:

generators	orientation
[1, 9, 33, 41, 138, 513, 521, 545, 553]	[0.7981891503332307, 0.0, 0.0, -0.6024069059118722, 0.0]
[1, 9, 33, 41, 138, 2050]	[0.0, 0.0, 0.0, 0.3054275543593127, -0.952215316531975]
[1, 9, 33, 41, 513, 521, 545, 553, 2050]	[0.0, 0.0, 0.0, -1.0, 0.0]
[1, 9, 138, 513, 521, 2050]	[0.0, 0.0, -0.4911409242478349, 0.871080129798047, 0.0]
[1, 33, 138, 513, 545, 2050]	[0.0, -0.6319383146726604, 0.0, 0.7750186878060926, 0.0]

7.2 Performance tests: code

The tests are run using the default test implemented in the package and plotted with `Plots.jl`. Fitting of data to $N \mapsto c_0 + c_1 N + c_2 N \ln N$ is done using `DataFitting.jl`. This yields the following code.

```
using HighVoronoi
using Plots
using DataFitting
using SpecialFunctions
```

```
### Run performance test and store data in `A`
nodeslist = [200,500,1000,1500,2000,3000,4000,6000,8000,10000,12500,15000,
             17500,20000,22500,25000,27500,30000]
 dim = 5
 #performance test in R^dim for nodes[...] iid distributed points and
 ↪  averages data over 4 independent calculations
 #returns values for further evaluation and also stores them in file
 A = HighVoronoi.collect_statistics(
 ↪  HighVoronoi.statistic_samples(dim,nodeslist,4),
 ↪  txt="results$(dim)D-30000-new.txt")

#try the following to make a performance test on a periodic mesh for
↪  various amounts of repetitions of the cell.
# It creates the periodic grid from two random points and creates several
↪  samples from 2 to 10 repetitions in each direction.
# A = HighVoronoi.collect_statistics( rand(dim,2), dim, 2*ones(Int64,dim),
↪  10*ones(Int64,dim), txt="resultsper$(dim)D-8000-new.txt")


### Fit N*ln(N) model to data
f(x, p1, p2, p3 ) = @. (p1  +  p2 * x  +  p3 * x * log(x)   )
# if you do not trust in N*ln(N) you may also try polynomial fit:
# f(x, p1, p2, p3 ) = @. (p1  +  p2 * x  +  p3 * x^2   )

dom = Domain(A[1,:])
data = Measures(A[3,:],1.0)

params = [1.0,1.0,1.0]
model1 = Model(:comp1 => FuncWrap(f, params...))
prepare!(model1, dom, :comp1)
result1 = fit!(model1, data)

### Plot test results
plot(A[1,:], A[3,:], color=:blue, label="nodes vs time")

### Plot fittet N*ln(N)
my_p1 = result1.param[:comp1__p1].val
my_p2 = result1.param[:comp1__p2].val
my_p3 = result1.param[:comp1__p3].val
fitted_f(x) = my_p1  +  my_p2 * x  +  my_p3 *x * log(x)
#cubic verion:
#fitted_f(x) = my_p1  +  my_p2 * x  +  my_p3 *x * x


oRound(x) = round(x, digits = 3 - floor(Int64,log10(abs(x))))
plot!(x->fitted_f(x), color=:red, label = """f(x)=$(oRound(my_p1))  +
↪  $(oRound(my_p2)) * x  +  $(oRound(my_p3)) * x * log(x)""")
#or
#plot!(x->fitted_f(x), color=:red, label = """f(x)=$(oRound(my_p1))  +
↪  $(oRound(my_p2)) * x  +  $(oRound(my_p3)) * x^2""")

savefig("plot$(dim)D.pdf")

#print out some measure for relative error. Start from third entry as
↪  deviations ar extremely high for small number of nodes...
total_error = sum( map( i-> ((A[3,i]-fitted_f(A[1,i]))/(A[3,i]))^2,
↪  3:length(A[1,:])) )
```

	4D	5D	6D
$O(N^2)$ fit	0.0193359	0.0789179	0.0706837
$O(N \ln N)$ fit	0.0115859	0.0140203	0.0272475

Table 3: The relative deviation of the measured data vs. the fitted curve for a quadratic or a $N \ln N$ curve. It can be seen that the error is smaller for the $N \ln N$, implying that the complexity follows this type of law.

```
println("total error: ",total_error)
```

Explanations of the code are given as comments. The performance data that was obtained was fitted against a N^2 and a $N \ln N$ growth of time vs. nodes. If $A(N)$ is the measured time for a computation of N generators and $f(N)$ is the fitted curve, then our measure for the error when sampling over K different numbers of nodes is

$$E = \sum_{k=3}^{K} \frac{(f(N) - A(N))^2}{A(N)^2} \,. \tag{9}$$

We start at $k = 3$ since the error at 1 and 2 turned out to be significantly dominant in this metric.

7.3 Performance tests: results

As for the matrix `A` in the above simulations, we obtain results in 4 and 5 dimensions on a conventional notebook and in 6 dimensions on a stationary pc. The `collect_statistics` method yields additional data on the internal flow of the algorithm, but we restrict here to „nodes vs. time". For more information, we refer to the manual [9].

7.3.1 Nodes in general position

Sample computations were performed on i.i.d. distributed Nodes and data was collected from the following number of nodes: 200, 500, 1000, 1500, 2000, 3000, 4000, 6000, 8000, 10000, 12500, 15000, 17500, 20000, 22500, 25000, 27500, and 30000. The collected performance data are shown in Figures 4 and 5 together with an $O(N \ln N)$ fit. However, as this may not be convincing enough, also a quadratic fit was performed, which is not displayed as graph but rather we collect the error according to (9).

The results of the error analysis is summarized in Table 3 and demonstrate that the $O(N \ln N)$ fits much better than the quadratic curve.

7.3.2 Nodes in non-general position: classical mode

A test was performed on quasi-periodic sets of points generated from the identical pair of nodes, copied k_i-times in dimension i. This was done using the following call.

```
dim = 5
A = HighVoronoi.collect_statistics( rand(dim,2), dim, 2*ones(Int64,dim),
↪  10*ones(Int64,dim),txt="resultsper$(dim)D-8000-new.txt")
```

In total, we obtain 43 samples for the following number of nodes: 64, 96, 144, 216, 324, 486, 648, 864, 1152, 1536, 2048, 2560, 3200, 4000, 5000, 6250, 7500, 9000, 10800, 12960, 15552, 18144, 21168,

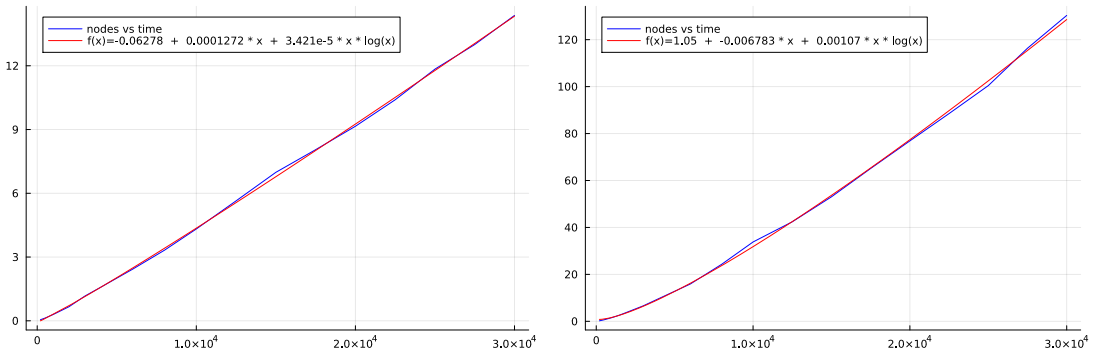

Figure 4: The costs to calculate the Voronoi diagram in 4 (left) and 5 (right) dimensions, with the time being given in seconds. The blue curve provides the actual costs averaged over 4 samples while the red curve provides an interpolation of the form $N + N \ln N$.

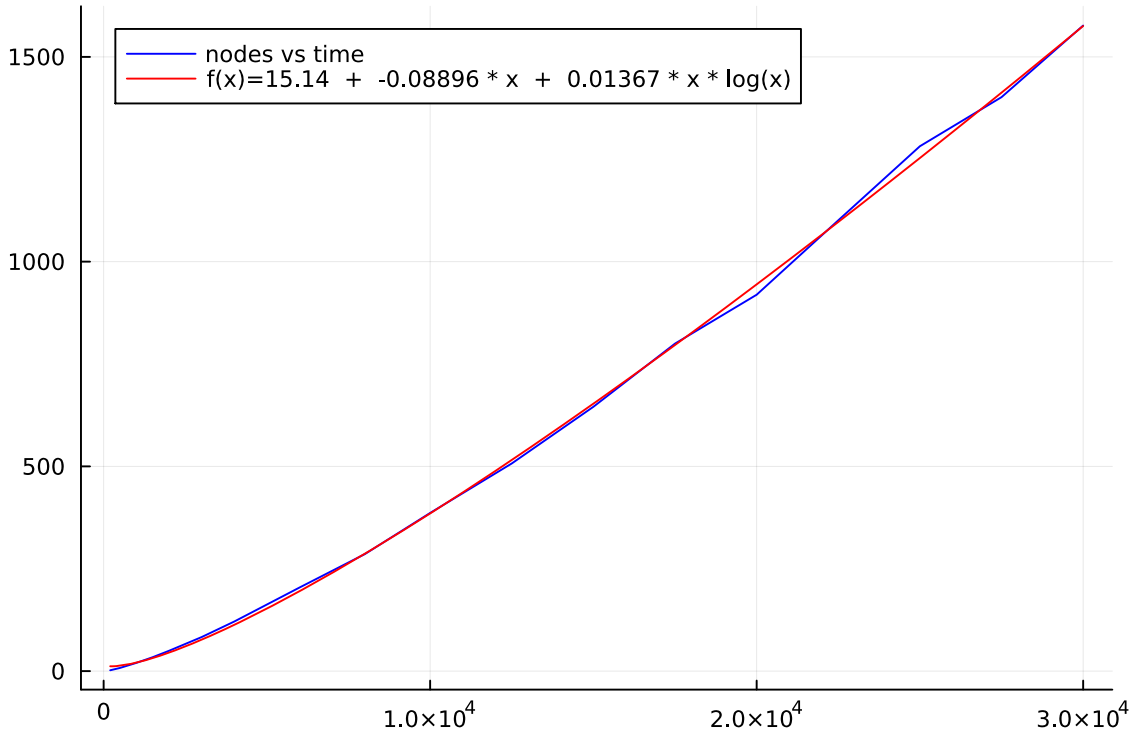

Figure 5: The costs to calculate the Voronoi diagram in 6 dimensions, with the time being given in seconds. The blue curve provides the actual costs averaged over 4 samples while the red curve provides an interpolation of the form $N + N \ln N$.

24696, 28812, 33614, 38416, 43904, 50176, 57344, 65536, 73728, 82944, 93312, 104976, 118098, 131220, 145800, 162000, 180000, 200000.

Unfortunately, Figure 6 shows that the performance in this case follows rather a quadratic curve than a $N \ln N$ curve, which is confirmed by the error expression introduced in (9). We first note that there are several reasons that can cause this behavior:

- In case there are many vertices with generators in non-general position, lots of `inrange` searches are needed. These are known to have a worst-case performance of $O(N^{1-1/d})$.
- There is a hidden operation in the `HighVoronoi` code that goes with N and kicks in only in non-general positions.
- There is an issue with the nearest neighbor search that shows up only in such pathological situations.

For this reason, another test was performed. More precisely, we implemented the following code:

```
using ProfileView

function test_periodic_mesh(dim,nn,f=true,scalefactor=4)
    nodes = VoronoiNodes(round.(rand(dim,nn),digits=2))
    @ProfileView.profview VG2 = VoronoiGeometry( nodes,
     ↪  periodic_grid=(periodic=[], dimensions=ones(Float64,dim),
     ↪  scale=scalefactor^(-1)*ones(Float64,dim),
     ↪  repeat=scalefactor*ones(Int64,dim),fast=f),
     ↪  integrator=HighVoronoi.VI_GEOMETRY, silence=true)
end

cases = [3,4,5,6]
for c in cases
    @time test_periodic_mesh(5,2,false,k)
end
```

The output by `@profview` was then split into the following major tasks by the code: edge identification and iteration, nearest-neighbor search, inrange search, neighbor identification from vertices. The results are given in Table 4 in the form „operation vs. percentage of time”. It has to be noted that we are only interested in the big picture and hence these numbers are not 100% accurate but very close to the truth.

As one can see, the proportion of the nearest neighbor search increases dramatically from 6250 to 15552 nodes. On the other hand, the percentual cost of the `inrange` search remains approximately the same. In the code, the inrange search is called once for every approximately 2.5 nearest neighbor searches but evidently does not significantly contribute to the computational effort.

This results suggest in total that the problem stems from the implementation of the nearest neighbor search. More precisely, the `RayCast` algorithm calls a conditioned nearest neighbor search, which seamingly is not a problem for nodes in general position, but causes a lot of problems in case the nodes are in non-general positions. One problem that `@profview` highlights is that the implementation of the condition itself is not (pre)compiled, a fact that may slow down the Julia code in these bottlenecks significantly. However, a future version of the `NearestNeighbor.jl` package or of Julia itself might solve this issue.

case	nodes	% edge iteration	% nearest-neighbor	% inrange	%neighbor ident.
3	486	74.7	15.8	2.3	6.7
4	2048	66.2	20.4	2.2	6.7
5	6250	60.9	26.7	2.3	5.9
6	15552	31.0	63.8	2.7	2.4

Table 4: Approximate share of the computational costs for different parts of the Voronoi diagram generation in % of total time. The residual percentage is distributed to unspecified iteration and storage related operations.

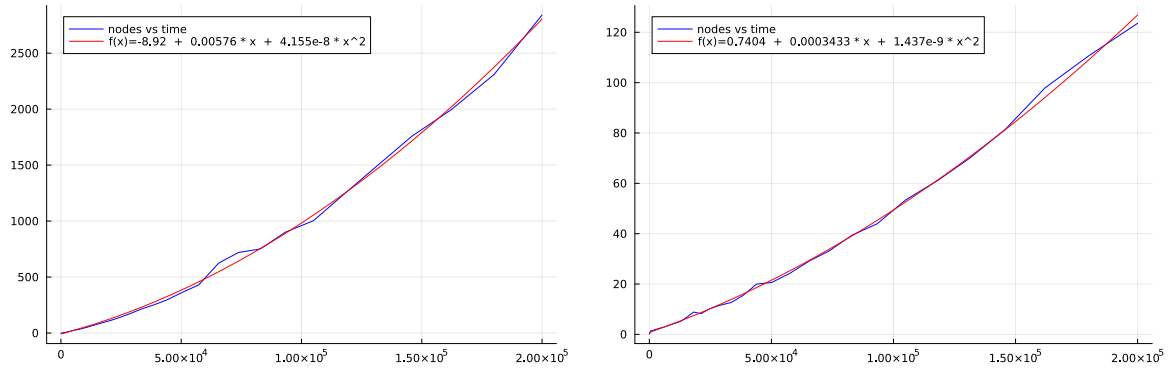

Figure 6: Calculations for a quasi-periodic Voronoi diagram in 5D from 2 nodes that are copied periodically in each dimension, with a vector B ranging from $B = [2, \dots, 2]$ to $B = [10, \dots, 10]$ (see Section 5). In the left we see the classical calculation for each generator without making use of the periodicity. On the right is the performance for a version of the code that heavily exploits the periodic structure.

7.3.3 Nodes in non-general position: fast mode

Finally, we test the fast mesh generation method. The copy method should be linear in the amount of unit cells that are copied, hence somewhat sublinear proportional to N and the rest of the method should grow with $N \ln N$ as discussed above. We test this with the following command.

```
dim = 5
A = HighVoronoi.collect_statistics( rand(dim,2), dim, 2*ones(Int64,dim),
↪  10*ones(Int64,dim), txt="resultsper$(dim)D-8000-new.txt", fast=true)
```

However, with regard to Figure 6 it turns out that we again face a quadratic growth. However, the share of the quadratic growth is mus less even though the total time is already strongly decreased. This is mostly due to the fact that there are only few nearest neighbor searches necessary compared to the total amount of generators or vertices.

8 Conclusions

We have described a new way to calculate Voronoi diagrams for nodes in general and non-general position based on a fully localized algorithm. The algorithm is suitable for local refinement or for benefiting from a periodic mesh structure using copy-modify-paste algorithms. We have demonstrated that the algorithm can be expected to behave as $O(N \ln N)$ under ideal implementation of the nearest-

neighbor search such as i.i.d distributed generators and general $O(E\,\mathbb{NN}(X))$ performance on any diagram.

Open topics for the future is an improvement of the performance on generators in non-general position. A possibility in periodic meshes could be to benefit from the periodicity by setting up a reference configuration and to reduce nearest neighbor search to this reference configuration. In case a periodic mesh is perturbed by a set of additional random nodes, nearest neighbor search could be split into to searches: One on the periodic and one on the random set of nodes.

Another open topic is the Delaunay triangulation based on the High dimensional Voronoi diagram, in particular for generators in non-general position. This seems to be completely open. Maybe there was hope to calculate the Delaunay grid for a perturbed Voronoi Diagram and to match the tetrahedrons afterwards. We leave this question open for future investigations.

References

[1] C. Barber. Qhull documentation. online, 2020.

[2] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] A. Bowyer. Computing dirichlet tessellations. *The computer journal*, 24(2):162–166, 1981.

[4] B. Delaunay et al. Sur la sphere vide. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, 7(793-800):1–2, 1934.

[5] L. Donati, M. Heida, B. G. Keller, and M. Weber. Estimation of the infinitesimal generator by square-root approximation. *Journal of Physics: Condensed Matter*, 30(42):425201, 2018.

[6] L. Donati, M. Weber, and B. G. Keller. Markov models from the square root approximation of the fokker–planck equation: calculating the grid-dependent flux. *Journal of Physics: Condensed Matter*, 33(11):115902, 2021.

[7] R. Eymard, T. Gallouët, and R. Herbin. Finite volume methods. *Handbook of numerical analysis*, 7:713–1018, 2000.

[8] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software (TOMS)*, 3(3):209–226, 1977.

[9] M. Heida. Highvoronoi.jl package. GitHub Repository, August(January) 2023.

[10] M. Laver and E. Sergenti. Modeling multiparty competition. *Party Competition: An Agent-Based Model*, pages 3–14, 2012.

[11] D.-T. Lee and C.-K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, 1977.

[12] V. Polianskii and F. T. Pokorny. Voronoi boundary classification: A high-dimensional geometric approach via weighted monte carlo integration. In *International Conference on Machine Learning*, pages 5162–5170. PMLR, 2019.

[13] V. Polianskii and F. T. Pokorny. Voronoi graph traversal in high dimensions with applications to topological data analysis and piecewise linear interpolation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2154–2164, 2020.

[14] A. Sikorski. Voronoigraph.jl package. GitHub Repository, in 2022.

[15] G. Voronoi. Nouvelles applications des paramètres continus à la théorie des formes quadratiques. premier mémoire. sur quelques propriétés des formes quadratiques positives parfaites. *Journal für die reine und angewandte Mathematik (Crelles Journal)*, 1908(133):97–102, 1908.

[16] D. F. Watson. Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *The computer journal*, 24(2):167–172, 1981.